Kubernetes & Prometheus

Introduction au DevOps

ALFRED DEIVASSAGAYAME

Les métiers de l'OPS et de la Data ?

Les métiers de l'OPS (opérations) et de la Data (données) sont vastes et couvrent un large éventail de fonctions au sein d'une organisation.

Métiers de l'OPS

- **Ingénieur DevOps** : Responsable de l'intégration des processus de développement de logiciels (Dev) et des opérations informatiques (Ops), favorisant une culture et un environnement où la construction, les tests et la libération de logiciels peuvent se produire rapidement, fréquemment et plus fiabilité.
- **Administrateur Système** : Gère et maintient les systèmes informatiques et les serveurs, en assurant leur bon fonctionnement, leur sécurité et leur efficacité.
- **Ingénieur de Production** : Se concentre sur la maintien en condition opérationnelle des systèmes et services informatiques, garantissant leur disponibilité, performance et réactivité aux incidents.
- **Ingénieur Réseau** : Conçoit, implémente et gère les infrastructures réseau pour garantir une connectivité fiable et sécurisée.
- **Responsable des Opérations IT** : Supervise les opérations quotidiennes des départements informatiques, y compris la gestion des équipes, la planification des projets et la garantie de la sécurité des systèmes d'information.

Métiers de l'OPS

- **Data Analyst** : Analyse les ensembles de données pour aider les entreprises à prendre des décisions éclairées. Utilise des outils statistiques et des techniques de visualisation de données pour interpréter les tendances et les modèles.
- **Data Scientist** : Combine l'expertise en statistique, en informatique et en affaires pour analyser des ensembles de données complexes et générer des insights. Utilise des techniques de machine learning et de modélisation prédictive.
- **Ingénieur Data** : Conçoit, construit et gère les architectures de données d'une entreprise, y compris les bases de données et les pipelines de données, pour faciliter l'analyse et l'exploitation des données.
- **Architecte de Données** : Élabore des plans stratégiques pour la gestion des données d'une entreprise, y compris la conception de schémas de bases de données, l'intégration de systèmes et la gouvernance des données.
- **Data Steward** : Assure la qualité et la gestion des données au sein de l'organisation, en établissant des politiques et des procédures pour leur utilisation, leur protection et leur conservation.

Le métier de SRE

Le métier de SRE, ou Site Reliability Engineering (Ingénierie de Fiabilité de Site), est une discipline qui combine des éléments du génie logiciel et les applique aux problèmes d'infrastructure et d'opérations, avec l'objectif principal de créer des systèmes scalables et hautement fiables.

Le concept a été popularisé par Google et est désormais adopté par de nombreuses organisations technologiques à travers le monde.

Le métier de SRE

Rôles et responsabilités

Les ingénieurs SRE ont pour mission de garantir que les services en ligne d'une entreprise sont fiables, disponibles et performants, tout en maintenant un rythme rapide de développement et de déploiement de nouvelles fonctionnalités. Voici quelques-unes de leurs principales responsabilités :

- **Développement d'outils et d'automatisations** : Les SRE développent des outils et des systèmes pour automatiser le déploiement, la surveillance et la gestion de l'infrastructure et des applications, réduisant ainsi la charge de travail opérationnelle et les risques d'erreur humaine.
- **Gestion des incidents** : Ils jouent un rôle clé dans la réponse aux incidents, la résolution des problèmes en temps réel et la mise en œuvre des mesures correctives pour éviter la récurrence des incidents.
- **Conception pour la fiabilité** : Les SRE collaborent avec les équipes de développement pour concevoir et implémenter des architectures résilientes et tolérantes aux pannes, en utilisant des pratiques telles que le chaos engineering et les tests de charge pour valider la robustesse des systèmes.
- **Surveillance et performance** : Ils mettent en place des systèmes de surveillance et d'alerte pour détecter et réagir rapidement aux anomalies de performance ou aux pannes, en s'assurant que les systèmes répondent aux objectifs de niveau de service (Service Level Objectives, SLOs).
- **Optimisation des coûts** : Les SRE travaillent également à optimiser l'utilisation des ressources pour équilibrer les coûts et la performance, notamment dans les environnements cloud où les ressources peuvent être ajustées dynamiquement.

