Docker By Treeptik

Contents

Architecture

Les composants

- Noeuds Master et Slave
- Controllers
- Services
- Pods
- Namespaces
- Network et Policies
- Storage

Noeuds

Noeuds

- La plus petite unité **physique** de calcul
- Peut-être aussi bien : - Une machine bare-metal - Une machine virtuelle dans une datacenter (GKE…) - Une RaspberryPI

Cluster

Mode Déclaratif

- On décrit le **quoi**, pas le **comment** (impératif)
- Kubernetes est un système où l'on définit l'état désiré d'un objet
- Quelles applications doivent être démarrées…
- Sur quels noeuds ses applications doivent être démarrées
- Quelle politique doit être appliqué à ses applications
- Quelles port de mon application doit être utilisé

API SERVER

- Point central de toutes les actions sur le cluster Kubernetes
- Tous les appels internes et externes passent ici
- Toutes les actions sont validés avant d'être exécutés
- Seul composant qui écrit dans la base ETCD
- Processus maître du cluster qui nécessite aussi d'être loadbalancé

KUBE SCHEDULER

- Il détermine quel pod doit tourner sur quel noeud en fonction des ressources
- Capable d'attendre et de retenter l'affectation dans le temps (disponibilité de volumes par exemple)

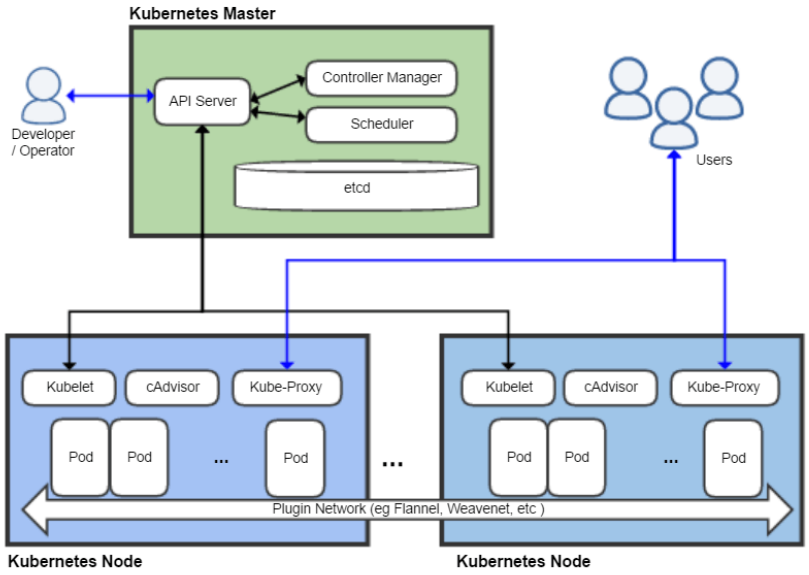


Figure 1: Architecture

KUBE CONTROLLER MANAGER

- Daemon fonctionnant en boucle infini qui interagit avec l'API-SERVER pour déterminer l'état du cluster
- Si l'état n'est pas celui souhaité, ce contrôleur va contacter le contrôleur adéquate
- Il existe de nombreux contrôleurs : EndPoint, Replication… Namespace…

WORKER Node

- Précédemment appelé Minion
- Ce sont les noeuds applicatifs
- Ils contiennent tous une **kubelet** et un **kube-proxy**
- Le **kubelet** pilote Docker ou RKT
- Le **kube-Proxy** est en charge de gérer la connection aux réseaux

---

Kubelet

- Responsable des changements d'état et de la configuration des noeuds WORKER
- Il accepte les requêtes au format YAML ou JSON respectant la spécification PodSpec
- Assure la création et l'accès aux pods pour les objets de type Storage, Secrets et ConfigMaps

---

Autres cas d'usage

- StatefulSets
- Jobs et CronJobs

StatefulSets

- Un StatefulSet ressemble à un ReplicaSet mais avec un lot de fonctionnalités avancées pour gérer les clusters qui nécessitent une certaine continuité en terme d'adressage réseau, de volume et un ordonnancement dans le démarrage des pods.
- Cela permet de déployer dans Kubernetes des applications clusterisées comme des bases de données.

Lancement

3 étapes permettent de créer un StatefulSet : - Création des volumes si nécessaire - Démarrage des éventuels conteneurs init du pod - Démarrage des conteneurs classiques des pods

Kubernetes attend que le readinessProbe soit bon avant de déployer le réplica suivant

---

StatefulSets

- Déploiement de 3 pods avec 3 volumes différentes `simple-statefulset.yaml`

```
kubectl create -f simple-statefulset.yml
kubectl get po,pvc,pv,svc
```

- Nous avons donc 3 volumes différents

```
kubectl exec -it nginx-statefulset-1 /bin/bash
echo '<p>Hello world!</p>' > /usr/share/nginx/html/index.html
exit
```

Exemple d'utilisation (2/2)

Les cycles de vie sont indépendants des pods :

```
kubectl delete -f simple-statefulset.yml
kubectl get pvc,pv
kubectl create -f simple-statefulset.yml
kubectl exec nginx-statefulset-1 cat /usr/share/nginx/html/index.html
```

StatefulSets

DL

Voir le lien d'objectif libre pour la mise en place d'un article ou autre … https://www.objectif-libre.com/fr/blog/2017/08/22/kubernetes-et-galera/

Les jobs

- Les jobs servent à lancer des pods voués à se terminer,
- contrairement aux autres types de contrôleurs (ReplicaSet, StatefulSet, DaemonSet).
- Il existe deux types de jobs : les **Jobs** et les **CronJobs**

Job à usage unique

- Un Job est exécuté qu'une seule fois.
- Il peut être utilisé pour exécuter un batch ou sauvegarder une base de données.
- Créer le fichier `job.yaml`

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```

Les Jobs simples

```
$ kubectl get job
$ kubectl describe job pi

$ pods=$(kubectl get pods  --show-all --selector=job-name=pi --output=jsonpath={.items..meta
$ kubectl logs $pods
```

CronJobs

- Les CronJobs sont des Jobs destinés à être joués à intervalle régulier.
- Création d'un fichier `cronjob.yaml`

```yaml
apiVersion: batch/v2alpha1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
```

```
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

EXERCICE FINAL: WORDPRESS-MYSQL

CNI

Features

---

performances

Consommation ressources

Sécurité

Résumé

Conclusion

Conclusion

Questions / réponses

Les solutions de configuration

- Variables d'environnements
- ConfigMaps
- Secrets

Les variables d'environnement

- Définies en **statique** par défaut dans le Dockerfile
- Surchargées en **dynamique** au runtime via des Containers ou Pod
- Dans notre cas évidemment des pods

Variable d'environnement Pods

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envs-demo
spec:
  containers:
  - name: envs-demo
    image: treeptik/envs-demo
    env:
    - name: SIMPLE_SERVICE_VERSION
      value: "1.0"
```

```
kubectl exec envs-demo -- printenv

PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=envs-demo
SIMPLE_SERVICE_VERSION=1.0
KUBERNETES_PORT_53_UDP_ADDR=172.30.0.1
KUBERNETES_PORT_53_TCP_PORT=53
ROUTER_PORT_443_TCP_PROTO=tcp
DOCKER_REGISTRY_PORT_5000_TCP_ADDR=172.30.1.1
KUBERNETES_SERVICE_PORT_DNS_TCP=53
ROUTER_PORT=tcp://172.30.246.127:80
```

Les configMaps

- Découplage d'une application et de sa configuration
- Portabilité
- But : limiter les variables d'environment
- Existe dans un namespace

Deux configurations globales

```
apiVersion: v1
  kind: ConfigMap
  metadata:
    name: env-config
    namespace: default
  data:
    log_level: INFO

apiVersion: v1
  kind: ConfigMap
  metadata:
    name: special-config
    namespace: default
  data:
    special.how: very
```

Configurations référencées par leurs clés

```
apiVersion: v1
   kind: Pod
   metadata:
     name: dapi-test-pod
   spec:
     containers:
       - name: test-container
         image: k8s.gcr.io/busybox
         command: [ "/bin/sh", "-c", "env" ]
         env:
           - name: SPECIAL_LEVEL_KEY
             valueFrom:
               configMapKeyRef:
                 name: special-config
                 key: special.how
           - name: LOG_LEVEL
```

```
            valueFrom:
              configMapKeyRef:
                name: env-config
                key: log_level
    restartPolicy: Never
```

Classic file

hardcoded.js ~~~ var http = require('http'); var server = http.createServer(function (request, response) { const language = 'English'; const API_KEY = '123-456-789'; response.write(`Language: ${language}\n`); response.write(`API Key: ${API_KEY}\n`); response.end(`\n`); }); server.listen(3000); ~~~

Un peu mieux...

Avec les variables d'environnement

```
var http = require('http');
var server = http.createServer(function (request, response) {
  const language = process.env.LANGUAGE;
  const API_KEY = process.env.API_KEY;
  response.write(`Language: ${language}\n`);
  response.write(`API Key: ${API_KEY}\n`);
  response.end(`\n`);
});
server.listen(3000);
```

Avec Docker ça va encore mieux

A la construction ~~~ FROM node:6-onbuild EXPOSE 3000 ENV LANGUAGE English ENV API_KEY 123-456-789 ~~~

Et même au lancement ~~~ docker run -e LANGUAGE=Spanish -e API_KEY=09876
-p 3000:3000
-ti envtest ~~~

Avec Kubernetes on s'approche encore du mieux

```
apiVersion: extensions/v1beta1
kind: Deployment
```

```
metadata:
  name: envtest
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: envtest
    spec:
      containers:
      - name: envtest
        image: gcr.io/<PROJECT_ID>/envtest
        ports:
        - containerPort: 3000
        env:
        - name: LANGUAGE
          value: "English"
        - name: API_KEY
          value: "123-456-789"
```

Création de notre première ConfigMaps

```
kubectl create configmap language --from-literal=LANGUAGE=English

kubectl get configmap
NAME      DATA    AGE
language  1       1m
```

Référencer la ConfigMap lors du déploiement

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: envtest
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: envtest
    spec:
      containers:
      - name: envtest
```

```
        image: gcr.io/nicolas/envtest
        env:
        - name: LANGUAGE
          valueFrom:
            configMapKeyRef:
              name: language
              key: LANGUAGE
```

Mettre à jour la configuration

```
kubectl create configmap language --from-literal=LANGUAGE=Spanish \
        -o yaml --dry-run | kubectl replace -f -
```

Allons plus loin avec les fichiers

```
mkdir config
echo '{"LANGUAGE":"English"}' > ./config/config
```

Puis mettons à jour le fichier : ~~Javascript var http = require('http'); var fs = require('fs'); var server = http.createServer(function (request, response) { fs.readFile('./config/config.json', function (err, config) { if (err) return console.log(err); const language = JSON.parse(config).LANGUAGE; fs.readFile('./secret/secret.json', function (err, secret) { if (err) return console.log(err); const API_KEY = JSON.parse(secret).API_KEY; response.write(Language: ${language}\n response.write(API Key: ${API_KEY}\n); response.end(\n); }); }); }); server.listen(3000);~~

Avec Docker, nous faisons

```
docker run -p 3000:3000 -ti \
  -v $(pwd)/config/:/usr/src/app/config/ \
  envtest
```

Avec K8S, nous faisons également

```
kubectl create configmap my-config --from-file=./config/config.json

kubectl get configmap
NAME        DATA        AGE
my-config   1           56s
```

Utiliser ceci dans le Deployment

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: envtest
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: envtest
    spec:
      containers:
      - name: envtest
        image: gcr.io/smart-spark-93622/envtest:file5
        ports:
        - containerPort: 3000
        volumeMounts:
          - name: my-config
            mountPath: /usr/src/app/config
      volumes:
      - name: my-config
        configMap:
          name: my-config
```

MISE A JOUR SANS DOWNTIME

- Mise à jour dynamique des volumes
- Attention à l'inconsistence possible si plusieurs containers
- Si problème sévère lié à l'inconsistence, ne pas utiliser et tout recréer dont le Deployment

```
echo '{"LANGUAGE":"Klingon"}' > ./config/config.json
kubectl create configmap my-config \
  --from-file=./config/config.json \
  -o yaml --dry-run | kubectl replace -f -
```

EXERCICE CONFIGMAPS

Les secrets

Bonnes pratiques

Ne jamais coder de mots de passe dans des fichiers ou images

Objectifs

- Protéger les données sensibles
- Dissocier les consommateurs / producteurs via une API

Utilisation des Secrets

- Ils sont définis dans un **namespace** donné
- Deux types d'accès : **volume** ou **variable d'environnement**
- Les secrets sont stockés dans un **tmpfs**
- Limite de 1 Mo
- L'API Server stocke le secret en *plaintext* dans ETCD

