

# Orchestration de conteneurs: Illustration avec Kubernetes

Emmanuel Coquery

Fiabilité et sécurité des Applications - Master 2 Technologies de l'Information et Web

Avril 2023

# Survol

- Retour sur certaines problématiques autour des microservices
- Orchestrateurs
- Focus sur Kubernetes

# Retour sur les microservices

- Application = assemblage de « petits » services
- Microservice :
  - processus/périmètre métier (gestion d'une commande, des personnel, etc)
  - ou préoccupation transverse (authentification, audit/supervision, persistance des données, équilibrage de charge)

# Granularité plus fine

- Meilleure possibilités de passage à l'échelle:
  - plus de flexibilité pour combiner le passage à l'échelle horizontal suivants les 3 axes classiques: répartir les fonctionnalités, paralléliser les calculs indépendants, distribuer les traitements selon les données
- Impact des pannes *potentiellement* réduit
- Plus de déploiements

# Complexité de déploiement

- Nombreuses briques à déployer → automatisation nécessaire
- Automatisation complexe
  - Plus de diversité
  - Plus de configuration: chaque microservice s'adresse à de nombreux autres

# Isolation des conteneurs

- Une couche d'isolation supplémentaire à gérer
  - mise en oeuvre de la communication plus complexe
    - en particulier entre machines
- gestion des données à conserver après disparition du conteneur

# Dynamicit  des services

- Plus de services sur plus de technologies:
  - plus de potentiel de dysfonctionnement
  -  volutions plus r guli res
- Un microservice peut  tre arr t  et remplac  plus souvent
  - robustesse des clients
  - probl mes de reconfiguration

# Supervision

- Nombreux microservices à surveiller
- Localisation des logs
- Exploitation des informations
  - levées d'alertes
  - réactions automatisées

# Orchestrateurs

- Objectif: aider à la gestion de nombreux déploiements
  - indispensables 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 (mise en place et appréhension), capacités assez limitées
- Docker Swarm: gestion de cluster de machines, scaling, intégré à Docker
- Mesos/Marathon/DCOS: gestion de (très gros) clusters de machines, scaling, gestion fine des ressources, templates et bibliothèques de déploiement
- Kubernetes (k8s): nombreuses fonctionnalités (clusters, scaling, templates, ressources, stockage), templates et bibliothèques (via helm), contrôle d'accès