Le métier de SRE

Compétences requises

Pour réussir en tant que SRE, un individu doit posséder un mélange de compétences en développement logiciel, en systèmes, en réseaux, et une solide compréhension des principes de l'ingénierie des systèmes. Les compétences clés incluent :

- **Programmation** : La maîtrise de langages de programmation tels que Python, Go ou Java est essentielle pour automatiser les tâches et développer des outils internes.
- **Connaissances systèmes** : Une compréhension approfondie des systèmes d'exploitation, des réseaux, des bases de données et du stockage est nécessaire pour diagnostiquer et résoudre les problèmes d'infrastructure.
- **DevOps et CI/CD** : Une expérience avec les pratiques DevOps et les pipelines d'intégration continue et de déploiement continu (CI/CD) est importante pour le déploiement rapide et fiable des applications.
- **Cloud et technologies de conteneurisation** : La connaissance des services cloud (AWS, Google Cloud, Azure) et des technologies de conteneurisation comme Docker et Kubernetes est souvent requise.
- **Compétences en communication** : Les SRE doivent être capables de communiquer efficacement avec les équipes de développement, les opérations et le support pour coordonner les efforts et résoudre les problèmes.

Les métiers de l'OPS et de la Data ?

The 15 most in-demand tech jobs for 2024

Job	25th percentile	50th percentile	75th percentile
Systems security manager	$137,250	$167,750	$198,500
Network/cloud architect	$133,000	$165,750	$196,750
Applications architect	$134,750	$163,250	$195,500
IT director	$132,000	$161,250	$192,000
ERP integration manager	$120,500	$153,250	$187,500
Big data engineer	$122,000	$149,500	$174,250
Data security analyst	$120,000	$147,500	$170,250
Data scientist	$119,000	$144,250	$167,000
DevOps engineer	$113,500	$140,250	$170,000
Network security engineer	$115,000	$140,000	$163,750
Senior web developer	$115,500	$137,750	$160,000
Database developer	$107,500	$130,500	$147,750
Software engineer	$108,500	$129,250	$152,500
Network/cloud engineer	$106,500	$127,750	$160,750
Help desk support manager	$83,250	$102,500	$114,750

- 25th percentile: new to the type of role, still acquiring relevant skills
- 50th percentile: average experience, has most of the necessary skills
- 75th percentile: above average experience, has all needed skills

Introduction à Kubernetes

Définition

Kubernetes est effectivement une plateforme open-source puissante qui permet d'automatiser le déploiement, la mise à l'échelle et la gestion d'applications conteneurisées.

Introduction à Kubernetes

Automatisation du déploiement

- Kubernetes permet de déployer rapidement des applications conteneurisées sur un cluster de machines sans avoir besoin d'intervenir manuellement pour démarrer les conteneurs sur les différents hôtes.

- Cela simplifie considérablement le processus de déploiement, surtout à grande échelle.

Introduction à Kubernetes

Mise à l'échelle dynamique

- Avec Kubernetes, vous pouvez ajuster automatiquement le nombre de réplicas de votre application en fonction de la demande.
- Ce qui garantit que votre application dispose des ressources nécessaires pour répondre à la charge de travail sans gaspiller de ressources lorsque la demande est faible.

Les réplicas, dans le contexte de Kubernetes, se réfèrent au nombre de copies d'un pod qui doivent être exécutées dans un cluster à un moment donné.

Introduction à Kubernetes

Gestion de l'état et des données persistantes

- Même si les conteneurs sont par nature éphémères, Kubernetes fournit des mécanismes pour gérer des données persistantes à l'aide de volumes, permettant ainsi aux applications stateful (qui nécessitent un stockage durable des données) de fonctionner efficacement.

Introduction à Kubernetes

Équilibrage de charge et découverte de services

- Kubernetes facilite la distribution du trafic réseau entre les différentes instances de votre application à l'aide de services, qui définissent des règles d'équilibrage de charge et permettent la découverte de services au sein du cluster.

Introduction à Kubernetes

Auto-réparation

- Kubernetes surveille en permanence l'état des applications et des nœuds du cluster.
- Si un conteneur échoue, il est redémarré ou remplacé automatiquement.
- Si un nœud devient défaillant, les conteneurs qui y sont exécutés sont redéployés sur d'autres nœuds disponibles.

Introduction à Kubernetes

Rollouts et Rollbacks

- Vous pouvez mettre à jour les applications déployées sur Kubernetes en toute sécurité, avec la possibilité de revenir à une version antérieure si nécessaire.
- Kubernetes gère ces mises à jour graduellement (rolling updates), minimisant ainsi les interruptions de service.

Introduction à Kubernetes

Portabilité et indépendance du fournisseur de cloud

- Kubernetes peut s'exécuter sur n'importe quel environnement : local, cloud public (comme AWS, Google Cloud, Azure), cloud privé ou hybride.
- Cela permet une grande flexibilité et évite de se retrouver enfermé chez un fournisseur spécifique.

Introduction à Kubernetes

Portabilité et indépendance du fournisseur de cloud

- Kubernetes peut s'exécuter sur n'importe quel environnement : local, cloud public (comme AWS, Google Cloud, Azure), cloud privé ou hybride.
- Cela permet une grande flexibilité et évite de se retrouver enfermé chez un fournisseur spécifique.