Différents types de Secrets

- Generic
- Docker-Registry
- TLS

Cas d'usage Generic

EXERCICE SECRETS

Pour aller plus loin

```
kubectl create secret \
                generic couple \
                --from-literal=user=nicolas \
                --from-literal=password=hk6504q
```

```
kubectl get secrets
```

```
NAME                  TYPE                                  DATA      AGE
chanson               Opaque                                 1         1h
couple                Opaque                                 2         19s
default-token-l7tvd   kubernetes.io/service-account-token   3         5d
```

---

describe

```
kubectl describe secrets couple
```

```
Name:         couple
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
user:      7 bytes
password:  7 bytes
```

Toutes les informations dont celle en base64

```
kubectl get secrets couple -o yaml
```

```yaml
apiVersion: v1
data:
  password: aGs2NTA0cQ==
  user: bmljb2xhcw==
kind: Secret
metadata:
  creationTimestamp: 2018-06-18T13:07:38Z
  name: couple
  namespace: default
  resourceVersion: "735530"
  selfLink: /api/v1/namespaces/default/secrets/couple
```

```
  uid: 936632d1-72f8-11e8-9d8f-7a2fde66199e
type: Opaque
```

Injection manuelle de Secrets

Créer à la main la Base64

```
echo -n "J'aime bien le rosé de Puyloubier" | base64
SidhaW1lIGJpZW4gbGUgcm9zw6kgZGUgUHV5bG91Ymllcg==
```

Créer le fichier : **monSecret.yml** ~~YAML apiVersion: v1 kind: Secret metadata: name: mon-secret data: dictondeprovence: SidhaW1lIGJpZW4gbGUgcm9zw6kgZGUgUHV5bG91Y~~

Créer le secret en ligne de commande: ~~SH kubectl create -f monSecret.yml~~

Fichier pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: afficheur-secret
spec:
  containers:
  - name: shell
    image: centos:7
    command:
      - "bin/bash"
      - "-c"
      - "sleep 10000"
    volumeMounts:
      - name: mondicton
        mountPath: "/etc/dicton"
        readOnly: true
  volumes:
  - name: mondicton
    secret:
      secretName: mon-secret
```

On peut afficher le contenu du secret: ~~~ kubectl exec afficheur-secret cat /etc/dicton/dictondeprovence ~~~

Variable d'environnement

- La grande différence avec Docker !

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo-frontend
spec:
  selector:
      matchLabels:
        app: demo-frontend
  replicas: 1
  template:
    metadata:
      labels:
        app: demo-frontend
    spec:
      containers:
        - name: nicolas
          image: nicolas:latest
          imagePullPolicy: Never
          ports:
            - containerPort: 8081
          env:
            - name: DEMO.PATH
              value: "Hello from the environment"
            - name: SECRETS.DEMO.PATH
              valueFrom:
                secretKeyRef:
                  name: spring-k8s-secrets
                  key: path
```

Cas d'usage Docker-Registry

- Authentification sur une Registry Docker
- Récupération des images privées

Création du Secret ~~~ kubectl create secret docker-registry nexus-credentials
–docker-server=nexus.foo.dev
–docker-username=admin
–docker-password=admin123
–docker-email=n.muller@treeptik.fr ~~~

Exploitation du Secret ~~~ apiVersion: v1 kind: Pod metadata: name: private-tomcat spec: containers: - name: chaton image: treeptik/tomcat:private imagePullSecrets: - name: nexus-credentials ~~~

Cas d'usage TLS

- Gestion des PKI
- Création à partir d'un couple clé public/privée

Création des clés ~~~ openssl req -newkey rsa:2048 -nodes -keyout key.pem -x509 -days 365 -out cert.pem ~~~

Création du Secret à partir des clés ~~~ kubectl create secret tls my-domain-certs –cert cert.pem -key key.pem ~~~

Exploitation du Secret ~~~ apiVersion: v1 kind: Pod metadata: name: proxy spec: containers: - name: proxy-nginx image: nginx:1.2 volumeMounts: - name: tls mountPath: "/etc/ssl/certs/" volumes: - name: tls secret: secretName: my-domain-certs ~~~

Contexte historique

Objectifs

- Techniquement ce qu'est la containerisation
- Bienfaits des conteneurs pour votre SI
- Différences entre les VMs et les conteneurs

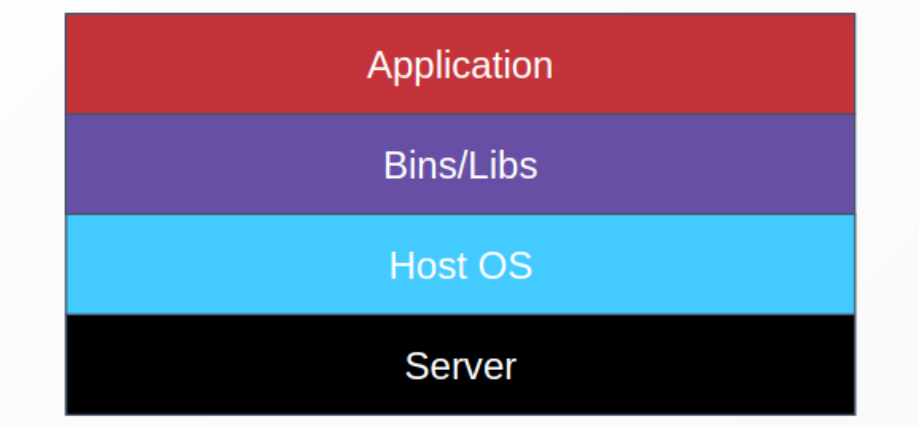
Evolution de la virtualisation et des applications

Serveur physique

De manière historique, le serveur physique était la plateforme de référence pour le déploiement d'une application. Certaines limites sur ce modèle sont apparues et ont été identifiées:

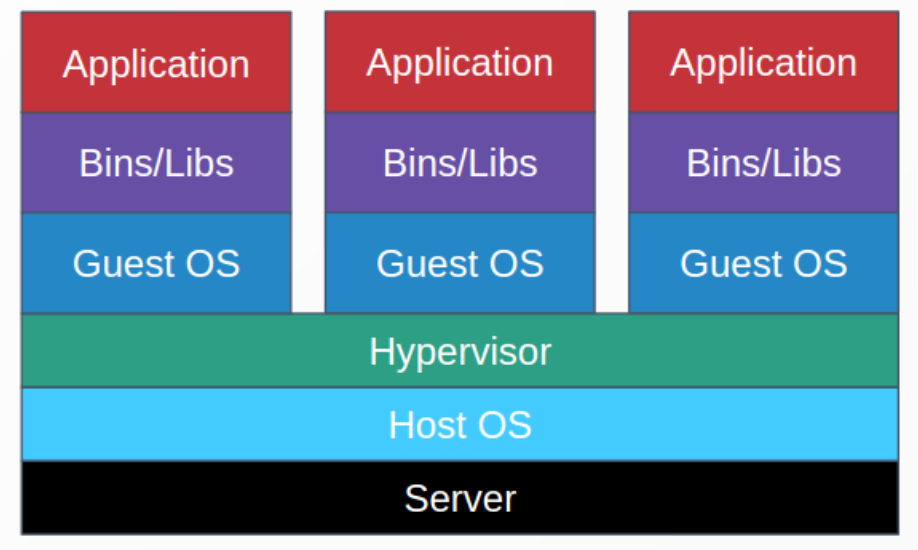
- Les temps de déploiements peuvent être longs
- Les coûts peuvent être élevés
- Beaucoup de ressources perdues
- Difficultés à mettre à l'échelle pour de la haute disponibilité
- Complexité de migration

Représentation d'une application sur serveur physique



Virtualisation basée sur hyperviseur

- Un serveur physique peut héberger plusieurs applications distinctes
- Chaque application tourne dans une machine virtuelle



Avantages du modèle de machine virtuelle

- Un serveur physique est divisé en plusieurs machines virtuelles
- Plus facile à mettre à l'échelle qu'un serveur physique
- Modèle de Cloud et paiement à la demande (AWS, Azure, Rackspace..)

Limitations du modèle de machine virtuelle

- Chaque VM nécessite une allocation de CPU, de stockage dédié, de RAM et un OS complet
- Modèle linéaire: l'augmentation du nombre de VM nécessite des ressource supplémentaires
- L'utilisation d'un système hôte complet entraîne une surcharge
- La portabilité des applications n'est toujours pas garantie

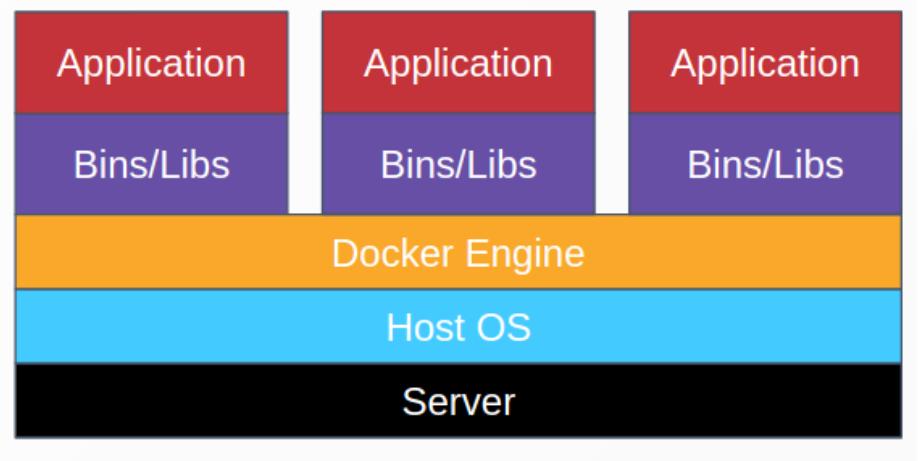
---

Historique du modèle d'isolation de processus

- UNIX chroot (1979-1982)
- BSD Jail (1998)
- Parallels Virtuozzo (2001)
- Solaris Containers (2005)
- Linux LXC (2008)
- Docker (2013)

Docker est une évolution de LXC qui a permis de rendre le container utilisable par un plus grand nombre d'utilisateurs grâce à son API et son client convivial.

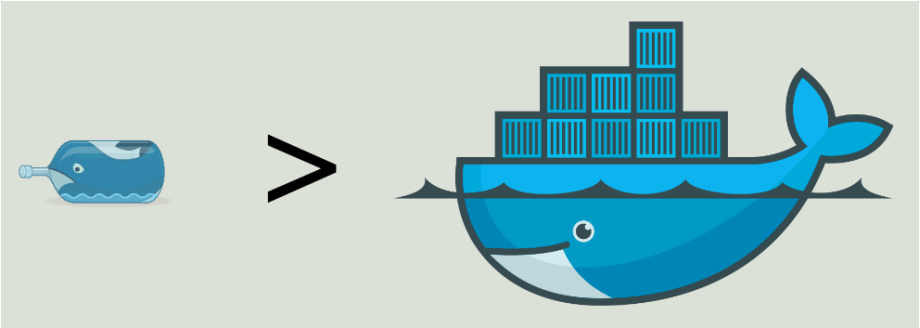
- La containerisation utilise le kernel du système hôte pour démarrer de multiples systèmes de fichiers racine
- Chaque système de fichier racine est appelé container
- Chaque container possède ses propres processus, mémoires, carte réseau



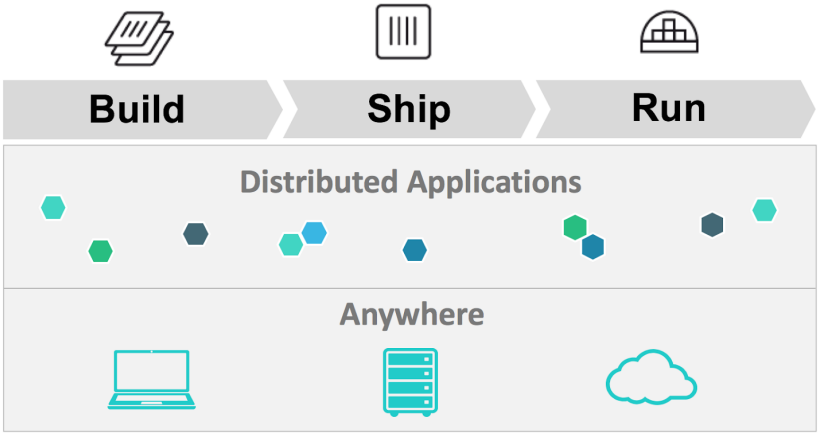
Pourquoi utiliser les containers ?

- Les containers sont plus légers et rapides que les VMs
- Pas besoin d'installer un système d'exploitation complet
- En conséquence, les besoins en CPU, RAM et stockage sont moins contraignants
- On peut faire tourner bien plus de containers sur un serveur que de VMs
- Le concept assure une meilleure portabilité
- Les containers représentent une meilleure solution pour développer et déployer des applications microservices

---



La mission de docker



Docker et le noyau Linux

NAMESPACES

- l'isolation des processus et du système de fichier
- l'isolation réseau et de disposer de ses propre interfaces

CGROUPS

- de mesurer et limiter les ressources (RAM, CPU, block I/O, network)
- de donner l'accès au différents périphériques (/dev/)

IPTABLES

- d'assurer la communication entre containers sur le même hôte
- d'assurer d'éventuelles communication entre les containers et l'extérieur

---

Le cycle de vie d'un container diffère de celui d'une VM

A l'opposé d'une VM, le conteneur n'est pas destiné à une existence perpétuelle. L'orchestrateur se chargera de redémarrer le container sur un autre hôte en cas de défaillance.

