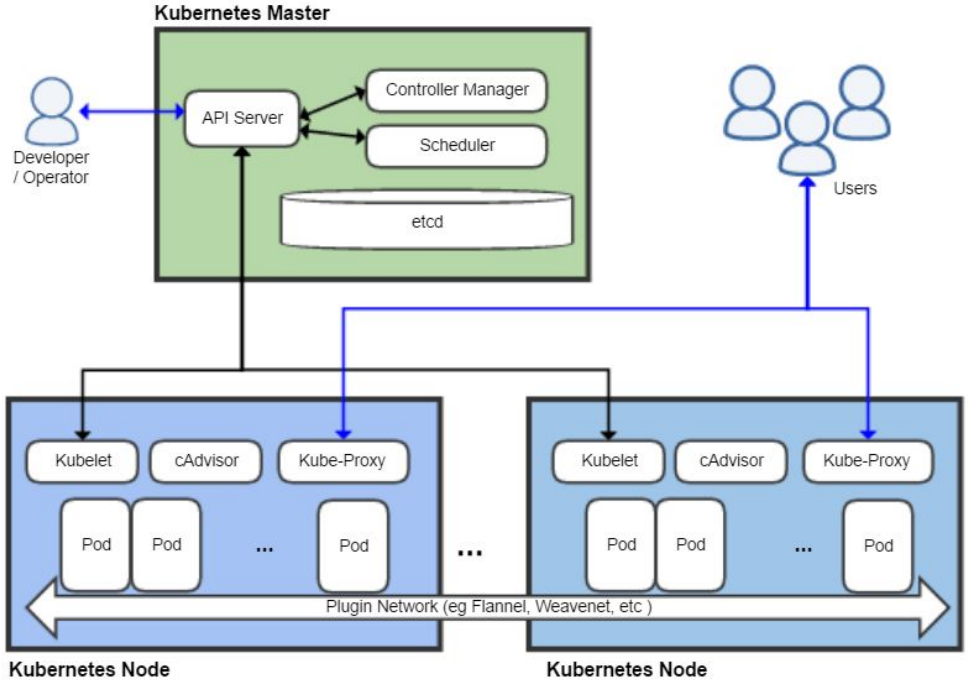
Kubernetes

Christophe Dufour

Architecture