Architecture de Kubernetes

L'architecture de Kubernetes est conçue pour être très modulaire, avec des composants distribués qui fonctionnent ensemble pour gérer des applications conteneurisées à grande échelle.

Architecture de Kubernetes

Master Node (Nœud Maître)

Le Master Node agit comme le cerveau du cluster Kubernetes. Il est responsable de la prise de décision globale dans le cluster, et il orchestre les nœuds worker (travailleurs). Les composants principaux du Master Node incluent :

- **API Server (kube-apiserver)** : Sert comme le point d'entrée de l'API Kubernetes, permettant la communication entre les différents composants du cluster.
- **Scheduler (kube-scheduler)** : Sélectionne quel nœud worker va exécuter les nouveaux pods basés sur la disponibilité des ressources.
- **Controller Manager (kube-controller-manager)** : Exécute les processus de contrôleur, y compris les contrôleurs de nœud, de réplication, et d'endpoints.
- **etcd** : Une base de données clé-valeur légère et distribuée qui stocke toutes les données de configuration du cluster, assurant la persistance de l'état du cluster.

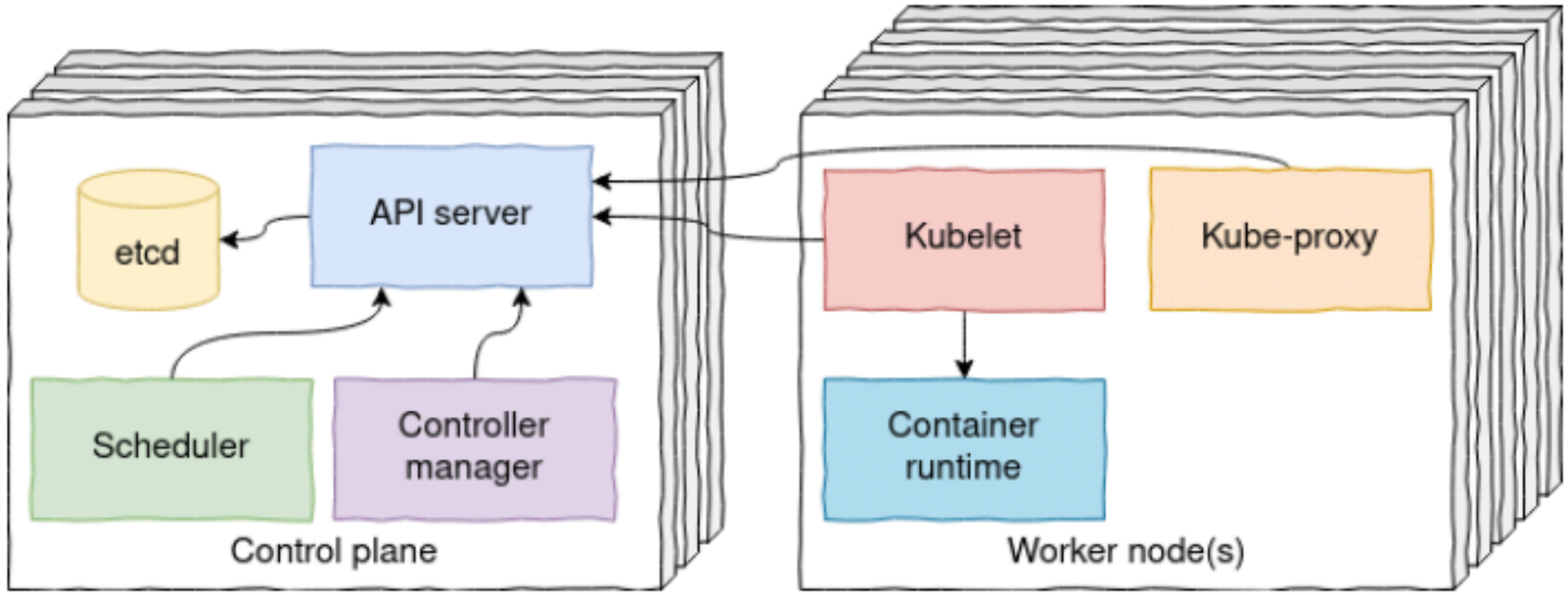
Architecture de Kubernetes

Worker Nodes (Nœuds Travailleurs)

Les nœuds workers exécutent les applications conteneurisées et sont les machines où les conteneurs (dans les pods) sont réellement lancés par Kubernetes. Les composants principaux des nœuds workers incluent :

- **Kubelet** : Un agent qui s'exécute sur chaque nœud du cluster, s'assurant que les conteneurs sont en cours d'exécution dans un Pod.
- **Kube-proxy** : Maintient les règles de réseau sur les nœuds, permettant la communication réseau aux pods à travers le cluster.
- **Container Runtime** : Le logiciel responsable de l'exécution des conteneurs, tel que Docker, containerd, ou CRI-O.