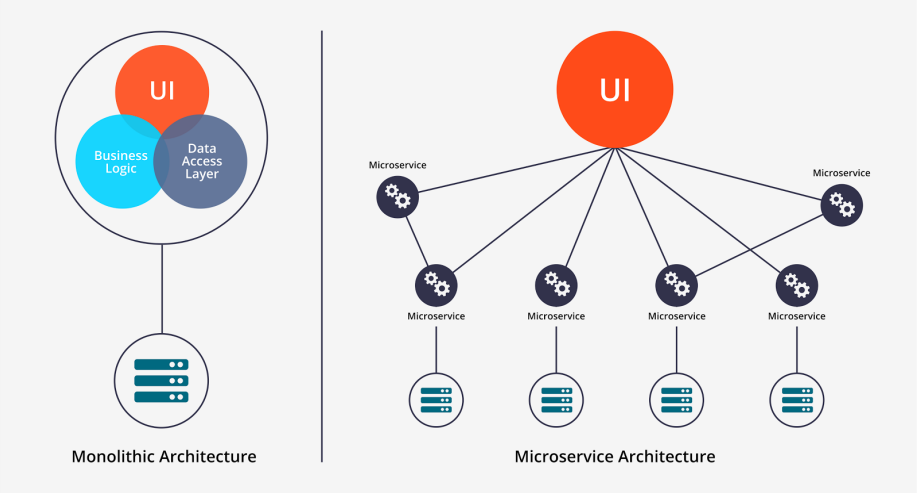
Cycle de vie basique d'un container: - Création d'un container à partir d'une image - Démarrage d'un container avec un processus - Le processus se termine et le container s'arrête - Le container est détruit

---

Applications modernes: architecture micro services

Une architecture microservices est un ensemble complexe d'applications décomposé en plusieurs processus indépendants et faiblement couplés.

Ces processus communiquent les uns avec les autres en utilisant des API. L'API REST est souvent employée pour relier chaque microservice aux autres.



Helm

Pourquoi ?

Arrêter de manipuler les fichiers K8S à la main

Définition

* **Helm** : Outils de gestion de déploiement d'application * **Charts** : Ensemble de ressources K8S configurable * **Release** : Livrable versionnable

---

Cas d'utilisations

Helm est utilisé pour : * Fabriquer ses fameux livrables configurables * Mettre à jour, supprimer et inspecter les livrables

---

Architecture

Helm est composé de deux parties distinctes : - **helm** client : créer, récupérer, rechercher et valider les **charts** - **tiller** server : agent dans le cluster K8S

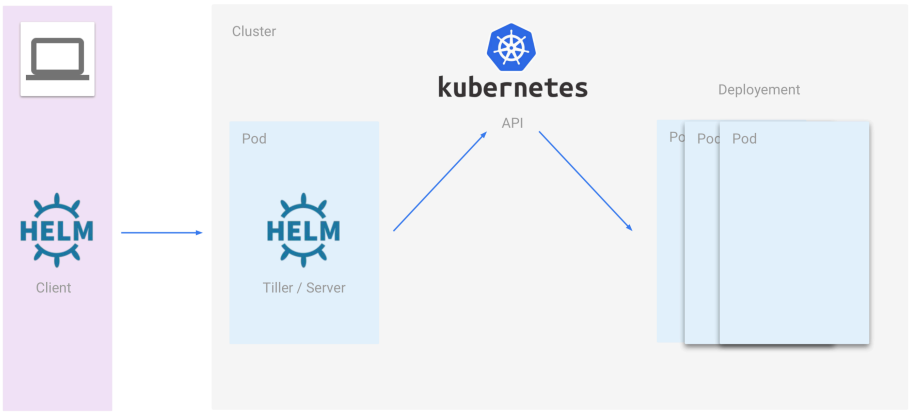


Figure 2: Architecture

Charts

- Ensemble d'objet K8S correspondant à une application - Instance d'une application Kubernetes - Exportable sous forme de package

~~bash helm create mychart~~

Chart

L'arborescence d'un chart est la suivante : ~~~bash mychart |– Chart.yaml |– charts |– templates | |– NOTES.txt | |– _helpers.tpl | |– deployment.yaml | |– ingress.yaml | `-- service.yaml`– values.yaml ~~~

Chart

Décrivons les fichiers d'un chart : * **Chart.yaml** : Manifest de l'application * **charts** : Dossier avec les dépendances du chart * **templates** : Dossier avec les définitions des objets K8S * **values.yaml** : Fichier avec les valeurs des variables

---

Templates

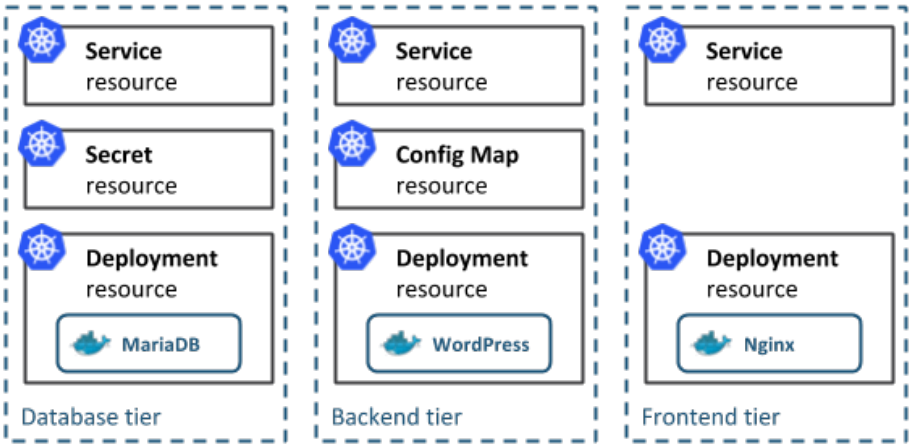


Figure 3: Charts

- Dossier de fichiers configurables - Fichiers yaml classiques étendus par helm. - Système de variables globales utilisables dans les fichiers yaml.

Templates

Exemple de fichier template : ~~~bash apiVersion: v1 kind: Service metadata: name: {{ include “mychart.fullname” . }} labels: app.kubernetes.io/name: {{ include “mychart.name” . }} helm.sh/chart: {{ include “mychart.chart” . }} app.kubernetes.io/instance: {{ .Release.Name }} app.kubernetes.io/managed-by: {{ .Release.Service }} spec: type: {{ .Values.service.type }} ports: - port: {{ .Values.service.port }} targetPort: http protocol: TCP name: http selector: app.kubernetes.io/name: {{ include “mychart.name” . }} app.kubernetes.io/instance: {{ .Release.Name }}~~~

Par exemple {{ .Values.service.port }} est le nom d'une variable.

Variables

Dans le fichier values.yaml : ~~~bash image: repository: nginx~~~

Au lancement de helm : ~~~bash helm install --name example ./mychart --set image.repository=alpine~~~

Installation 1 // 4

Télécharger helm avec la commande suivante : ~bash curl https://raw.githubusercontent.com/kubernetes/helm/ > get_helm.sh~ Puis installer le avec les commandes suivantes : ~bash chmod 700 get_helm.sh ./get_helm.sh~

Installation 2 // 4

Vérifier si le cluster role cluster-admin est présent sur le cluster : ~bash kubectl get clusterrole cluster-admin~ Si le résultat de la commande ressemble à ce qui suit passer directement a la diapo Instalation (4/4) ~bash NAME AGE cluster-admin 4h42m~ Sinon il faut créer ce cluster role.

Installation 3 // 4

Créer un fichier clusterrole.yaml : ~~~bash apiVersion: rbac.authorization.k8s.io/v1 kind: ClusterRole metadata: annotations: rbac.authorization.kubernetes.io/autoupdate: "true" labels: kubernetes.io/bootstrapping: rbac-defaults name: cluster-admin rules: - apiGroups: - '' *resources:* - '' verbs: - '' - *nonResourceURLs:* - '' verbs: - '*' ~~~ Puis créer le ClusterRole : ~bash kubectl create -f clusterrole.yaml~

Installation 4 // 4

Créer un service account avec la commande suivante : ~bash kubectl create serviceaccount -n kube-system tiller~ Associer le avec le cluster role cluster-admin : ~bash kubectl create clusterrolebinding tiller-cluster-rule -clusterrole=cluster-admin --serviceaccount=kube-system:tiller~ Et enfin initialiser helm : ~bash helm init --service-account tiller~

Pour continuer

https://github.com/helm/monocular

EXERCICE

HELM

Dans un premier terminal

```
helm serve
Regenerating index. This may take a moment.
Now serving you on 127.0.0.1:8879
```

Dans un second terminal

```
helm search local
NAME            VERSION DESCRIPTION
local/mychart   0.1.0   A Helm chart for Kubernetes
helm install --name example4 local/mychart --set service.type=NodePort
To setup a remote repository you can follow the guide in the Helm documentation.
```

Dépendences

```
cat > ./mychart/requirements.yaml <<EOF
dependencies:
- name: mariadb
version: 0.6.0
repository: https://kubernetes-charts.storage.googleapis.com
EOF
```

```
helm dep update ./mychart
Hang tight while we grab the latest from your chart repositories...
...Unable to get an update from the "local" chart repository (http://127.0.0.1:8879/charts):
   Get http://127.0.0.1:8879/charts/index.yaml: dial tcp 127.0.0.1:8879: getsockopt: connectio
...Successfully got an update from the "stable" chart repository
...Successfully got an update from the "incubator" chart repository
Update Complete. *Happy Helming!*
Saving 1 charts
Downloading mariadb from repo [https://kubernetes-charts.storage.googleapis.com](https://kube
ls ./mychart/charts
mariadb-0.6.0.tgz
```

```
helm install --name example5 ./mychart --set service.type=NodePort
NAME:   example5
LAST DEPLOYED: Wed May  3 16:28:18 2017
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Secret
NAME               TYPE    DATA  AGE
example5-mariadb  Opaque  2     1s

==> v1/ConfigMap
NAME               DATA  AGE
```

```
example5-mariadb  1     1s

==> v1/PersistentVolumeClaim
NAME          STATUS  VOLUME                                     CAPACITY  ACCESSMODES  AGE
example5-mariadb  Bound  pvc-229f9ed6-3015-11e7-945a-66fc987ccf32  8Gi     RWO       1s

==> v1/Service
NAME              CLUSTER-IP  EXTERNAL-IP  PORT(S)       AGE
example5-mychart  10.0.0.144  <nodes>      80:30896/TCP  1s
example5-mariadb  10.0.0.108  <none>       3306/TCP      1s

==> extensions/v1beta1/Deployment
NAME              DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
example5-mariadb  1        1        1           0          1s
example5-mychart  1        1        1           0          1s
```

CHART repository

Exposer ses packages à distance

Ceci est possible avec * GitHub * JFrog * Google Container Registry * Azure Container Registry * Tout serveur HTTP

DEFINITION

* Serveur HTTP hébergeant un fichier **index.yaml** * [Option] Les Charts associés

---

Exemple

https://example.com/charts pourrait répondre

```
charts/
  |
  |- index.yaml
  |
  |- alpine-0.1.2.tgz
  |
  |- alpine-0.1.2.tgz.prov
```

Index.yaml

- Contient les *metadata* du package
- Incluant le contenu des fichiers *Chart.yaml*
- Obligatoire pour que le repository soit valide
- La commande **helm repo index
  ** génére le fichier

```
apiVersion: v1
entries:
  mychart:
  - apiVersion: v1
    appVersion: "1.0"
    created: 2018-11-21T09:04:24.931099051Z
    description: A Helm chart for Kubernetes
   digest: 134c76a7a43932e47afec8b9655064d11d85e60f86c5e0efea361d5612ad4014
    name: mychart
    urls:
    - mychart-0.1.0.tgz
    version: 0.1.0
generated: 2018-11-21T09:04:24.929984579Z
```

```
helm repo index my-first-chart
helm repo index my-first-chart --url https://fantastic-charts.storage.googleapis.com
```

```
helm repo add fantastic-charts https://fantastic-charts.storage.googleapis.com
$ helm repo list
fantastic-charts   https://fantastic-charts.storage.googleapis.com
```

```
$ helm repo add fantastic-charts https://fantastic-charts.storage.googleapis.com --username my
$ helm repo list
fantastic-charts   https://fantastic-charts.storage.googleapis.com
```

HOOKS

Objectif

- Intervenir dans le cycle de vie de la livraison

- Executer une tâche de backup de base de données avant déploiement du nouveau chart par exemple puis enchaîner un autre job pour les restaurer
- Lancer un job de suppression de ressources externes avant de supprimer la release