https://kubernetes.io/fr/docs/concepts/overview/components/

Schéma général

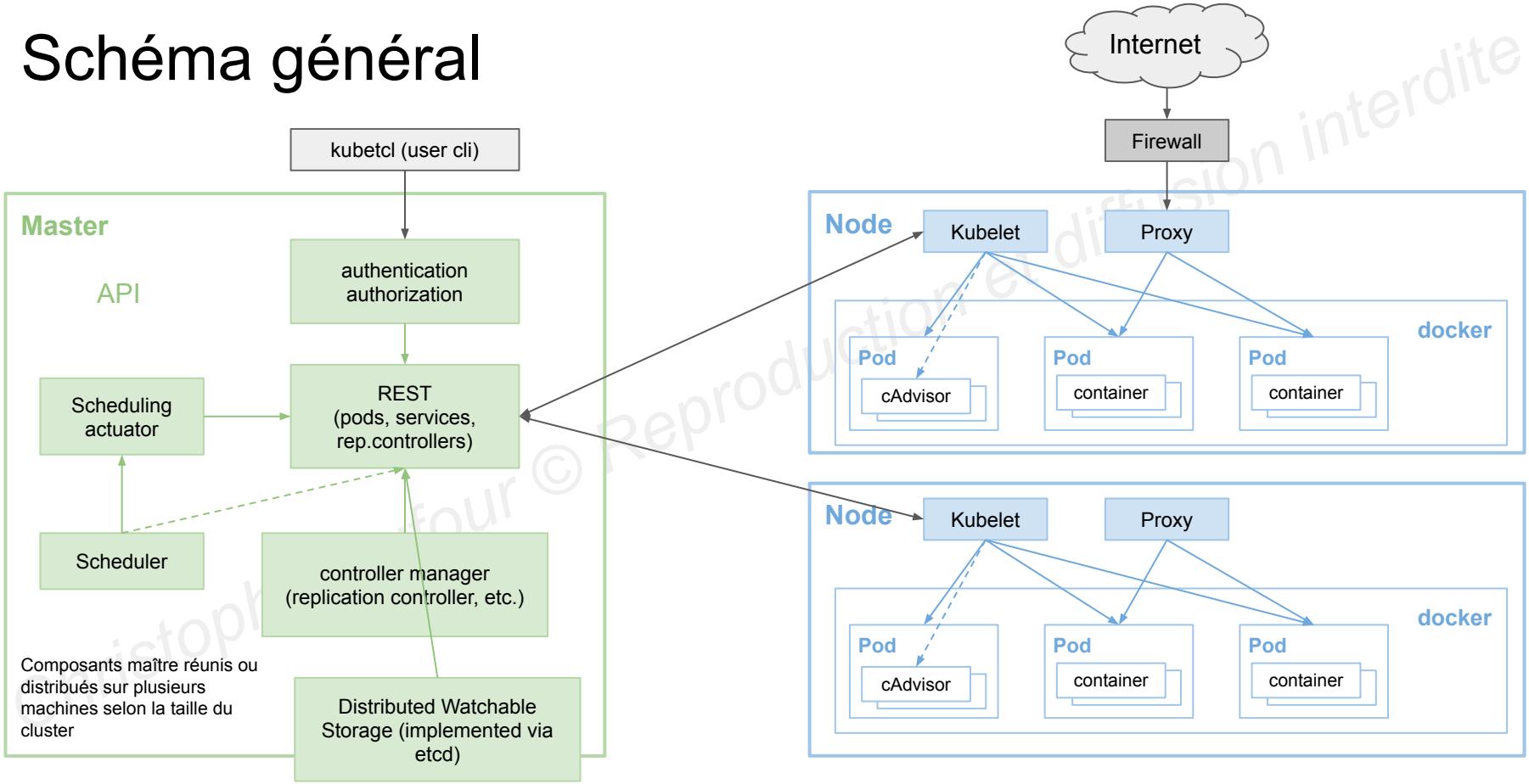