Architecture de Kubernetes




```
$ kubectl get componentstatus
NAME                 STATUS    MESSAGE              ERROR
scheduler            Healthy   ok
etcd-0               Healthy   {"health": "true"}
controller-manager   Healthy   ok
```

Concepts de base

Pods

- **Définition**:
  - Le Pod est l'unité de base de déploiement dans Kubernetes.
  - Il encapsule un ou plusieurs conteneurs, leurs ressources de stockage et de réseau, et une spécification sur la manière d'exécuter les conteneurs.
  - Les conteneurs dans un Pod partagent le même espace de noms réseau, ce qui signifie qu'ils ont la même adresse IP et le même port.
- **Utilisation**:
  - Les Pods sont utilisés pour exécuter une instance d'une application ou d'un service dans votre cluster.
  - Si votre application nécessite plusieurs conteneurs qui doivent être exécutés ensemble, ils seront regroupés dans un même Pod.

Concepts de base

ReplicaSets

- **Définition**:
  - Un ReplicaSet est un objet qui définit le nombre souhaité de réplicas d'un Pod spécifique à maintenir dans le cluster à tout moment.
  - Il veille à ce que le nombre de Pods en exécution corresponde au nombre spécifié dans la définition.
- **Utilisation**:
  - Les ReplicaSets garantissent la disponibilité et la scalabilité des applications en maintenant un ensemble de réplicas de Pods.
  - Ils remplacent les Pods qui échouent, sont supprimés ou terminés.

Concepts de base

Deployments

- **Définition**:
  - Un Deployment est une abstraction supérieure qui gère les ReplicaSets et permet la mise à jour déclarative des Pods et des ReplicaSets.
  - Les Deployments prennent en charge la mise à l'échelle, le rollback et la mise à jour des applications.
- **Utilisation**:
  - Les Deployments sont utilisés pour gérer le cycle de vie des applications déployées sur Kubernetes, en permettant des mises à jour sans interruption du service grâce à des stratégies comme le déploiement progressif (rolling update).

Concepts de base

Services

- **Définition**:
  - Un Service est une abstraction qui définit un ensemble logique de Pods et une politique d'accès à ces Pods, généralement via un réseau.
  - Les Services permettent d'exposer des applications en cours d'exécution sur un ensemble de Pods comme un service réseau.
- **Utilisation**:
  - Les Services sont utilisés pour permettre la communication et l'accès aux applications déployées sur les Pods, de l'intérieur ou de l'extérieur du cluster Kubernetes, en fournissant une adresse IP fixe et un nom DNS pour un ensemble de Pods.

Concepts de base

Volumes

- **Définition**:
  - Un Volume dans Kubernetes est un objet qui permet de stocker des données en dehors du cycle de vie des Pods individuels, garantissant ainsi la persistance des données.
- **Utilisation**:
  - Les Volumes sont utilisés pour fournir un stockage persistant aux applications qui nécessitent un stockage au-delà de la vie d'un Pod, pour partager des données entre les conteneurs d'un Pod ou pour stocker des données sensibles comme des secrets.

Concepts de base

Namespaces

- **Définition**:
  - Les Namespaces sont des partitions virtuelles au sein d'un cluster Kubernetes, permettant de diviser les ressources du cluster entre plusieurs utilisateurs ou groupes d'utilisateurs.
- **Utilisation**:
  - Les Namespaces sont utilisés pour organiser les objets dans un cluster, offrant une manière de segmenter le cluster en sous-ensembles pour gérer les droits d'accès, les quotas et les limites de ressources, facilitant ainsi la gestion multi-tenant du cluster.

Installation et Configuration

Pour mettre en place un environnement Kubernetes, vous pouvez utiliser différents outils en fonction de vos besoins, que ce soit pour un apprentissage, un développement, un test ou une production.