---

Déclaration comme annotations

```
apiVersion: ...
kind: ....
metadata:
  annotations:
    "helm.sh/hook": "pre-install"
# ...
```

Hooks disponibles

- pre-install
- post-install
- pre-delete
- post-delete
- pre-upgrade
- post-upgrade
- pre-rollback have been rolled back
- post-rollback
- crd-install

```
apiVersion: batch/v1
kind: Job
metadata:
  name: "{{.Release.Name}}"
  labels:
    app.kubernetes.io/managed-by: {{.Release.Service | quote }}
    app.kubernetes.io/instance: {{.Release.Name | quote }}
    helm.sh/chart: "{{.Chart.Name}}-{{.Chart.Version}}"
  annotations:
   # This is what defines this resource as a hook. Without this line, the
    # job is considered part of the release.
    "helm.sh/hook": post-install
    "helm.sh/hook-weight": "-5"
    "helm.sh/hook-delete-policy": hook-succeeded
spec:
```

```
template:
  metadata:
    name: "{{.Release.Name}}"
    labels:
     app.kubernetes.io/managed-by: {{.Release.Service | quote }}
      app.kubernetes.io/instance: {{.Release.Name | quote }}
      helm.sh/chart: "{{.Chart.Name}}-{{.Chart.Version}}"
  spec:
    restartPolicy: Never
    containers:
    - name: post-install-job
      image: "alpine:3.3"
    command: ["/bin/sleep","{{default "10" .Values.sleepyTime}}"]
```

Trois sorties possibles

```
annotations:
    "helm.sh/hook-delete-policy": hook-succeeded
```

```
annotations:
    "helm.sh/hook-delete-policy": hook-failed
```

```
annotations:
    "helm.sh/hook-delete-policy": before-hook-creation
```

Exemple NGINX complet

https://github.com/helm/helm/tree/master/docs/examples/nginx

Installation (A REPRENDRE)

Installation de Kubernetes

Les différentes possibilités d'installer Kubernetes

En local avec minikube : - VT-x ou AMD-v virtualization doivent être activer dans le BIOS - Installer un hypervieur (virtualbox) - Installer Kubectl (client qui permet de communiquer avec le cluster) - Installer Minikube (outil qui permet

de déployer tous les composants de kubernetes dans une VM) - Minikube installe par défault les outils réseaux permettant la communication au sein du cluster

Sur un serveur avec kubedam : - Outil qui permet de déployer un cluster Kubernetes facilement - Utiliser CentOS ou Ubuntu - 2GB par serveur - 2CPUS ou plus pour le master - Installer Docker - Kubeadm n'installe pas par défault les outils réseaux permettant la communication au sein du cluster

---

Installation de Kubernetes

Installation en local avec minikube (1/2)

- Installer kubectl sur Linux

```
$  curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://stor
```

- Installer kubectl sur MacOS

```
$  curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://stora
```

Rendre le binaire éxecutable

```
$  chmod +x kubectl
```

Déplacer le binaire dans votre PATH

```
$  sudo mv kubectl /usr/local/bin/kubectl
```

Installation de Kubernetes

Installation en local avec minikube (2/2)

- Installation de minikube

Récupérer le dernière version de minikube correspondant à votre OS: https://github.com/kubernetes/minikube/releases

```
$  minikube start
```

Il est possible de spécifier l'hyperviseur lors du démarrage de minikube

```
$  minikube start --driver=xxx
```

Installation de Kubernetes

Installation sur un serveur avec kubeadm (1/5)

Sur l'ensemble des serveurs :

Installer Docker avec la version recomandée 1.12. Les versions 1.11, 1.13, et 17.03 fonctionnent également ~~bash $ apt-get install docker.io~~

Installer kubelet, kubeadm, kubectl

```
apt-get update && apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF
apt-get update
apt-get install -y kubelet kubeadm kubectl
```

Installation de Kubernetes

Installation sur un serveur avec kubeadm (2/5)

Initialiser le master (cela peut prendre plusieurs minutes) ~~bash sudo kubeadm init –pod-network-cidr=10.244.0.0/16~~

L'installation effectue les actions suivantes : - Création des clés et certificats - Ecriture sur le disque les fichiers de configurations dans /etc/kubernetes - Déploiement des composants du cluster - Affiche la Token qui permet de joindre des noeuds

Notez bien cette token, elle sera utilisée dans la suite de l'installation

Par défault, l'installation de Kubernetes n'installe pas de CNI (Container Network Interface) qui est nécessaire pour la communication entre les pods

---

Installation de Kubernetes

Installation sur un serveur avec kubeadm (3/5)

Une fois l'installation du master terminé, exportez le fichier de configuration de Kubernetes dans votre home pour pouvoir intéragir avec le cluster

```
$mkdir -p $HOME/.kube
$sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Installer un CNI pour la communication entre les pods Le CNI utilisé sera Flannel

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/v0.9.1/Documentation/kube-
```

Installation de Kubernetes

Installation sur un serveur avec kubeadm (4/5)

Vérifier que tous les composants sont dans l'état "Ready" et avec le status "Running" ~~~bash $kubectl get pods --all-namespaces~~~

```
NAMESPACE     NAME                                          READY     STATUS    RESTARTS   AGE
kube-system   etcd-treeptik.mylabserver.com                 1/1       Running   0          10m
kube-system   kube-apiserver-treeptik.mylabserver.com       1/1       Running   4          13m
kube-system   kube-controller-manager-treeptik.mylabserver.com   1/1       Running   0          10m
kube-system   kube-dns-6f4fd4bdf-qvbhh                      3/3       Running   0          10m
kube-system   kube-flannel-ds-7r67v                         1/1       Running   0          3m
kube-system   kube-proxy-shxtq                              1/1       Running   0          10m
kube-system   kube-scheduler-treeptik.mylabserver.com       1/1       Running   0          10m
```

Installation de Kubernetes

Installation sur un serveur avec kubeadm (5/5)

Une fois les pods "kube-dns" sont démarrés, le cluster est prêt à accepter d'autres membres dans le cluster

Pour ajouter un membre dans le cluster, récupérer la token affichée sur le terminal à la fin de l'installation, et la coller sur le serveur que l'on souhaite joindre au cluster

Si n'avez pas noter la token, il est possible d'en générer une nouvelle

```
$kubeadm token create --print-join-command
```

Depuis le master, vérifier que le serveur est présent dans le cluster ~~~bash $kubectl get nodes~~~

Job

Objets K8S

- Job
- CronJob

Cas d'usage

- Traitement Batch
- Traitement horaires

Nature du Job

- **Contrôleur** qui crée et s'assure que les pods se terminent bien.
- Supprimer un **Job** implique la suppression de ses pods
- Recréation des pods si ces derniers sont perdus (node crash ou suppression manuelle)

Exemple

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
      command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

```
kubectl create -f ./examples/controllers/job.yaml
job "pi" created
```

```
kubectl describe jobs/pi
Name:             pi
Namespace:        default
```

```
Selector:       controller-uid=b1db589a-2c8d-11e6-b324-0209dc45a495
Labels:         controller-uid=b1db589a-2c8d-11e6-b324-0209dc45a495
                   job-name=pi
Annotations:       <none>
Parallelism:       1
Completions:       1
Start Time:        Tue, 07 Jun 2016 10:56:16 +0200
Pods Statuses:     0 Running / 1 Succeeded / 0 Failed
Pod Template:
  Labels:       controller-uid=b1db589a-2c8d-11e6-b324-0209dc45a495
                job-name=pi
  Containers:
   pi:
    Image:      perl
    Port:
    Command:
      perl
      -Mbignum=bpi
      -wle
      print bpi(2000)
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Events:
  FirstSeen   LastSeen   Count   From        SubobjectPath   Type      Reason         Message
  ---------   --------   -----   ----        -------------   --------  ------         -------
  1m        1m       1     {job-controller }           Normal    SuccessfulCreate  Created pod:
```

```
$ pods=$(kubectl get pods --selector=job-name=pi --output=jsonpath={.items..metadata.name})
$ echo $pods
pi-aiw0a

$ kubectl logs $pods
3.1415926535897...
```

Spécifications

Pod Template

RestartPolicy * Never * OnFailure

Parallel Jobs

Trois types de Jobs * Non-Parallel * Parallel Job with fixed completion count `.spec.completions` * Parallel Jobs with a work queue * `.spec.parallelism` à définir * `.spec.completions` à exclure

---

Terminaison

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-with-timeout
spec:
  backoffLimit: 5
  activeDeadlineSeconds: 100
  template:
    spec:
      containers:
      - name: pi
        image: perl
      command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```

Nettoyage automatique

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-with-ttl
spec:
  ttlSecondsAfterFinished: 100
  template:
    spec:
      containers:
      - name: pi
        image: perl
      command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```

Tâches planifiées

```
apiVersion: batch/v1beta1
```

```
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

```
$ kubectl create -f ./cronjob.yaml
cronjob "hello" created
```

```
$ kubectl run hello --schedule="*/1 * * * *" --restart=OnFailure
                    --image=busybox
              -- /bin/sh -c "date; echo Hello from the Kubernetes cluster"
cronjob "hello" created`
```

```
$ kubectl get cronjob hello
NAME      SCHEDULE      SUSPEND   ACTIVE    LAST-SCHEDULE
hello     */1 * * * *   False     0         <none>
```

```
$ kubectl get jobs --watch
NAME               DESIRED   SUCCESSFUL   AGE
hello-4111706356   1         1         2s
```

```
$ kubectl get cronjob hello
NAME      SCHEDULE      SUSPEND   ACTIVE    LAST-SCHEDULE
hello    */1 * * * *  False    0       Mon, 29 Aug 2016 14:34:00 -0700
```

Network Policies et Kubernetes

Rappel sur les Namespaces

- Scope de travail pour les objets Kubernetes
- Assimilable à un environnement de travail

- Evite les collisions de noms
- QoS sur le namespace concerné

EXERCICE NAMESPACES

Network Policies

Introduction

- Non actif par défaut
- Depuis la version 1.7 de Kubernetes
- ACL entre pods
- Application en temps réel (avec peu d'impact sur la performance)

Note: Network Policies is a new Kubernetes feature to configure how groups of pods are allowed to communicate with each other and other network endpoints In other words, it creates firewalls between pods running on a Kubernetes cluster. This guide is meant to explain the unwritten parts of Kubernetes Network Policies.

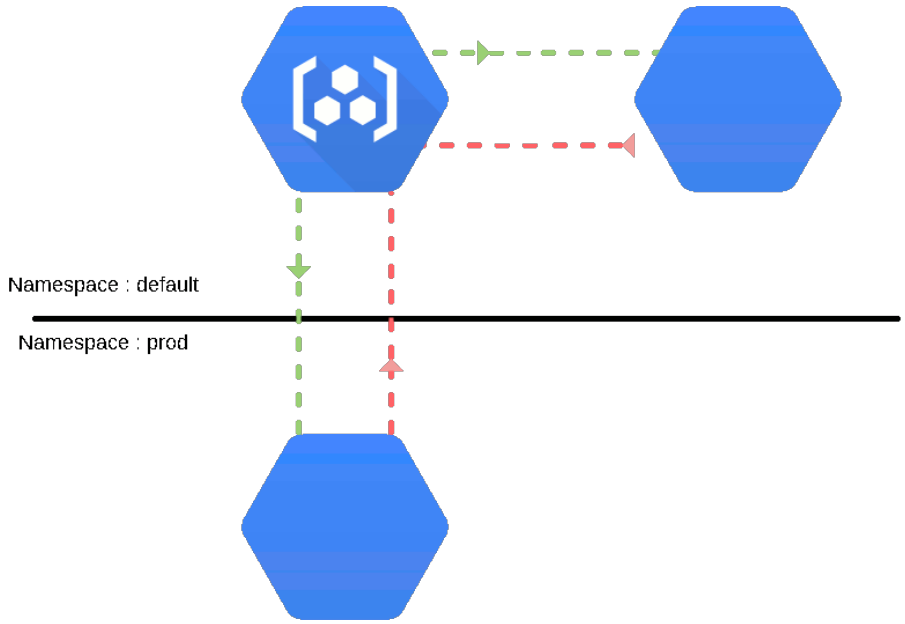
Network Policies

Cas d'utilisation Simple

- DENY all : refuser tout le trafic vers une application
- LIMIT : limiter le trafic vers une application
- ALLOW all : autoriser tout le trafic vers une application

Network Policies

Cas d'utilisation : DENY all



Deny All

- Lancer un pod nginx écoutant sur la port 80 :

```
kubectl run web --image=nginx --labels app=bibliotheque --expose --port 80
```

- Lancer un pod alpine pour tester la connexion avec le premier pod

```
kubectl run --rm -i -t --image=alpine test-$RANDOM -- sh
/ # wget -qO- http://web
```

Note: En cas de bug avec les images faire un docker pull avant

---

Deny All (2/3)

- Créer un fichier `web-deny-all.yaml`

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
```

```
metadata:
  name: default-deny-ingress
  namespace: advanced-policy-demo
spec:
  podSelector:
    matchLabels: {}
  policyTypes:
  - Ingress
```