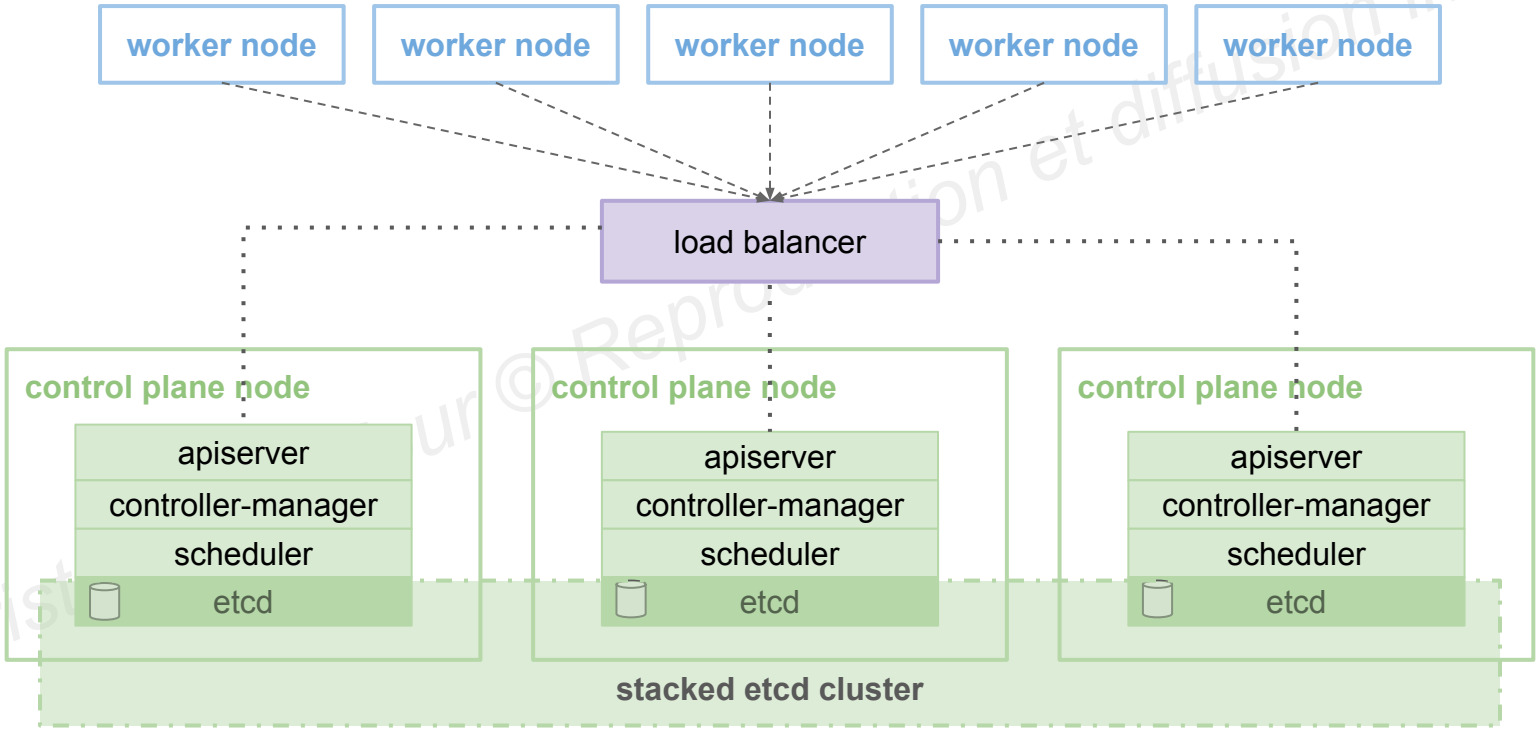
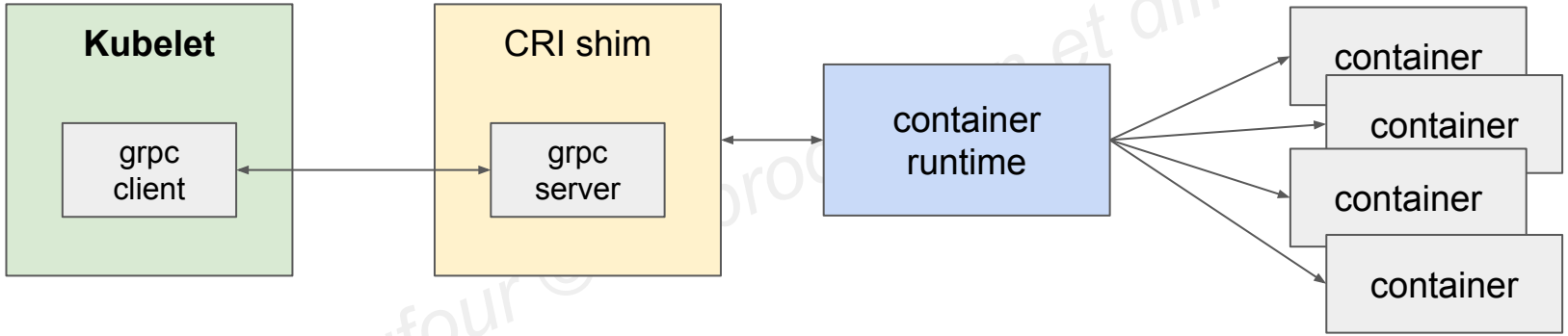