Installation et Configuration

Compatibité

- Système d'exploitation et virtualisation :
    - Linux : docker, KVM2 ou VirtualBox
    - macOS : docker, HyperKit, VirtualBox, Parallels ou VMWare
    - Windows : Hyper-V, docker ou VirtualBox

Options de virtualisation VT-x/AMD-v activées

Connexion Internet lors du premier lancement

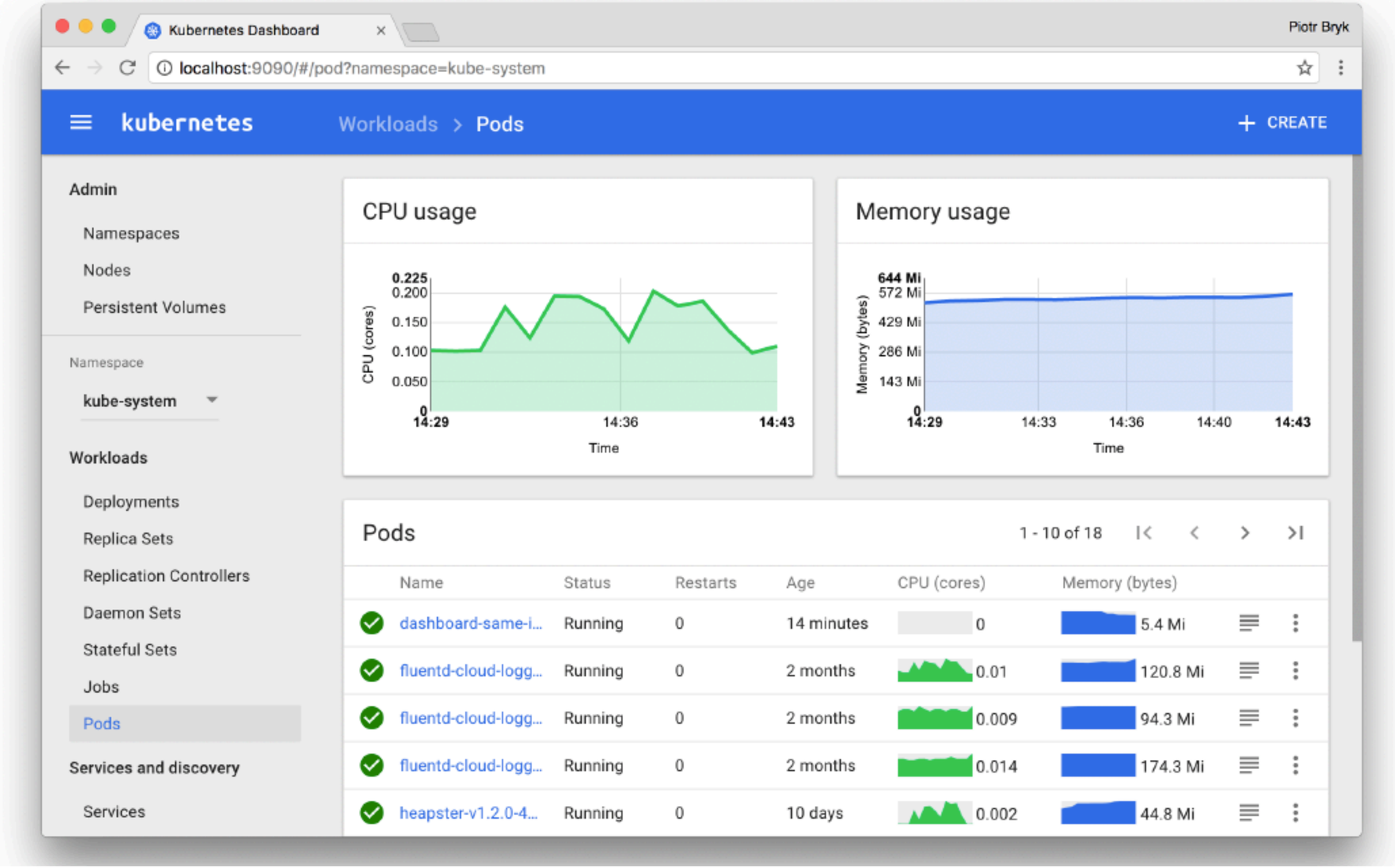
Installation et Configuration

Minikube

Minikube est un outil qui permet de déployer un cluster Kubernetes simple sur une machine locale, idéal pour l'apprentissage et le développement.

- **Installation**:
  - Téléchargez et installez Minikube à partir du site officiel de Kubernetes ou en utilisant un gestionnaire de paquets.
- **Démarrage du Cluster:**
  - Lancez le cluster avec la commande ***minikube start.***
  - Une fois démarré, Minikube configure automatiquement ***kubectl*** pour utiliser le contexte du cluster Minikube.
- **Utilisation:**
  - Utilisez ***kubectl*** pour interagir avec votre cluster.
  - Minikube offre également un Dashboard pour une gestion visuelle, activable via ***minikube dashboard***.

