Les 12 Factor Kubernetes

Speakers : Etienne Coutaud (Octo)
Format : Conférence
Date : 20 avril 2018
Code source des démos : https://github.com/etiennecoutaud/12factorsK8s

Les 12 factors ne viennent pas de la littérature mais du retour d'XP d'Etienne et d'Octo.

#1 1 pod = 1 ou n conteneurs

Raccourci vrai dans 99% des cas

#2 Label everywhere

On peut mettre des étiquettes sur tous les composants : pod, deployment, namespace …
Double intérêt :

1. Usage technique : Kubernetes utilise ces labels pour réaliser certaines tâches.
   Exemple 1 : pour compter le nombre de pod.
   Exemple 2 : permettre aux pods ayant le label A d'accéder aux pods avec le label B.
2. Usage organisationnel : permet de faire des recherches (ex : pod d'un environnement dédié)

#3 Infrastructure as Code & Versionning

Les fichiers YAML de configuration sont du code d'infrastructure. Il faut l'historiser sur le dépôt de code.
Outillage : test, linter, CI
Process : pull request, merge …

#4 Un Service pour exposer mes applications

Un Service est un objet au sens Kubernetes.
Un Service se trouve dans l'iptable.
Plusieurs typologies de services : External, ClusterIP, Headless, NodePort, LoadBalancer

#5 ConfigMap et Secret pour configurer

La configuration devient une ressource. Le conteneur est agnostique de l'environnement sur lequel il est déployé.
Kubernetes permet de déployer de la configuration accessible par le Pod.
Le développeur peut fournir de la conf applicative et les sysadmin de la conf d'infra.
Les fichiers de configuration peuvent être montés dans un volume (ex : /config/nginx.properties) ou des variables d'environnement.
La configuration a un cycle de vie différent de l'application.
Kubernetes est capable de la déployer à chaud. L'application, pas forcément.

#6 Limits et Request pour maitriser l'usage des ressources

Mécanisme à utiliser dès le départ : mécanisme des Requests et des Limits.
Permet d'aider le scheduler à se protéger des pods cannibales.
On indique la conf minimale du pod (ex : RAM : 250 Mo et 1 CPU).
La limite permet d'autoriser Kubernetes à tuer un pod lorsque la limite est dépassée (ex : RAM à 1,5Go).

#7 Penser au cycle de vie des Pods

Le Replica Set de Kubernetes permet à Kubernetes de connaître le nombre de pods à démarrer et de pouvoir redémarrer les pods quand ils crashent.
Pour réaliser une montée de version de nginx, il faut passer par le fichier deployment.yml.
Kubernetes va alors faire un rolling update : redémarrer un à un tous les pods en faisant la montée de version demandée.

#8 Resilience

Pour rendre le système résilient, on peut utiliser les sondes LivenessProbe et ReadinessProbe.

#9 Latest is not the version

Ne pas pousser les images latest car on peut embarquer des commits non testés.
Utiliser des pipelines de promotion de container.

#10 Les Pods sont stateless (sauf s'ils n'ont pas de raison de l'être)

Penser les applications stateless.
Le mécanisme d'auto-scaling permet de scaller très facilement. Pas de session. Les 3 pods ont des états différents.
Pour les applications statefull, passer par un cache distribué.

#11 Les Volumes sur stockage distribué

Pour faire du stateful, il faut une solution de stockage distribué.
Utilisation des objets PersistentVolumeClaim.

#12 Mes applications sont 12 factor apps

Respecter les 12 principes émis par Heroku (PaaS)

Take Away

- Acceptez le paradigme de Kubernetes
- Ne déportez pas la complexité de l'infrastructure dans les applications
- Pensez vos applications « cloud ready » pour en simplifier l'admin

- N’installez pas un Kubernetes ! Utilisez les services managés proposés par AWS, Google et Microsoft
- « Think Big, Start Small »
- Appuyez-vous sur la communauté Kubernetes
- Ces 12 principes ne sont qu’un avant-goût.