- Appliquer la configuration

```
kubectl apply -f web-deny-all.yaml
```

Deny All (3/3)

- Tester la connexion

```
kubectl run test-$RANDOM --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=10 http://web
```

- Nettoyer l'environnement

```
kubectl delete deploy web
kubectl delete service web
kubectl delete networkpolicy web-deny-all
```

Network Policies

Cas d'utilisation : LIMIT

Limit

- Lancer une application

```
kubectl run apiserver --image=nginx --labels app=bibliotheque,role=api --expose --port 80
```

- Créer une Network policy `api-allow.yaml`

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: bibliotheque
      role: api
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bibliotheque
```

- Appliquer la configuration

```
kubectl apply -f api-allow.yaml
```

Limit (2/3)

- Tester la connexion

```
kubectl run test-$RANDOM --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=10 http://apiserver
```

- Tester la connexion avec les bons paramètres

```
kubectl run test-$RANDOM --rm -i -t --image=alpine --labels app=bibliotheque,role=front -- s
/ # wget -qO- --timeout=2 http://apiserver
```

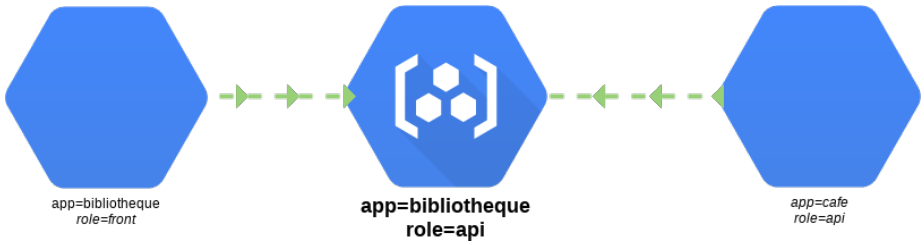
Limit (3/3)

- Nettoyer l'environnement

```
kubectl delete deployment apiserver
kubectl delete service apiserver
kubectl delete networkpolicy api-allow
```

Network Policies

Cas d'utilisation : ALLOW all



Network Policies

Cas d'utilisation sur les Namespaces

- DENY all non-whitelisted traffic in the current namespace
- DENY all traffic from other namespaces (a.k.a. LIMIT access to the current namespace)
- ALLOW traffic to an application from all namespaces
- ALLOW all traffic from a namespace

ALLOW all

- Lancer une application

```
kubectl run web --image=nginx --labels=app=web --expose --port 80
```

- Créer une Network policy `web-allow-all.yaml`

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all
  namespace: default
```

```
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - {}
```

- Appliquer la configuration

```
kubectl apply -f web-allow-all.yaml
```

ALLOW all (2/2)

- Tester la connexion

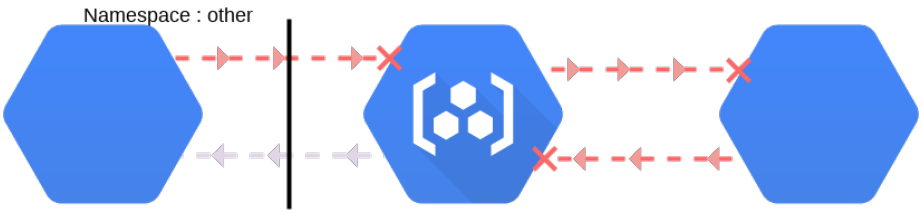
```
kubectl run test-$RANDOM --rm -i -t --image=alpine --labels app=whitelisted -- sh
/ # wget -qO- --timeout=2 http://web
```

- Nettoyer l'environnement

```
kubectl delete deployment,service web
kubectl delete networkpolicy web-allow-all web-deny-all
```

Network Policies

Namespaces : DENY all non-whitelisted



DENY all non-whitelisted

- Lancer une application

```
kubectl run web --image=nginx --labels=app=web --expose --port 80
```

- Créer une Network policy `web-allow-whitelisted.yaml`

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: whitelisted
```

- Appliquer la configuration

```
kubectl apply -f web-allow-whitelisted.yaml
```

DENY all non-whitelisted (2/3)

- Tester la connexion avec les bons paramètres

```
kubectl run test-$RANDOM --rm -i -t --image=alpine --labels app=whitelisted -- sh
/ # wget -qO- --timeout=2 http://web

kubectl run test-$RANDOM --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://web
```

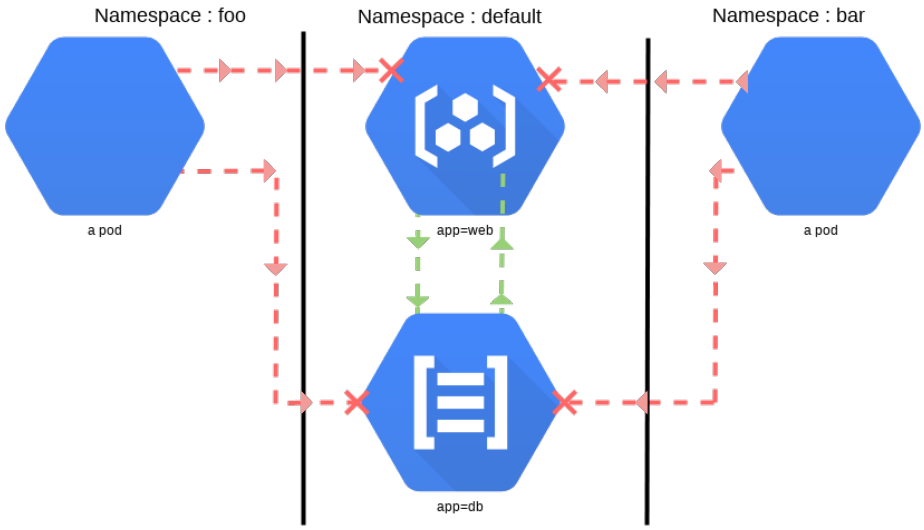
DENY all non-whitelisted (3/3)

- Nettoyer l'environnement

```
kubectl delete deployment,service web
kubectl delete networkpolicy web-allow-all
```

Network Policies

Namespaces : DENY all other namespaces



DENY all other

- Création d'un second namespace

```
kubectl create namespace foo
kubectl create namespace bar

kubectl run web --namespace foo --image=nginx --labels=app=web --expose --port 80
```

DENY all other (2/3) BUGS

- Créer une configuration `deny-from-other-namespaces.yaml`

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: foo
  name: deny-from-other-namespaces
spec:
  podSelector:
```

```
    matchLabels:
  ingress:
    - from:
      - podSelector:
          matchLabels: {}
```

- Appliquer la configuration

```
kubectl apply -f deny-from-other-namespaces.yaml
```

DENY all other (3/3)

- Tester la connexion

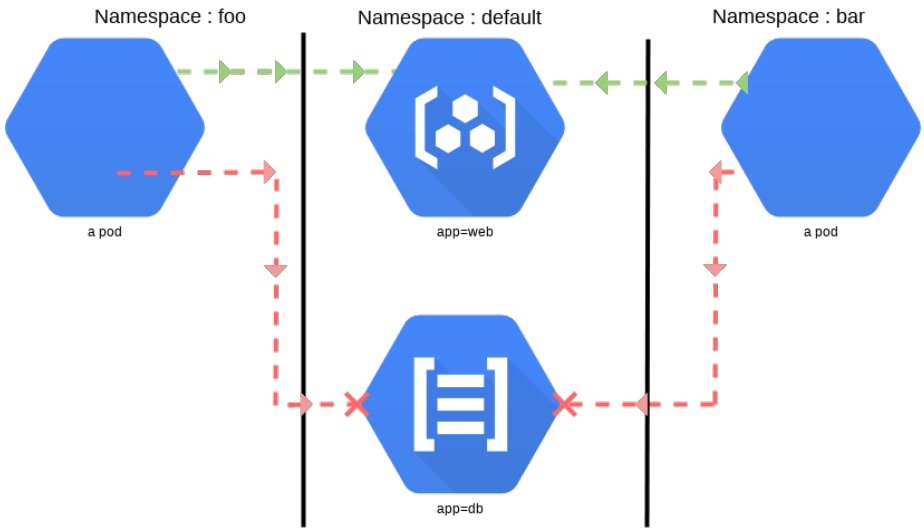
```
kubectl run test-$RANDOM --namespace=default --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://web.foo


kubectl run test-$RANDOM --namespace=foo --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://web.foo
/ # wget -qO- --timeout=2 http://web


kubectl run test-$RANDOM --namespace=bar --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://web.foo
```

- Nettoyer :

```
kubectl delete deployment web -n foo
kubectl delete service web -n foo
kubectl delete networkpolicy deny-from-other-namespaces -n foo
kubectl delete namespace foo
kubectl delete namespace bar
```

Network Policies

Namespaces : Allow from other Namespaces



Allow from other

- Création d'un second namespace

```
kubectl create namespace foo
kubectl create namespace bar

kubectl run web --namespace default --image=nginx --labels=app=web --expose --port 80
kubectl run web2 --namespace default --image=nginx --labels=app=web2 --expose --port 80
```

Allow from other (2/3)

- Créer une configuration `web-allow-all-namespaces.yaml`

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: foo
  name: web-allow-all-namespaces
spec:
```

```
  podSelector:
    matchLabels:
      app: web
  ingress:
  - from:
    - namespaceSelector: {}
```

- Appliquer la configuration

```
kubectl apply -f deny-from-other-namespaces.yaml
kubectl apply -f web-allow-all-namespaces.yaml
```

Allow from other (3/3)

- Tester la connexion

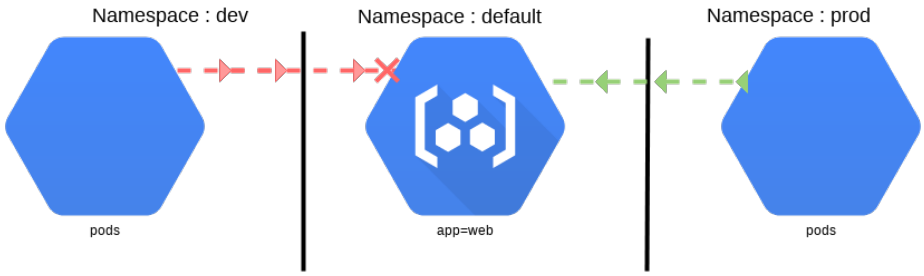
```
kubectl run test-$RANDOM --namespace=bar --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://web.foo
/ # wget -qO- --timeout=2 http://web2.foo
```

- Nettoyer :

```
kubectl delete deployment,service web -n foo
kubectl delete deployment,service web2 -n foo
kubectl delete networkpolicy web-allow-all-namespaces -n foo
kubectl delete namespace foo
kubectl delete namespace bar
```

Network Policies

Namespaces : Allow from a namespace



Autorisation depuis un namespace

- Création de 2 namesapces et d'un pod nginx

```
kubectl create namespace dev
kubectl create namespace prod

kubectl run web --namespace default --image=nginx --labels=app=web --expose --port 80
```

- Ajout d'un label sur les namespaces

```
kubectl label namespace/dev purpose=testing
kubectl label namespace/prod purpose=production
```

Autorisation depuis un namesapce (2/3)

- Création d'un fichier `web-allow-prod.yaml`

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: production
```

- Appliquer la configuration

```
kubectl apply -f web-allow-prod.yaml
```

Autorisation depuis un namesapce (3/3)

- Tester l'accès depuis le namespace de dev :

```
kubectl run test-$RANDOM --namespace=dev --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://web.default
```

- Tester l'accès depuis le namespace de prod :

```
kubectl run test-$RANDOM --namespace=prod --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://web.default
```

- Nettoyage

```
kubectl delete networkpolicy web-allow-prod
kubectl delete deployment,service web
kubectl delete namespace {prod,dev}
```