Installation et Configuration



Dashboard Kubernetes

- Le Dashboard est une interface web permettant d'interagir avec une instance Kubernetes
- Le Dashboard permet de :
    - déployer des applications
    - rechercher des informations suite au comportement anormal d'une application déployée
    - visualiser l'ensemble des applications déployées
    - modifier la configuration des applications déployées et les mettre à jour
- Le Dashboard permet aussi de connaître l'état des ressources d'une instance et d'accéder aux logs des composants du cluster ainsi que ceux des applications
- Pour que le Dashboard puisse afficher les métriques et les graphiques associés, l’addon metrics-server doit être installé dans le cluster

Dashboard Kubernetes

- Dans tous les cas, il est possible de lancer la commande ***kubectl proxy*** qui permet d'accéder au Dashboard depuis son poste local
- On peut alors accéder au dashboard via l'url suivante:

http://localhost:8001/api/v1/namespaces/kubernetesdashboard/services/https:kubernetes-dashboard:/proxy/

Remarque : sous minikube, il est exposé en http. L'accès au dashboard se fait donc sur http :

http://localhost:8001/api/v1/namespaces/kubernetesdashboard/services/http:kubernetes-dashboard:/proxy/

Kubectl

Lister les commandes

- Toutes les informations disponibles et les actions réalisables par le biais du Dashboard le sont aussi en passant par la ligne de commande ***kubectl***

- ***kubectl*** permet d'interagir en ligne de commande avec vos instances Kubernetes

```
$ kubectl
kubectl controls the Kubernetes cluster manager.

 Find more information at: https://kubernetes.io/docs/reference/kubectl/overview/

Basic Commands (Beginner):
  create        Create a resource from a file or from stdin.
  expose        Take a replication controller, service, deployment or pod and expose it as a new
Kubernetes Service
  run           Run a particular image on the cluster
  set           Set specific features on objects

Basic Commands (Intermediate):
  explain       Documentation of resources
  get           Display one or many resources
  edit          Edit a resource on the server
  delete        Delete resources by filenames, stdin, resources and names, or by resources and
label selector
(...)
```

Kubectl

Aide en ligne de commande

- Lancer ***kubectl*** pour voir la liste de toutes les commandes disponibles
- Lancer ***kubectl <command> --help*** pour voir l'aide pour une commande en particulier
- Lancer ***kubectl options*** pour voir la liste des options globales (qui s'appliquent à toutes les commandes)

Démarrer son premier conteneur dans K8s

- Le meilleur moyen de déployer une application dans ***Kubernetes*** de manière reproductible est depasser par un fichier descripteur au format yaml ou ***json***
- Mais il est aussi possible d'utiliser la commande ***run*** qui permet de se passer d'un tel descripteur

Kubectl

Utilisation de ‘Kubectl run’

La commande :

```
$ kubectl run whoami --image=containous/whoami:latest --port=80
```

- démarre un ***Pod*** (que l'on verra en détail dans le chapitre)
- à partir de l'image Docker ***containous/whoami:latest***
- et référence le port ***80*** du conteneur créé
    - "***référence***" dans le sens ou c'est une déclaration d'intention permettant d'utiliser par la suite ce port
    - Ne pas spécifier de port n'empêche pas le port d'être accessible

Kubectl

Exemples d’utilisation de ‘Kubectl run’

- **kubectl run nginx —image=nginx :**
  - Démarre une instance unique à partir de l'image nginx
- **kubectl run hazelcast --image=hazelcast —port=5701 :**
  - Démarre une instance hazelcast et expose le port 5701 du conteneur
- **kubectl run hazelcast --image=hazelcast —env=« DNS_DOMAIN=cluster" :**
  - Démarre une instance hazelcast en spécifiant des variables d'environnement
- **kubectl run nginx --image=nginx --dry-run=client --output yaml**
  - Affiche les descripteurs sans réellement créer les objets

Kubectl

‘Kubectl get containers’

- Il n'existe pas de commande ***kubectl*** permettant de lister les conteneurs lancés
- L'unité la plus petite est le ***Pod*** (que l'on verra en détail dans le chapitre)
- On peut donc lancer kubectl get pods (ou ***kubectl get po*** en utilisant le raccourci de la ressource ***Pods***)

Kubectl

Lister les pods

```
$ kubectl get pods
NAME          READY     STATUS    RESTARTS   AGE
whoami        1/1       Running   0          2m
```

À noter, par défaut, les informations remontées sont :

- Le nom du Pod
- Le nombre de conteneurs ***prêts*** (un Pod peut contenir plusieurs conteneurs)
- Le status du Pod (Running, Terminating, CrashLoopBackOff, …)
- Le nombre de ***restarts*** éventuels de l'un des conteneurs du Pod
- L'âge du Pod : depuis combien de temps la demande de création du Pod a-t-elle été prise en compte

Kubectl

Lister plus d’informations sur les pots

Il est possible d'utiliser l'option --output (ou -o) pour affiner le format de sortie