Schéma général - Cluster HA (haute disponibilité)



Kubelet et container runtime



https://kubernetes.io/fr/docs/concepts/overview/components/

Introduction

- Plateforme de déploiement d'applications conteneurisées
- Projet initial de Google (Borg) puis Cloud Native Computation Foundation
- Open source
- Version 1: juillet 2015
- Solution idéale pour la gestion d'un grande quantité de **microservices**

Retour sur les microservices

- Application = assemblage de “petits” services spécialisés
- Microservice:
  - processus/périmètre métier (ex: gestion d’une commande, validation d’un compte utilisateur, etc.)
  - problèmatique transversale (ex: identification, monitoring de ressource)

Granularité plus fine

- Mise à échelle (scaling) horizontale facilitée
- Plus forte résilience aux pannes
- Plus grande fréquence de déploiement

Complexité de déploiement

- Nombreuses briques à déployer => automatisation nécessaire
- Automatisation complexe
  - diversité des services
  - configurations multiples: chaque microservice est en relation avec d'autres services

Isolation des conteneurs

- Couche d’isolation supplémentaire à gérer
- Communication plus complexe (notamment entre les machines)
- Persistence des données à gérer (survie au conteneur)

Dynamicité des services

- Plus de services sur plus de technologies
  - dysfonctionnement potentiel accru
  - évolutions/changements de version plus fréquents
- Un microservice peut être arrêté et remplacé plus souvent
  - question de la robustesse des clients
  - problèmes de reconfiguration

Supervision

- Nombreux microservices à surveiller
- Localisation des logs
- Exploitation des informations
  - levée d’alertes
  - réactions automatisées

Orchestrateurs

- Objectif: faciliter la gestion de nombreux déploiements
  - indispensable dans le cadre d'une architecture à base de microservices
- Fournissent des services support pour faciliter la gestion des microservices

Quelques orchestrateurs de conteneurs

- **docker-compose**: surcouche à Docker. Simple, installation et prise en main rapide, capacitées limitées
- **Docker Swarm**: gestion de cluster de machines, scaling, intégrée à Docker
- **Mesos/Marathon/DCOS**: gestion de (gros) clusters de machines, scaling, gestion fine des ressources, templates et bibliothèques de déploiement
- **Kubernetes (k8s)**: nombreuses fonctionnalités. Clusters, scaling, templates, ressources, stockage, blibliothèques (via helm), contrôle d'accès

Kind - Kubernetes in docker

```
stagiaire@opusidea:~$ kind create cluster
Creating cluster "kind" ...
 ✓ Ensuring node image (kindest/node:v1.21.1) 
 ✓ Preparing nodes 
 ✓ Writing configuration 
 ✓ Starting control-plane 
 ✓ Installing CNI 
 ✓ Installing StorageClass 
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Not sure what to do next?   Check out https://kind.sigs.k8s.io/docs/user/quick-start/
```

K8S - ressources

- Terme générique désignant les “éléments” ou “objets” gérés par k8s
  - conteneurs et assimilés
  - stockage (volumes et ressources connexes)
  - réseau (services, load balancer)
- Système de labels et de selectors
  - clé/valeur, utilisable comme label ou comme sélecteur

```
selector:
  app: server
```

  - expressions de sélection

```
environment in (production, qa)
tier not in (frontend, backend)
```

K8S - utilisation

- **kubectl**: CLI pour gérer le cluster
- Utilise de nombreux fichiers YAML permettant de décrire les différentes ressources à mettre en place
- Exemple: kubectl create -f simpleweb-dpl.yml