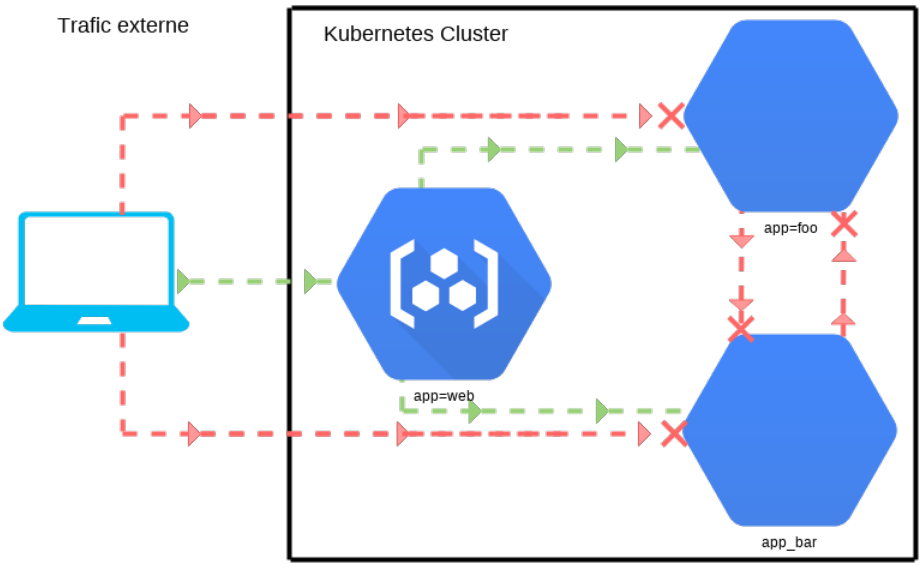
Network Policies

Cas d'utilisation particulier

- External
- Autorisation sur un port particulier
- Multiple sélection

Network Policies

External



Filtrage d'accès depuis l'extérieur

- Création d'un pod nginx

```
kubectl run web --namespace default --image=nginx --labels=app=web --expose --port 80
kubectl run test-nginx --namespace default --image=nginx --labels=app=test-nginx --expose --
```

- Exposition du pod nginx à l'extérieur et attrendre l'obtention de l'ip public

```
## erreur
## kubectl expose deployment/web --type=LoadBalancer
## watch kubectl get service

#temp :
kubectl port-forward web-669c8bff75-nw4fc 8081:80
```

Filtrage d'accès depuis l'extérieur (2/4)

- Création d'un fichier `web-allow-external.yaml`

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-external
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - from: []
```

- Appliquer la configuration de restriction d'accès complète et l'autorisastion d'accès au pod nginx

```
kubectl apply -f default-deny-all.yaml
kubectl apply -f web-allow-external.yaml
```

Filtrage d'accès depuis l'extérieur (3/4)

- Test d'accès http://EXTERNAL-IP
- Test depuis les autres pods

```
kubectl run test-$RANDOM --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://web
/ # wget -qO- --timeout=2 http://test-nginx
```

Filtrage d'accès depuis l'extérieur (4/4)

- Restriction d'accès sur un port uniquement

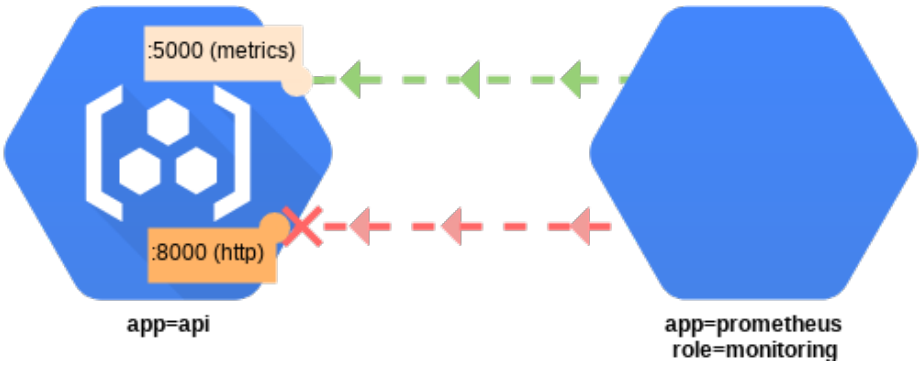
```
ingress:
  - ports:
    - port: 80
    from: []
```

- Nettoyage

```
kubectl delete deployment, service web
kubectl delete networkpolicy default-deny-all
kubectl delete networkpolicy web-allow-external default-deny-all
```

Network Policies

Port particulier



Filtrage sur un port particulier

- Mise en place d'un serveur et création d'un accès :

```
kubectl run apiserver --image=treeptik/nginx:alpine --labels=app=apiserver

kubectl create service clusterip apiserver \
    --tcp 8001:8000 \
    --tcp 5001:5000
```

Filtrage sur un port particulier (2/3)

- Création d'un fichier `api-allow-5000.yaml`

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow-5000
spec:
  podSelector:
    matchLabels:
      app: apiserver
  ingress:
  - ports:
    - port: 5000
    from:
    - podSelector:
        matchLabels:
          role: monitoring
```

- Application de la règle

```
kubectl apply -f api-allow-5000.yaml
```

Filtrage sur un port particulier (3/3)

- Test de la connexion

```
kubectl run test-$RANDOM --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://apiserver:8001
/ # wget -qO- --timeout=2 http://apiserver:5001/metrics
```

- Test avec un pod avec le role de monitoring

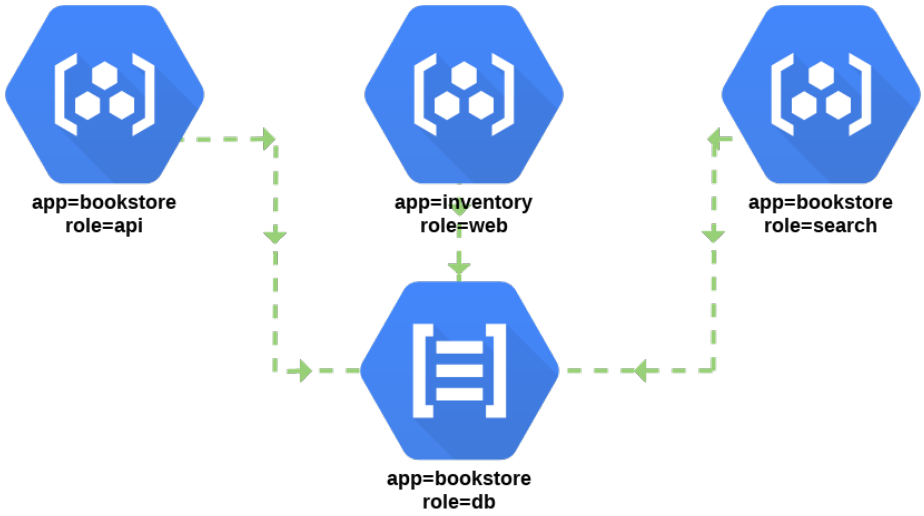
```
kubectl run test-$RANDOM --labels=role=monitoring --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout=2 http://apiserver:8001
/ # wget -qO- --timeout=2 http://apiserver:5001/metrics
```

- Nettoyage

```
kubectl delete deployment,service apiserver
kubectl delete networkpolicy api-allow-5000
```

Network Policies

Filtre multiple



Filtre multiple

- Mise en place d'une base redis :

```
kubectl run db --image=redis:4 --port 6379 --expose --labels app=bookstore,role=db
```

- Création d'une règle de filtrage `redis-allow-services.yaml`

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: redis-allow-services
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: db
```

```
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: bookstore
          role: search
    - podSelector:
            matchLabels:
              app: bookstore
              role: api
    - podSelector:
            matchLabels:
              app: inventory
              role: web
```

Filtrage sur un port particulier (2/3)

- Appliquer la règle

```
kubectl apply -f redis-allow-services.yaml
```

- Test de la configuration

```
kubectl run test-$RANDOM --labels=app=inventory,role=web --rm -i -t --image=alpine -- sh
/ # nc -v -w 2 db 6379
```

- Test de la configuration

```
kubectl run test-$RANDOM --labels=app=other --rm -i -t --image=alpine -- sh
/ # nc -v -w 2 db 6379
```

Filtrage sur un port particulier (3/3)

- Nettoyage de l'environnement

```
kubectl delete deployment db
kubectl delete service db
kubectl delete networkpolicy redis-allow-services
```

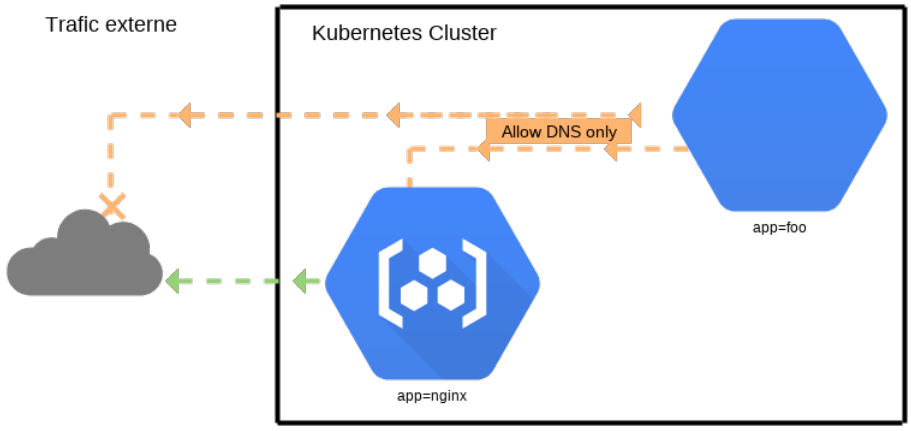
Network Policies

Gestion du trafic sortant (Egress)

- DENY egress traffic from an application
- DENY all non-whitelisted egress traffic in a namespace
- LIMIT egress traffic from an application to some pods
- ALLOW traffic only to Pods in a namespace
- LIMIT egress traffic to the cluster (DENY external egress traffic)

Network Policies

Interdire tout le trafic sortant



Interdire tout le trafic sortant

- Mise en place d'un pod nginx :

```
kubectl run web --image=nginx --port 80 --expose --labels app=web
```

- Création d'une règle de blocage de tout le trafic sortant `foo-deny-egress.yaml`

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: foo-deny-egress
spec:
  podSelector:
```

```
    matchLabels:
      app: foo
  policyTypes:
  - Egress
  egress: []
```

- Appliquer la règle

```
kubectl apply -f foo-deny-egress.yaml
```

Interdire tout le trafic sortant (2/4)

- Test de la configuration

```
kubectl run test-$RANDOM --labels=app=foo --rm -i -t --image=alpine -- sh
/ # wget -qO- --timeout 1 http://web:80/
/ # wget -qO- --timeout 1 http://www.example.com/
/ # ping google.com
```

Filtrage sur un port particiluer (3/4)

- Edition pour ouvrir les ports DNS

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: foo-deny-egress
spec:
  podSelector:
    matchLabels:
      app: foo
  policyTypes:
  - Egress
  egress:
  # allow DNS resolution
  - ports:
    - port: 53
      protocol: UDP
    - port: 53
      protocol: TCP
```

Filtrage sur un port particulier (4/4)

- Edition pour ouvrir les ports DNS

```
kubectl run --rm --restart=Never --image=alpine -i -t -l app=foo test -- ash
/ # wget --timeout 1 -O- http://web
/ # wget --timeout 1 -O- http://www.example.com
/ # ping google.com
/ # exit
```

- Nettoyage de l'environnement