- json ou yaml
- wide ou name
- custom-columns=... ou custom-columns-file=... ( )
- go-template=... ou go-template-file=... ( )
- jsonpath=... ou jsonpath-file=... ( )

```
$ kubectl get pods --output wide
NAME         READY     STATUS    RESTARTS   AGE       IP           NODE
whoami       1/1       Running   0          1h        172.17.0.7   minikube
```

```
$ kubectl get pods -o name
pods/whoami
```

```
$ kubectl get pods \
      -o=custom-columns=NAME:.metadata.name,RSRC:.metadata.resourceVersion
NAME         RSRC
whoami       103751
```

Kubectl

Accéder à tous les détails d’un pod

La commande kubectl describe pod <pod-name> permet d'accéder à des informations détaillées sur un Pod

```
$ kubectl describe pod whoami
Name:           whoami
Namespace:      default
Node:           minikube/192.168.99.100
Start Time:     Sun, 08 Oct 2017 19:02:51 +0200
Labels:         run=whoami
Annotations:    kubernetes.io/created-by=
{"kind":"SerializedReference","apiVersion":"v1","reference":
{"kind":"ReplicaSet","namespace":"default","name":"whoami","uid":"850adebf-ac4a-11e7-b4c7-
0800279f22af","a...
Status:         Running
IP:             172.17.0.7
.
.
Node-Selectors:  <none>
Tolerations:     <none>
Events:          <none>
```

Kubectl

Exposer l’application démarrée (contexte)

- Jusqu'à présent, l'utilisation de ***kubectl run*** a créé un Pod
- L'application ***whoami*** n'est pour l'instant accessible que depuis l'intérieur du cluster Kubernetes, par le biais de l'adresse IP attribuée au Pod

```
$ kubectl describe pod whoami | grep IP
IP:               172.17.0.7
```

- Comment faire si on veut accéder à l'application ***whoami*** de manière fiable ?
    - l'ip associée au Pod n'est pas stable dans le temps, si le Pod était redémarré, l'adresse pourrait changer
    - dans le cas d'une mise à l'échelle, nous aurions plusieurs Pods avec plusieurs IP, comment faire pour s'y connecter ?

=> La solution : passer par un ***Service*** (les détails seront vus dans le chapitre) en utilisant la commande kubectl expose

Kubectl

Accéder à un pod sans passer par un service

- Il est aussi possible d'accéder à un Pod sans passer par un Service de manière temporaire
- La commande ***kubectl port-forward*** permet de forwarder temporairement un port local (du host d'où est lancée la commande ***kubectl***) vers le port d'un Pod

```
$ kubectl port-forward whoami 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Handling connection for 8080
Handling connection for 8080
```

Pod

Modèle/concept du pod

- Un ***Pod*** (une cosse, une gousse, une coque) est un groupe d'un ou plusieurs conteneurs (Docker par ex)
- Le Pod est la brique de base d'un déploiement k8s
- C'est la brique de déploiement la plus petite que l'on puisse créer / déployer

Le concept du Pod permet à des conteneurs de partager des ressources afin de pouvoir collaborer plus facilement.

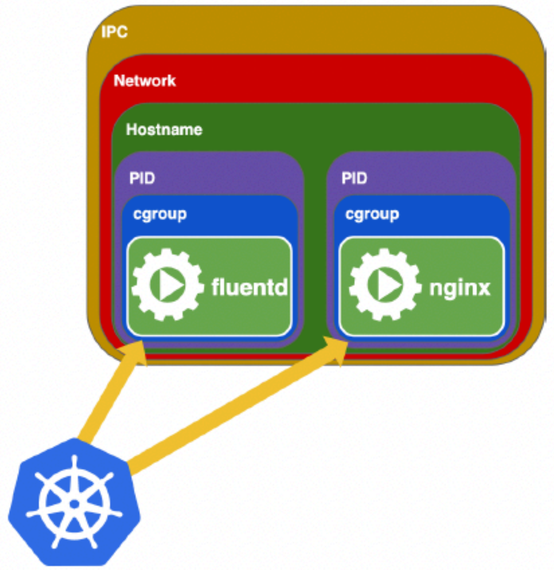
- Les conteneurs au sein d'un Pod partagent le même IPC et peuvent donc communiquer en utilisant les sémaphores ***SystemV*** ou la mémoire partagée POSIX
- Les conteneurs au sein d'un Pod partagent la même adresse IP, le même espace de ports et peuvent se 'trouver' par le biais de ***localhost***
- Les conteneurs au sein d'un Pod peuvent accéder aux mêmes volumes
- Les conteneurs d'un Pod sont ***toujours*** co-localisés et co-gérés

Pod

Conteneurs du pod vus par Kubernetes

- Grace au modèle des Pods, Kubernetes supervise chacun des process et connaît donc les états de chacun d'entre eux

- De cette manière, il est possible de redémarrer individuellement les process/ conteneurs si besoin (ou d'en voir les logs !)