K8S - exemple de fichier de ressource

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: simpleweb-depl
spec:
  replicas: 3
  selector:
    matchLabels:
      app: simpleweb
  template:
    metadata:
      name: simpleweb-pod
      labels:
        app: simpleweb
    spec:
      containers:
        - name: simpleweb
          image: opusidea/simpleweb:v2
```

Déploiement de conteneurs

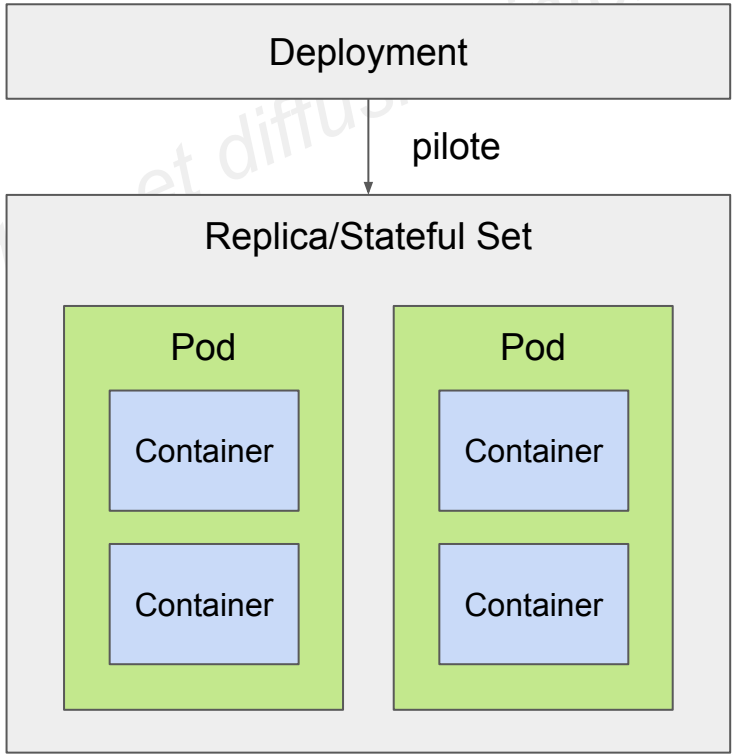
- Fonctionnalité de base
- Impératif
  - sous forme d'une ligne de commande avec options
- Déclaratif
  - sous forme d'un fichier de configuration (.yml)
  - limite le volumen de script pour l'automatisation
  - gestion de la configuration (envrionnement, fichiers)
- Redimensionnement
  - gestion de plusieurs conteneurs ayant même image/même config
  - gestion du nombre de copies (replicas) à la hausse (scaling in) et à la baisse (scaling out)

Déploiement et supervision

- Supervision simple: état du conteneur
  - en construction, en matche, arrêté, etc.
- Supervision avancée
  - le service est-il fonctionnel ? (un service peut être “running” tout en renvoyant 503)
  - *health check*: code exécuté pour vérififer le bon fonctionnement du service
- Réaction
  - quelle mesure prendre en case de défaillance ?

K8S - conteneurs, pods, sets

- Encapsulation des conteneurs
- Pods: groupe de conteneurs
  - partage de ressources
  - démarrage et suppression simultanés
- Contrôleurs
  - ensemble de pods
  - surveillance, redémarrage
  - Replica/Stateful/Daemon Set
  - Deployments



K8S - configuration

- “à la Docker”: ligne de commande, variables d’environnement
- ConfigMap
  - dictionnaire clés-valeurs
  - visible comme un répertoire
    - clé == fichier
    - valeur == contenu
  - visible via des variables d’environnement
  - gestion distincte des deployments
  - peut être partagé
- Secrets
  - similaires à ConfigMap mais sécurisé

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: game-demo
data:
  player_initial_lives: "3"
  ui_properties_file_name: "user-interface.properties"
  game.properties: |
    enemy.types=aliens,monsters
    player.maximum-lives=5
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

K8S - ConfigMap

- namespaced
- référencable par tout pod

```yaml
apiVersion: v1
kind: ConfigMap
  metadata:
    name: super-config
    namespace: default
  data:
    NUM_LIVES: "3"
    PORT: "9090"
```

```yaml
...
spec:
  containers:
  - name: super-container
    image: opusidea/super-image:0.1
    ports:
    - containerPort: 8080
  envFrom:
  - configMapRef:
      name: super-config
```

Clés-valeurs projetées en tant que variables d'environnement dans super-container
NUM_LIVES=3
PORT=9090

K8S - ConfigMap - ligne de commande