```
kubectl delete deployment,service cache
kubectl delete deployment,service web
kubectl delete networkpolicy foo-deny-egress
```

Network Policies

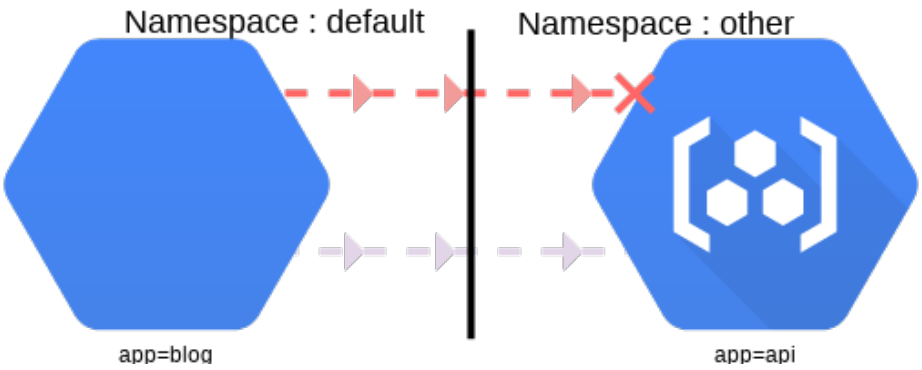
TO DO

Gestion du trafic sortant (Egress)

- DENY egress traffic from an application
- DENY all non-whitelisted egress traffic in a namespace
- LIMIT egress traffic from an application to some pods
- ALLOW traffic only to Pods in a namespace
- LIMIT egress traffic to the cluster (DENY external egress traffic)

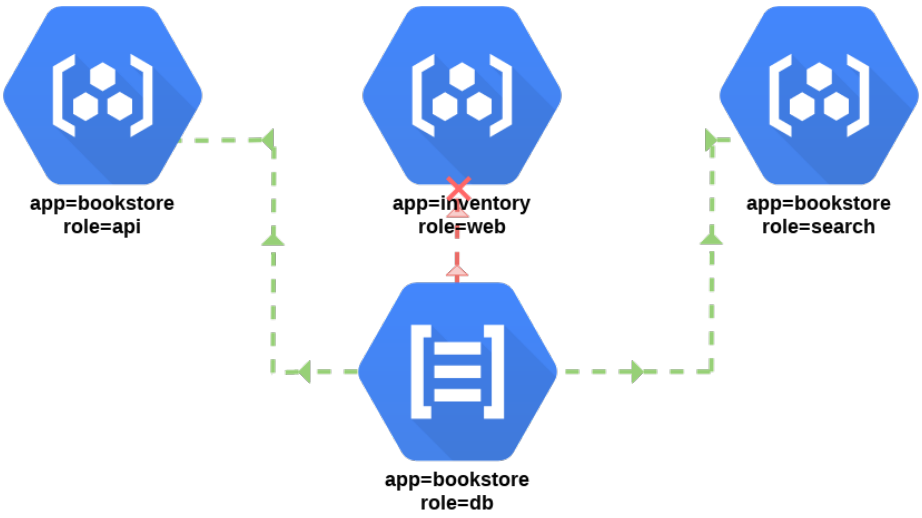
Network Policies

Deny all non-whitelisted egress



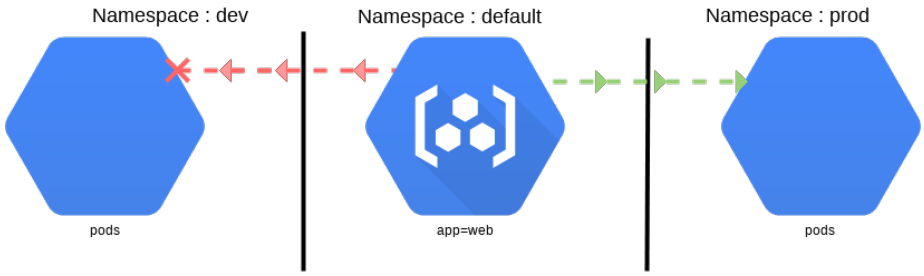
Network Policies

Allow egress to some pods



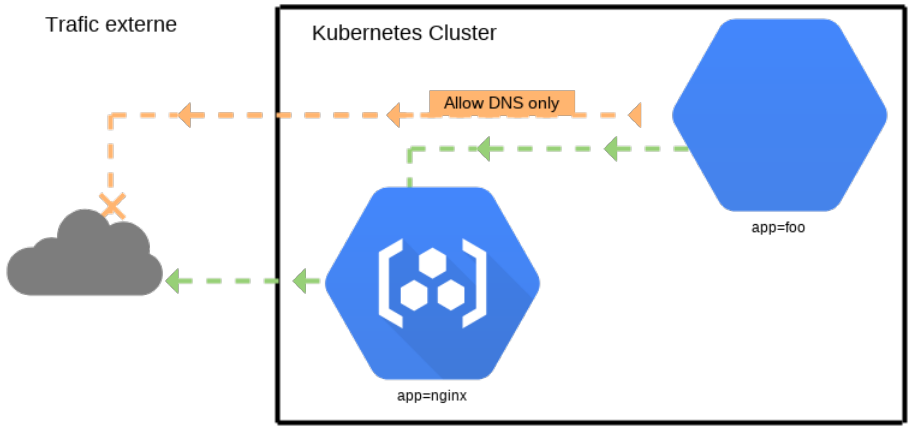
Network Policies

Allow egress to specific namespace



Network Policies

Allow egress to cluster only



Nettoyage de l'installation

- Pour nettoyer votre installation

```
# stop the cluster
./dind-cluster-v1.8.sh down

# remove DIND containers and volumes
./dind-cluster-v1.8.sh clean
```

Les principaux objets

Pods

Pods

- La plus petite unité avec laquelle on peut travailler
- On doit respecter le pattern : “Un processus par container” même dans le cas d’un Pod
- Une unique IP au sein d’un Pod pour tous les containers
- Ils sont démarrés sur le même noeud
- Le nombre de containers au sein d’un pod ne peut être modifié après lancement
- Les containers peuvent discuter via IPC ou des FS partagés
- En général un pod contient un container sauf Sidecar Pattern
- Ils ne sont pas exposés par défaut

Pods . Phase

- Pending : Requête accepté mais pod non encore créé
- Running : Le pod a été démarré sur un noeud avec au moins un container
- Succeeded : Tous les containers du pod sont bien démarrés
- Failed : Tous les containers du pod sont arrêtés avec a minima un container en échec
- Unknown : impossible de se connecter au pod pour lui demander son état

---

Pods . Probe

- Elles remplacent les Healtcheck de docker
- Il existe deux types de sonde : - livenessProbe - readinessProbe
- Il existe trois types d’action - ExecAction - TcpSocketAction - HttpGetAction

---

EXERCICE PODS

EXERCICE LABELS

EXERCICE PROBES

La réplication

- Pourquoi ?
- Comment ?

A quoi sert la réplication avec K8S ?

- Reliability (fiabilité)
- Load balancing
- Scaling

Dans quel cas ?

- Microservices-based applications
- Cloud native applications
- Mobile applications

Les différents types de réplication avec K8S

- Replication Controller
- Replica Sets
- Deployments

Les ReplicaSets

- Les **ReplicaSets** permettent de dupliquer sur le cluster le pod en question
- Permet de maintenir l'état désiré dans le Cluster
- Doit être manipulé directement si l'on veut faire sa propre orchestration (déconseillé)

ReplicationController // ReplicaSets

- Les ReplicaSet disposent de presque toutes les mêmes commandes que les ReplicationController
- Plus de possibilités avec les Selectors que les ReplicationController
- Ne permet d'utiliser la commande **rolling-update rc/...** qui est déclarative
- Il vaut mieux utiliser les **Deployment** pour configurer une réplication

---

EXERCICE REPLICASETS

Alternatives aux ReplicaSet

- Bare Pods : Equivant à revenir directement à "Docker run"
- Les Jobs pour les tâches qui doivent se terminer
- Les DaemonSets : chapitre à venir
- Les Deployment : chapitre à venir

Les Deployments

- On utilise désormais l'objet **Deployment** plutôt que **ReplicaSet**
- **Attention** ceci s'applique aux applications StateLess
- Un **Deployment** fournit à la fois la déclaration implicite des Pods et des ReplicaSets

---

EXERCICE DEPLOYMENTS

Les DaemonSets

- Ce contrôleur assure qu'un pod tourne sur tous les noeuds ou une sélection
- Lors d'un ajout de noeud, le pod est automatiquement déployé
- Les DaemonSets sont utilisés pour les applications systèmes comme: - un système de stockage comme Gluster, Ceph - un système de gestion de logs comme FluentD, logstash… - un système de monitoring comme Prometheus, CollectD…

Persistent Volumes

Persistent Volumes

- Aucune garantie d'avoir un pod fonctionnant indéfinimment sur le même noeud
- Le système de fichier persistant n'est pas géré par le cluster via ETCD
- Des disques locaux ou distants peuvent être montés

EXERCICE VOLUMES

Orchestration

Questions

- **Pod**
- **ReplicaSet**

- **Deployement**
- **Service**

Controller

Ils gèrent l'orchestration suivant les stratégies décidées pour maintenir et faire évoluer les fonctions applicatives - Stratégie de placement - Stratégie de mise à jour

---

Scheduler et strategie

- Kubernetes fournit un orchestrateur : **kube-scheduler**.
- Possibilité d'implémenter des "customs" schedulers.
- Stratégie « Un (et un seul) Pod sur chaque nœud » : **DaemonSet**
- Stratégie « N copies, un Pod par nœud si possible » : **ReplicatSet** (et Replication Controller), **Deployment** (via leur ReplicatSet)

Stratégie de placement par défaut

- Trouver là où il y a de la place sur des nœuds éligibles
- Répartir les pods correspondant à un même service sur plusieurs nœuds

---

Stratégie node Selector

Au préalable, il faut labelliser les noeuds :

```
kubectl label nodes k8s-foo-node1 disktype=ssd
```

Puis on peut choisir où lancer le pod :

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

Build-in node Labels

Labels pré-configurés mais **dépendant** de la plateforme.

```
kubernetes.io/hostname
failure-domain.beta.kubernetes.io/zone
failure-domain.beta.kubernetes.io/region
beta.kubernetes.io/instance-type
beta.kubernetes.io/os
beta.kubernetes.io/arch
```

Deux types d'affinités

Successeur de **nodeSelector**

- **node affinity**
- **inter-pod affinity/anti-affinity**

Stratégie node affinity

Plus expressif que **nodeSelector** sur base de **regexp**

- *requiredDuringSchedulingIgnoredDuringExecution* (hard)
- *preferredDuringSchedulingIgnoredDuringExecution* (soft)

Stratégie node affinity

Dans un futur plus ou moins proche

- *requiredDuringSchedulingRequiredDuringExecution* (hard)

Exemple

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/e2e-az-name
            operator: In
            values:
```

```
          - Europa-North
          - Europa-South
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 1
      preference:
        matchExpressions:
        - key: disktype
          operator: In
          values:
          - ssd
containers:
- name: with-node-affinity
  image: k8s.gcr.io/pause:2.0
```

Les opérateurs de nodeSelectorTerms

- In
- NotIn
- Exists
- DoesNotExist
- Gt
- Lt.

Utiliser **NotIn** et **DoesNotExist** pour **anti-affinity**

Règles de mix

Si **nodeSelector** et **nodeAffinity** les deux doivent être **vrais**

Si plusieurs **nodeSelectorTerms** avec **nodeAffinity** alors au moins un **nodeSelectorTerms** vrai

Si plusieurs **matchExpressions** avec **nodeSelectorTerms** alors toutes les **matchExpressions** doivent être vraies

---

(Anti)Affinity

Un Pod peut être schédulé ou non sur un noeud en fonction des pods qui sont déjà présents ou non.

topologyKey : clé du noeud que K8S utilisera pour définir une topologie des pods

---

Exemple simple

Pod sur la même zone mais sans avoir d'autres pods à côté avec le label **security=S1**

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - S1
        topologyKey: failure-domain.beta.kubernetes.io/zone
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security
              operator: In
              values:
              - S2
          topologyKey: kubernetes.io/hostname
  containers:
  - name: with-pod-affinity
    image: k8s.gcr.io/pause:2.0
```

Exemple : colocation sur le même noeud // REDIS

Chacun des trois réplicas redis doit se trouver sur un noeud différent

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-cache
spec:
  selector:
    matchLabels:
```

```
      app: store
  replicas: 3
  template:
    metadata:
      labels:
        app: store
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
                operator: In
                values:
                - store
            topologyKey: "kubernetes.io/hostname"
      containers:
      - name: redis-server
        image: redis:3.2-alpine
```

Exemple : colocation sur le même noeud // WEBSERVER

Chacun des trois réplicas des nginx doit être avec un unique REDIS

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-server
spec:
  selector:
    matchLabels:
      app: web-store
  replicas: 3
  template:
    metadata:
      labels:
        app: web-store
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: app
```

```
          operator: In
          values:
          - web-store
      topologyKey: "kubernetes.io/hostname"
  podAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
        - key: app
          operator: In
          values:
          - store
      topologyKey: "kubernetes.io/hostname"
containers:
- name: web-app
  image: nginx:1.12-alpine
```

EXERCICE ORCHESTRATION

TAINTS & TolerationS

Stratégie de placement de Pods : Taints & Tolerations

- Notion de boules puantes
- Une **Taint** permet à un noeud de refuser qu'un Pod soit schedulé si le pod ne possède les **Toleration** correpondantes
- **Taint** et **Toleration** consiste en une pair de Key/Value plus un "effect" (lorsque Key=Value)

Stratégie de placement de Pods : Taints & Tolerations

Par exemple : on définit 3 "Taints" sur le Node 1 :

```
$ kubectl taint nodes node1 key1=value1:NoSchedule // "NoSchedule" = effect
$ kubectl taint nodes node1 key1=value1:NoExecute  // "NoExecute" = effect
$ kubectl taint nodes node1 key2=value2:NoSchedule
```

Stratégie de placement de Pods : Taints & Tolerations

On définit pour un Pod les **Tolerations** suivantes :

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoExecute"
```

Le Pod ne sera pas Schedulé car il n'y a pas de **Toleration** correpondant à la 3eme **Taint** du Noeud

Présentation

Kubernetes

Objectifs

- Connaître l'origine et l'historique du projet
- Connaître l'architecture de k8s

- Kubernetes vient du mot grec **timonier**
- Le logo représente l'idée de "pilote de conteneurs"

K8S

Le chiffre 8 représente le nombre de caractères entre la première et dernière lettre.

Borg

- Réécriture en GO du système développé en interne chez Google
- Google Search, Maps, Gmail, Youtube reposent sur le système **Borg**

CNCF

Partenariat de Google & Fondation Linux pour créer la **Cloud Native Computing Foundation**

Kubernetes version 1.0 sortie en Juillet 2015 et depuis géré par la CNCF

Rôle d'un orchestrateur

- Le déploiement
- La montée en charge
- Le cycle de vie des conteneurs

Quelques chiffres et points clés...