Pod

Comment choisir entre plusieurs pods ou plusieurs conteneurs dans un pod ?

Se poser les questions :

- Les conteneurs doivent-ils tourner au même endroit ou peuvent-ils être répartis sur plusieurs noeuds ?
- Les conteneurs représentent-ils un seul tout ou sont-ils des composants indépendants ?
- Les conteneurs peuvent-ils/doivent-ils être mis à l'échelle ensemble individuellement ?

La plupart du temps, vous devriez séparer vos conteneurs dans des Pods distincts à moins qu'il n'y ait une raison particulière de ne pas le faire !

Pod

Cas d’usage de plusieurs conteneurs dans un pod

- Recharger de la configuration à chaud sur déclenchement d’évènement
- Conteneur web (Nginx, Apache, ...) + conteneur php-fpm
- Service routing
- Agrégation de logs
- Télémétrie

Le gros avantage est de ne pas avoir à modifier l'application principale

Dans ce cadre d'utilisation, les conteneurs associés au conteneur principal sont parfois appelés sidecar containers ou sidekick containers

Pod

Exemple de descripteur au format YAML

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: whoami
  labels:
    app: whoami
spec:
  containers:
    - name: whoami
      image: containous/whoami:latest
      imagePullPolicy: Always
      ports:
        - containerPort: 80
        - containerPort: 443
  restartPolicy: Always
```

Pod

Rappel : Obtenir un template de descripteur

```
kubectl run whoami \
  --image=containous/whoami:latest \
  --port=80 \
  --dry-run=client \
  --output yaml
```

Pod

Créer un pod à partir d'un fichier de description

Pour créer un Pod de manière ***déclarative*** (vrai pour les autres types de ressources), utiliser la commande ***kubectl apply*** avec l'option ***--filename*** (ou -f)

```
$ kubectl apply --filename pod-from-file.yml
pod "k8s-rulez" created

$ kubectl get pod k8s-rulez
NAME         READY     STATUS    RESTARTS   AGE
k8s-rulez    1/1       Running   0          11s
```

Anatomie d’un fichier descripteur

- Le descripteur yaml peut se décomposer en plusieurs sous-parties :
  - apiVersion
  - kind
  - metadata
  - spec, data, rules ou stringData
- Ces sous-parties se retrouvent dans tous les types de ressource

Utiliser les paths Json pour aller plus loin

```
$ kubectl explain pods.spec.containers

RESOURCE: containers <[]Object>

DESCRIPTION:
     List of containers belonging to the Pod. Containers cannot currently be
     added or removed. There must be at least one container in a Pod. Cannot be
     updated.

    A single application container that you want to run within a Pod.

FIELDS:
   workingDir    <string>
     Container's working directory. If not specified, the container runtime's
     default will be used, which might be configured in the container image.
     Cannot be updated.

   command    <[]string>
     Entrypoint array. Not executed within a shell. The docker image's
     ENTRYPOINT is used if this is not provided. Variable references $(VAR_NAME)
     (...)
```

Utiliser les paths Json pour aller plus loin

```
$ kubectl explain pods.spec.containers
...
FIELDS:
...
   imagePullPolicy    <string>
     Image pull policy. One of Always, Never, IfNotPresent. Defaults to Always
     if :latest tag is specified, or IfNotPresent otherwise. Cannot be updated.
     More info:
     https://kubernetes.io/docs/concepts/containers/images#updating-images

   name    <string> -required-
     Name of the container specified as a DNS_LABEL. Each container in a Pod
     must have a unique name (DNS_LABEL). Cannot be updated.
...
```

Plusieurs ressources dans un seul fichier

Il est tout à fait possible de décrire plusieurs ressources (de type différent ou pas) dans un unique fichier.

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: first-of-two
spec:
  containers:
    - image: nginx:alpine
...
---
apiVersion: v1
kind: Pod
metadata:
  name: second-of-two
spec:
  containers:
    - image: redis:alpine
...
```

```
$ kubectl apply -f two-pods-in-one-file.yml
pod "first-of-two" created
pod "second-of-two" created
```

Suppression d’une ressource

La suppression d'une resource s'effectue avec la commande ***kubectl delete*.**

- ***kubectl delete <type> <name>*** : suppression de la resource <name> de type <type>
- ***kubectl delete --filename <descripteur>*** : suppression de toutes les resources

déclarées dans le descripteur

```
$ kubectl delete pod k8s-rulez
pod "k8s-rulez" deleted

$ kubectl delete -f two-pods-in-one-file.yml
pod "first-of-two" deleted
pod "second-of-two" deleted
```