```
$ kubectl create configmap test --dry-run=client --from-literal=lives=3 -o yaml
apiVersion: v1
data:
  lives: "3"
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: test
```

K8S - supervision applicative

- Health checks (probes)
  - Readyness / Liveness
  - Différents types de scondes
    - commandes exécutées dans le conteneur
    - requête http
    - vérification port TCP
  - succès: code de retour = 0
- A utiliser avec restartPolicy

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - args:
    - /server
    image: k8s.gcr.io/liveness
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: X-Custom-Header
          value: Awesome
      initialDelaySeconds: 15
      timeoutSeconds: 1
    name: liveness
```

K8S - batchs

- Processus à durée de vie limitée
- A gros grain, certaines problématiques similaires aux services
  - supervision
  - relance (partielle) en cas d’échec
- Nécessite parfois un accès privilégié à certains services

K8S - jobs

- Possibilité d'avoir plusieurs pods: parallèlisme
- Redémarrage en cas d'échec avec nombre d'essais limités
  - restartPolicy: OnFailure ou Always
- Cron Jobs
- TTL Controllers: pour nettoyer les ressources
- Pas de système de type Spring Batch pour redémarrer un job à mi-chemin

```yaml
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
```

K8S - stockage

- Partage de données entre différents conteneurs
- Persistence des données au-delà de la vie des conteneurs
- Utilisation de systèmes de stockage externes fiables

K8S - volumes

- Système de volumes
  - proche de celui de Docker dans l'utilisation
  - plus générique
- Volume lié à in pod
  - même durée de vie
  - montable par les conteneurs du pod (possiblité de montage partagé)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

K8S - stockage persistant

- Données à durée de vie plus longue que celle du pod
  - **PersistentVolume**: données
    - En général, hébergé par un systèmes tiers (Ceph, NFS)
  - **PersistentVolumeClaim** (PVC): demande d'utilisation des données (binding)
  - Utilisation dans un pod: création d'un volume basé sur un certain PVC
- Peut être créé à la demande lors d'un déploiement de Stateful Set

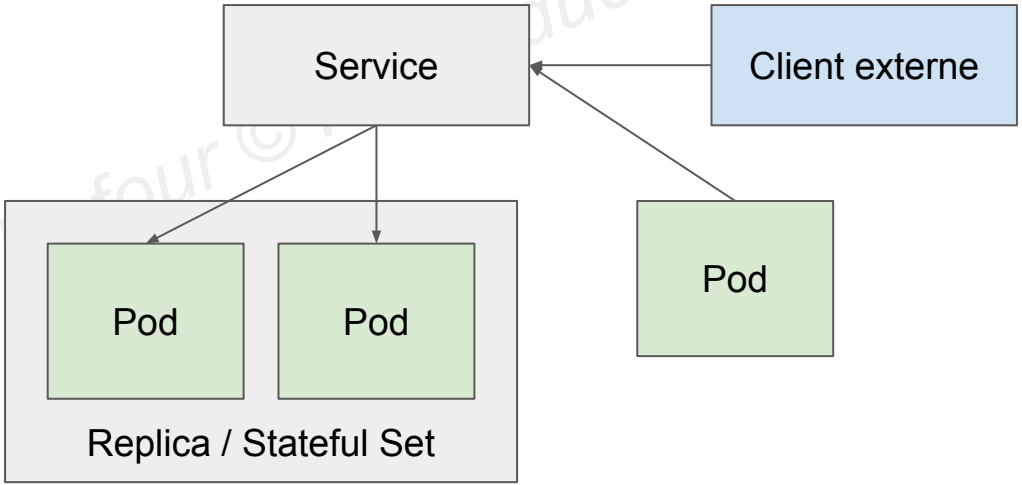
```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongo-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

K8S - réseau et API Management

- Pouvoir questionner l’orchestrateur pour savoir qui fait quoi
- Fourniture de services réseau
  - IPs virtuelles
  - DNS
  - intégration de reverse proxies / load balancers

K8S - services

- Points d’accès virtuels pour un Replica / Stateful Set
- Peut intégrer un load balancer
- Permet d’offrir un point d’accès depuis l’extérieur du cluster



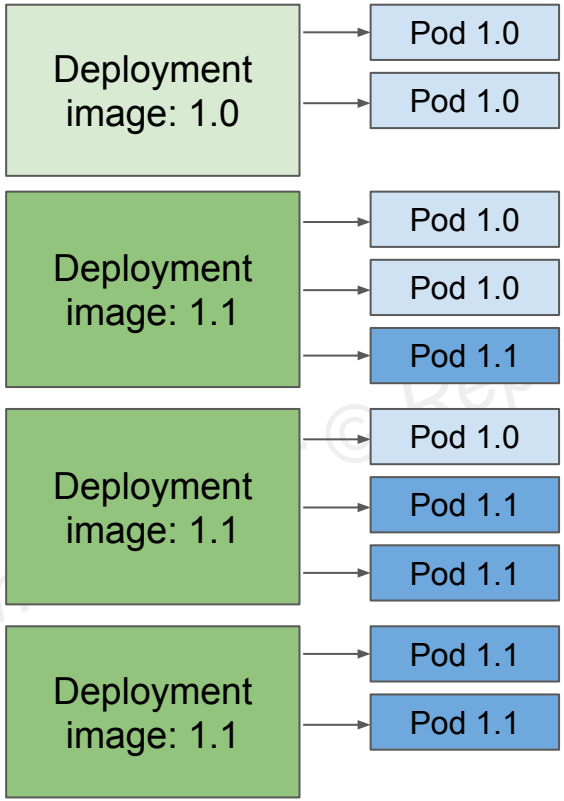
K8S - DNS

- Plugin pour déployer un DNS interne
  - Adresse pour chaque pod
  - Adresse pour chaque service

K8S - sécurité et isolation

- Organisation de services en espaces logiques
- Possibilités d’isolation réseau
- Utilisateurs multiples et droits associés
- Limites imposées par pod/set/namespace

K8S - Déploiement RollingUpdate