# k8s: ressources

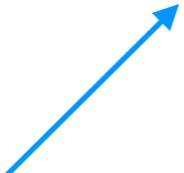
- Point de vue générique ce qui est géré par k8s:
  - Conteneurs et assimilés
  - Stockage (volumes et ressources connexes)
  - Réseau (services, load balancer)
- Système de *labels* et de *selectors*:
  - clé/valeur, utilisable comme label ou comme sélecteur  
app: cinema
  - mini langage de sélecteurs:  
environment in (production, qa)  
tier notin (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 apply -f mondeploiement.yml
```

Description de la ressource à créer / mettre à jour



# k8s: exemple de fichier de ressource

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

# Orchestration

## Déploiement de conteneurs

- Fonctionnalité de base
- Déclaratif
  - Sous forme d'un fichier de configuration, pas une commande avec des options
  - Limite la quantité de script pour l'automatisation
  - Gestion de la configuration (environnement, fichiers)

# Orchestration

## Déploiement de conteneurs: scaling

- Gérer plusieurs conteneurs ayant:
  - Même image
  - Même configuration
- Pouvoir in/décrémenter le nombre de copies  
(a.k.a. *horizontal scaling*)

# Orchestration

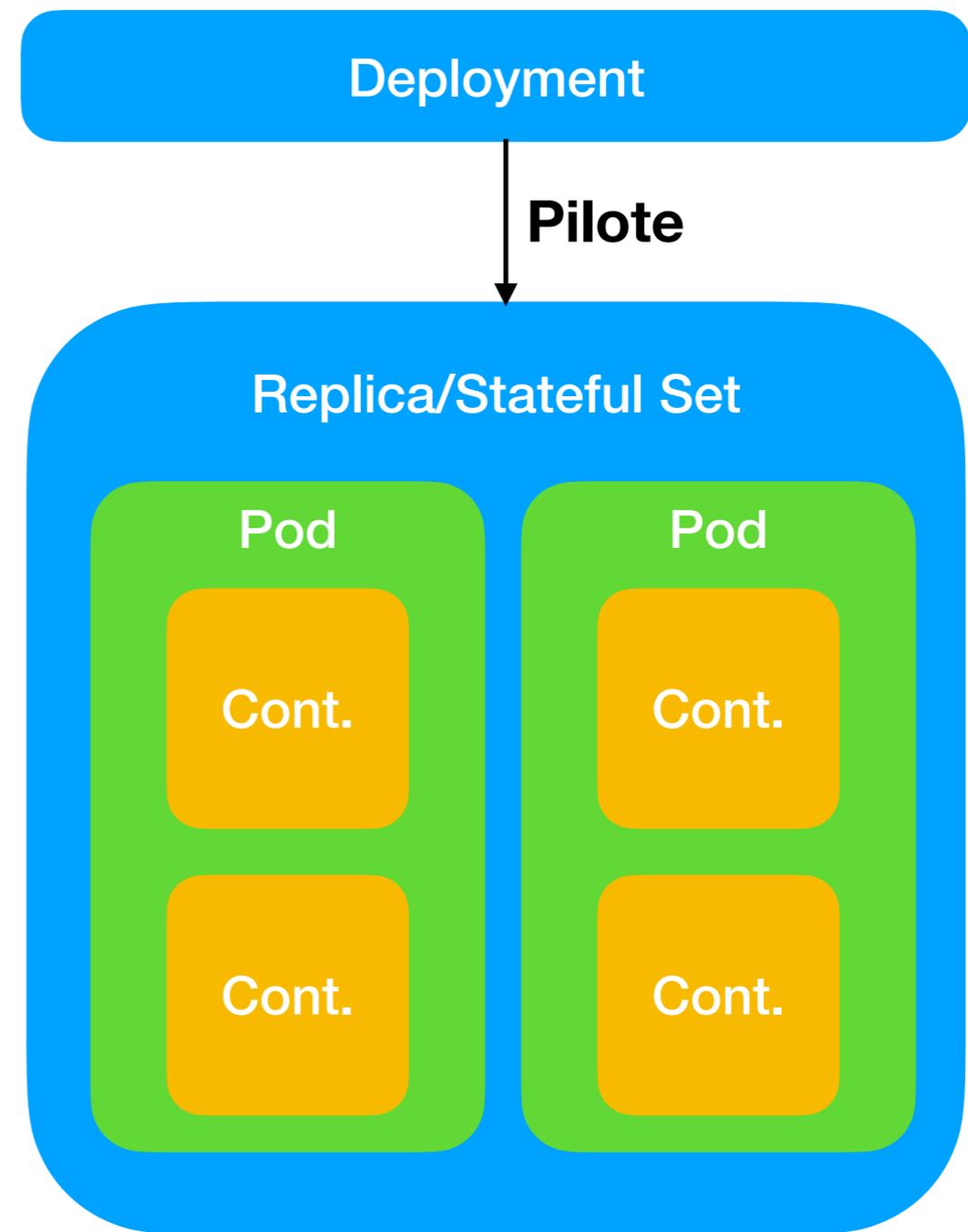
## Déploiement et supervision

- Supervision simple: état du conteneur
  - en construction, en marche, arrêté, etc
- Plus avancé
  - Le service est-il fonctionnel ?  
Un process peut tourner et renvoyer des 503
  - *Health check*: code exécuté pour vérifier le bon fonctionnement du service  
→ code *ad-hoc* pour chaque service
- Réaction
  - Que faire si un service est défaillant ?  
→ kill/reboot ?

# k8s: conteneurs, pods, sets

Encapsulation des conteneurs

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



# k8s: configuration

- à la Docker: command line, variables d'environnement
- ConfigMap:
  - dictionnaire clés-valeurs
  - visible comme un répertoire,  
clé ↔ fichier  
valeur ↔ contenu
  - ou visible via des variables d'environnement
  - gestion distincte des deployments
  - peut être partagée
- Secrets:
  - Similaire à ConfigMap, mais sécurisé

# k8s: supervision applicative

- Health checks (probes)
  - Readyness / Liveness
  - différents types de sondes