- Implémenté dans 40% des environnements Docker (Datadog)
- Forte croissance : un des projets les plus en vue de 2018/2019
- Plus de 1500 contributeurs sur Github
- Fortement adopté en production par les entreprises
- Intégration par les fournisseurs de cloud : EKS, AKS, GKE…
- Intégration par les éditeurs : Docker EE, Openshift, CoreOS… RancherOS
- Conteneur-agnostique (Docker ou RKT)
- Adapté pour les architectures logicielles en microservices

---

fonctionnalités

- Container grouping using pod
- Self-healing
- Auto-scalability
- DNS management
- Load balancing
- Rolling update or rollback
- Resource monitoring and logging

RBAC

NEEDS

Have multiple users with different properties, establishing a proper authentication mechanism.

Have full control over which operations each user or group of users can execute.

Have full control over which operations each process inside a pod can execute.

Limit the visibility of certain resources of namespaces.

The key to understanding

RBAC in Kubernetes

Subjects

The set of users and processes that want to access the Kubernetes API.

Resources

The set of Kubernetes API Objects available in the cluster. Examples include Pods, Deployments, Services, Nodes, and PersistentVolumes, among others.

Verbs

The set of operations that can be executed to the resources above.

Different verbs are available (examples: get, watch, create, delete, etc.),

---

Understanding RBAC API Objects

Roles

Definition of the permissions for each Kubernetes resource type

---

RoleBindings

Definition of what Subjects have which Roles

Users and ... ServiceAccounts

Users

These are global, and meant for humans or processes living outside the cluster.

ServiceAccounts

These are namespaced and meant for intra-cluster processes running inside pods.

ClusterRoles

ClusterRoleBindings

Give permissions for

- non-namespaced resources like nodes
- permissions for resources in all the namespaces of a cluster
- permissions for non-resource endpoints like /healthz

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

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: example-clusterrolebinding
subjects:
- kind: User
  name: example-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: example-clusterrole
  apiGroup: rbac.authorization.k8s.io
```

Les différents services

L'objet Service peut être vu naïvement comme un LoadBalancer entre pods

Il s'agit surtout de proposer plusieurs points d'accès en fonction du type de ressource.

Trois types différents

- ClusterIP
- NodePort
- LoadBalancer

Ainsi que la notion de *Ingress*

ClusterIP

Type par défaut

Une IP virtuelle est fourni par K8S pour l'accès entre Pods

Cluster IP

- Pas d'accès externe. Uniquement interne
- Accessible via un ReverseProxy **kube-proxy**

  kubectl proxy –port=8080

Accès via kube-proxy

http://localhost:8080/api/v1/proxy/namespaces//services/:/

http://localhost:8080/api/v1/proxy/namespaces/default/services/my-internal-service:http/

Quand utiliser Kube Proxy ?

- Debugger des services directement depuis votre poste
- Autoriser des traffics externes (Client Mysql…)
- Afficher des dashboards internes
- Attention à ne pas exposer sur Internet !

NodePort

Traffic extérieur

Approche la plus basique pour exposer un pod vers l'extérieur.

Ouvre un port sur tous les nodes

Champ spécifique

- **nodePort** est optionnel
- Il permet de choisir le port à exposer
- Il est conseillé de laisser K8S le fixer lui-même

Inconvénients de NodePort

- Seulement un accès d'un service pour un port donné
- Plage restreinte de 30000 à 32767
- Si les IPs de vos nodes changent, vous devez gérer ceci en amont

---

Cas d'usage de NodePort

Usage possible mais peu recommandé en production

- Utile pour des demos
- Potentiellement utilisable avec un ReverseProxy type Traefik

---

LoadBalancer

Cas d'usage de LoadBalancer

Exposer directement un service sur Internet

- Tout le traffic sera redirigé vers le service
- Pas de filtering, routing…
- Tout type de traffic est routé : HTTP, gRPC, UDP, WebSocket…
- S'utilise avec les Cloud providers

Inconvénients de LoadBalancer

Le coût sur le Cloud Public

- Chaque service va exposer un Load Balancer avec sa propre IP
- https://cloud.google.com/compute/docs/load-balancing/network/.

Service avec Selecteur

- Il permet de renvoyer le trafic vers un ensemble de pods présents dans le même namespace.
- L'identification des pods vers lesquels diriger le trafic est basée sur des labels et des sélecteurs.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort
  selector:
    app: nginx
    version: red
  ports:
    - port: 80
    targetPort: 80
```

Service sans Selecteur

Il permet de renvoyer le trafic vers des pods présents dans un autre namespace ou vers des services extérieurs au cluster K8S.

```
apiVersion: v1
kind: Service
metadata:
  name: ws-external
spec:
  ports:
    - port: 80
    targetPort: 80
---
apiVersion: v1
```

```
kind: Endpoints
metadata:
  name: ws-external
subsets:
  addresses:
    - ip: xxx.xxx.xxx.xxx
    ports:
      - port: 80
```

Service Headless

- Un service headless consiste à désactiver le clusterIP.
- La résolution DNS interne renverra l'adresse IP de chaque pod.
- Ce type de service est principalement utilisé pour demander au serveur DNS interne de Kubernetes de renvoyer l'adresse IP des pods à la place d'une IP de loadbalancer.

```
apiVersion: v1
kind: Service
metadata:
  name: pgpool
spec:
  ports:
    - name: pg
      port: 1234
    ClusterIP: None
    selector:
      app: pgpool
```

Ingress

PAS UN SERVICE MAIS...

SMART ROUTER

- Il gère les entrypoints pour votre cluster
- Il existe de nombreux Ingress Controller
- Par défaut GKE démarrera un HTPP(s) Load Balancer

Cas d'usage de Ingress

Le plus puissant mais le plus compliqué

- Volonté d'exposer différents services sous la même IP
- Nécessité d'utiliser le même protocole L7 (typiquement HTTP)
- Ne payer qu'un seul LoadBalancer sur GKE tout en profitant avec Nginx de SSL, Auth, Routing…
- Peut devenir un point de contention // LoadBalancer en option précédente.

EXERCICE SERVICES

DEMO

https://github.com/sozu-proxy/sozu-demo/tree/master/kubernetes-using-tube-cheese

Figure 4: logo_treeptik

Kubernetes

Figure 5: Logo_k8s

Formateur

Consultant DEVOPS - Treeptik

Sommaire

Stratégies de déploiement

Plusieurs types

De nombreuses stratégies sont possibles mais elles ont un coût différent

- Sur l'infrastructure
- Sur le consommateur

Les différentes stratégies

- Suppression/création
- Rampe
- Blue/green
- Canary
- A/B Testing
- Shadow

Recreate

- La version A est supprimée
- La version B est démarrée

Recreate

Avantages - Facile à mettre en oeuvre - Etat de l'application entièrement neuf

Inconvénients - Coupure du consommateur

Ramped - RollingUpdate - Incremental

Changement lent et progressif des instances A par des instances B

En fonction de l'orchestrateur, il est possible de changer : - le parallélisme - le nombre d'instance max indisponibles - le nombre de nouvelles instances B à ajouter (exemple 130%)

Ramped - RollingUpdate - Incremental

Ramped - RollingUpdate - Incremental

Avantages - Facile à mettre en oeuvre - Versions lentement mises à jour sans interruption de Services - Parfait pour les applications StateLess - Peut convenir **toutefois** aux applications StateFull !

Inconvénients - Rollback peut prendre du temps - Difficulté de supporter différentes API en // - Pas de contrôle sur le traffic.

Blue / Green

> Après déploiement de la nouvelle version et tests de conformitétout le traffic est redirigé dessus

---

Blue / Green

Blue / Green

Avantages - Rollback instantané - Pas de collision de version tout est changé en one shot !

Inconvénients - Coûteux car nécessite le double de ressources - Besoin de jouer des tests sur la version nouvellement déployée - La gestion d'applications avec états peut être compliquée

---

EXERCICE BLUEGREEN

Canary

> Même famille que le Blue/Green Graduellement traffic redirigé de la version A à la version B

Par exemple : - 90% des requêtes vont à la version A - 10% des requêtes vont à la version B

> Bien pratique quand on ne peut tester le B/G

Canary

Canary

Avantages - Version disponibles seulement pour un sous-ensemble d'utilisateurs - Parfait pour tester le monitoring et des métriques sur la nouvelle version - Rollback rapide et facile

Inconvénients - Même que Blue/Green (attention au cas des tests)

A/B Testing

> Redirection des utilisateurs sous conditions vers une des deux versions
>
> Permet de prendre des décisions business sur base de statistiques

---

A/B Testing

conditions: - Cookies - Query Parameters - Geo location - Browser (version, os…) - Langue

---

A/B Testing

A/B Testing

Avantages - Plusieurs versions disponibles en parallèle - Contrôle totale de la distribution du traffic - Rollback rapide et facile

Inconvénients

- Nécessite un LB dit “intelligent”
- Difficile de débogguer les erreurs pour une session donnée
- Besoin d’utiliser de distribution tracing (zipkin, sleuth…)

A/B Testing

Redirection des utilisateurs sous conditions vers une des deux versions

Permet de prendre des décisions business sur base de statistiques

---

Shadow

Duplication des applications et des flux réseaux. Chaque version recevant la copie du flux.

Shadow

Shadow

Avantages - Test de la nouvelle application avec un vrai traffic - Aucun impact sur l’utilisateur - Aucun changement jusqu’à obtenir les garanties en terme de stabilité et performance

Inconvénients

- Coûteux car nécessite le double de ressources
- Limite de l’exercice quand on doit écrire en base de données…
- Compliqué à mettre en oeuvre (écriture…)
- Nécessite des Mocks pour certains services –> effet de bords

---

Les Volumes

Applications Statefuls mais pas que…

Problématique

- Les containers sont éphémères par leur conception même.
- Les Volumes permettent de sauvegarder les données qui méritent d'être persistées.

Pourquoi les utiliser ?

- Communication / Synchronisation entre pods
- Découpler les données du cycle de vie d'un pods et de ses containers
- Point de montage avec le système de fichiers local

Les Différents types de Volume

- Locaux aux noeuds : **emptyDir** ou **hostPath**
- Partage de fichiers : **nfs**
- Cloud Provider : **gcePersistentDisk**, **awsElasticBlockStore**…
- Système distribué : **glusterfs**, **cephfs**…
- Spéciaux comme **secret**, **gitRepo**

---

https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes

---

emptyDir

- Vide à la création
- Suit le cycle de vie d'un pod
- Pas de suppression si un des containers du pod crash
- Peut-être monté en RAM (tmpfs)
- Peut-être montés n-fois dans n-containers d'un pod

EXERCICE VOLUME_INTRA_PODS

---

hostPath

Monter une ressource de l'hôte dans le Pod

Types de HostPath

- Directory
- File
- Socket
- CharDevice
- BlockDevice

Cas d'utilisation

- Monitoring de la machine hôte
- Socket Docker
- Monter un GPU :)

Exemple ~~~ apiVersion: v1 kind: Pod metadata: name: test-pd spec: containers: - image: k8s.gcr.io/test-webserver name: test-container volumeMounts: - mountPath: /test-pd name: test-volume volumes: - name: test-volume hostPath:
path: /data ~~~

---

objets

- PersistenceVolume (PV)
- PersistenceVolumeClaim (PVC)
- StorageClass

Persistence Volume

- **Abstraction** de la gestion des volumes
- Stockage provisionné statiquement ou dynamiquement (via **StorageClass**)
- Nombreux **types** Différents (NFS, gcePersistentDisk, Ceph…)

Persistent Volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: "pv"
spec:
  storageClassName: manual
  capacity:
    storage: "1Gi"
  accessModes:
    - "ReadWriteOnce"
  hostPath:
    path: /data/pv
```

```
kubectl get pv
NAME    CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS    CLAIM    STORAGECLASS  REASON   AGE
pv      1Gi      RWO         Retain        Available         manual              5m
```

Persistent Volume Claim

- Une demande de stockage
- Consomme un Persistent Volume
- Spécifie des contraintes supplémentaires
- Création d'un binding entre PVC et PV

Persistent Volume Claim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: requetevolume
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

```
kubectl create -f pvc.yml
persistentvolumeclaim "requetevolume" created

kubectl get pvc
NAME         STATUS   VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  AGE
requetevolume  Bound    pv      1Gi      RWO         manual      13s
```

Création container Mongo

```
apiVersion: v1
kind: Pod
metadata:
  name: mongo
spec:
  containers:
    - name: mongo
      image: mongo:3.6
      volumeMounts:
      - mountPath: /data/db
        name: data-db
  volumes:
    - name: data-db
      persistentVolumeClaim:
        claimName: requetevolume
```

```
kubectl create -f mongo.yml
pod "mongo" created

ls /data/pv
mongo.yml  pvc.yml  pv.yml
root@user1:~/volumes# ls /data/pv/
collection-0-7031728124031270860.wt index-1-7031728124031270860.wt _mdb_catalog.wt storage.
collection-2-7031728124031270860.wt index-3-7031728124031270860.wt mongod.lock    WiredTige
diagnostic.data               journal                 sizeStorer.wt   WiredTigerLAS.wt Wired
```

EXERCICE VOLUMECLAIMWITHMONGO

EXERCICE VolumeClaimSharing