```
Spec:
  replicas: 2
    stategy:
      rollingUpdate:
        maxSurge: 1
        maxUnavailable: 1
    type: RollingUpdate
```

K8S - namespaces

```
kubectl get pod -n=exo5
```

```
kubectl config set-context --current --namespace=exo5
```

$ kubectl create ns demo
namespace/demo created

$ kubectl.exe apply -f tmp/demo/php-apache.yml -n=demo

```
C:\Users\chris\.kube
λ cat config
apiVersion: v1
clusters:
- cluster:
    certificate-authority: C:\Users\c
    extensions:
    - extension:
        last-update: Thu, 20 May 2021
        provider: minikube.sigs.k8s.i
        version: v1.17.0
      name: cluster_info
    server: https://192.168.99.101:84
  name: minikube
contexts:
- context:
    cluster: minikube
    extensions:
    - extension:
        last-update: Thu, 20 May 2021
        provider: minikube.sigs.k8s.i
        version: v1.17.0
      name: context_info
    namespace: exo5
```

K8S - utilisateurs

- Deux types: humain vs pod (User vs Service Account)
- Authentification: mécanismes variés, possibilité d’injection par secret pour les Service Accounts
- Systèmes d’autorisation variés: RBAC, ABAC, etc

K8S - service account

- Fournit une identité aux micro-services
- Associé à certains droits, privilères, restrictions
- Associable à un pod (pod spec), les micro-services internes au pod seront affectés par le service account
- Il existe un SA par défaut (namespace default)
- Chaque SA est associé à un **secret**

K8S - isolation réseau

- Network Policies
- Permet de restreindre l’accès à certains pods
  - traffic entrant et/ou sortant
  - pods/namespaces/ips

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
---
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

K8S - limitations de ressources

- Possibilité de limiter les ressouces associées à un pod
- Bonne pratique: toujours fixer des limites CPU/RAM

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: cpu-demo
spec:
  containers:
  - name: cpu-demo-ctr
    image: vish/stress
    resources:
      limits:
        cpu: "1"
      requests:
        cpu: "0.5"
    args:
    - -cpus
    - "2"
```