    - commande exécutée dans le conteneur
    - requête HTTP
    - vérification port TCP
  - succès: code retour = 0
- À utiliser avec restartPolicy

# Orchestration Batches

- Processus à durée de vie limitée
- À gros grain, certaines problématiques similaires aux services
  - Supervision
  - Relance (partielle) en cas d'échec
- Nécessite parfois un accès privilégié à certains services  
→ exécution au sein de l'environnement d'orchestration

# k8s: Jobs

- Possibilité d'avoir plusieurs pods : parallélisme
- Redémarrage en cas d'échec, avec limite sur le nombre d'essais
  - Attention à la restartPolicy: OnFailure ou Never
- Cron Jobs
- TTL Controllers: pour nettoyer les ressources
- Pas de système type Spring Batch pour redémarrer un job à mi-chemin

# Orchestration

# Stockage

- Partage de données entre différents conteneurs
- Persistance 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'usage
  - Plus générique
- Volume lié à un pod
  - même durée de vie
  - montable par les conteneurs du pod (possibilité de montage partagé)

# k8s: stockage persistant

- Données à durée de vie plus longue que celle du pod
  - PersistentVolume : données
    - Hébergée en général sur un système tiers (e.g. 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

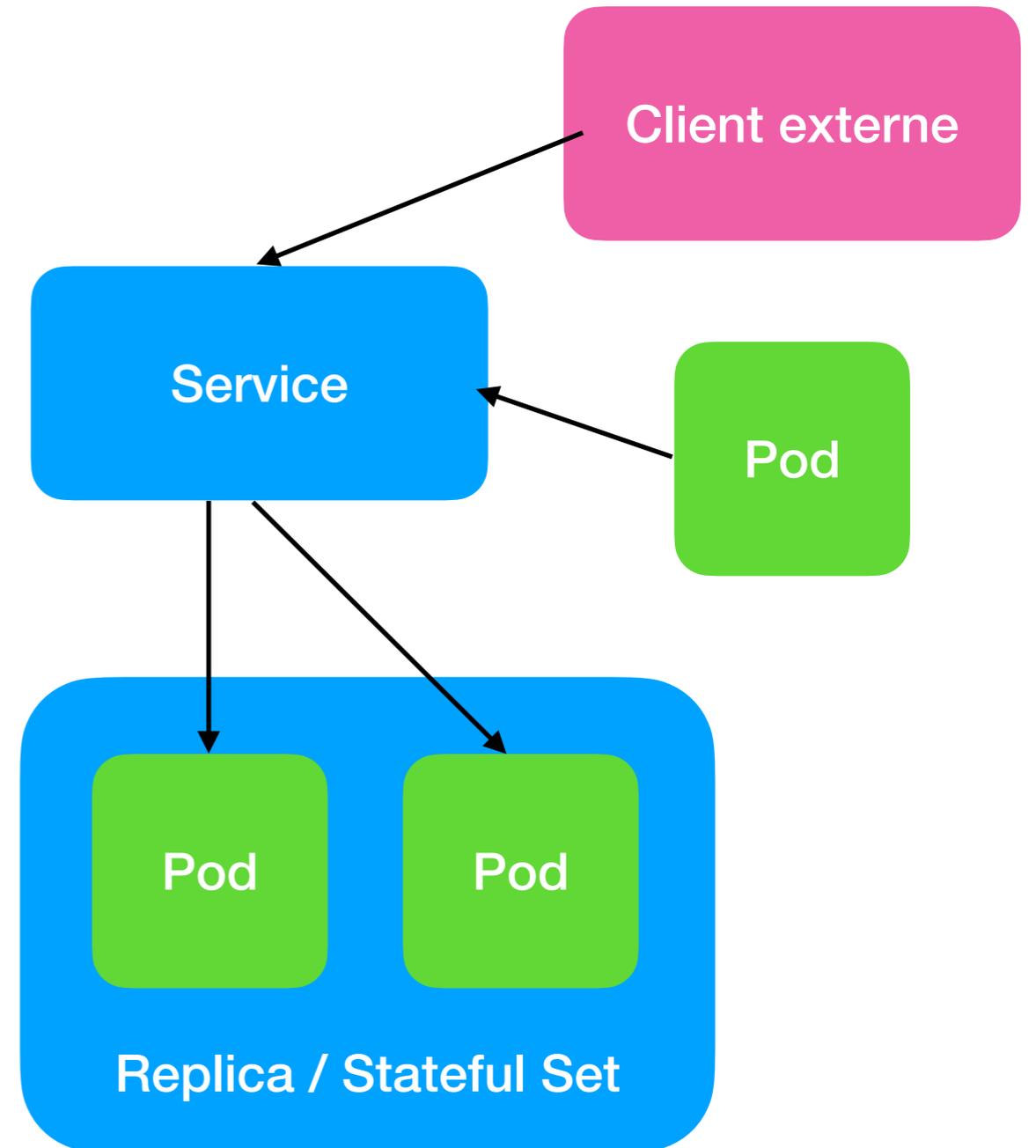
# Orchestration

## Réseau et API Management

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

# k8s: services

- Points d'accès virtuel pour un XXX 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

# Orchestration

## Sécurité et isolation

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

# k8s: utilisateurs

- Deux types: humains vs pods (User vs ServiceAccount)
- Authentification: mécanismes variés, possibilité d'injection par secret pour les ServiceAccounts
- Systèmes d'autorisations variés: RBAC, ABAC, etc

# k8s: isolation réseau

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

# k8s: limitations de ressources

- Possibilité de limiter les ressources associées à un pod
- Bonne pratique: toujours fixer des limites CPU/RAM
- système de ressources ouvert, e.g.

```
limits:
```

```
nvidia.com/gpu: 1 # requesting 1 GPU
```