



# INTRO À KUBERNETES

# RAPPELS

# IMAGES

- Construites avec docker
- Utilisation d'un `Dockerfile` ou d'une lib dans le code
- Snapshot d'un *file system*

# CONTENEUR

- Matérialisation d'une image
- Lancement avec `docker run [...]`
- Exécution dans un environnement clos

# EXPOSITION DE PORTS

- Relie un port du conteneur avec celui de la machine hôte

# À QUOI ÇA SERT ?

Orchestrateurs

# APPLICATION SIMPLE

- Un back
- Un front
- Une base de donnée

# COMMENT GÉRER LES PROBLÈMES NON-PRÉVUS ?

- Problème du back qui crashe
- La base de donnée qui ne répond pas dans un délai imparti



# **COMMENT GÉRER LES MONTÉES DE VERSIONS APPLICATIVES ?**

- Montées de version sans down time

# COMMENT GÉRER LES PICS D'ACTIVITÉ ?

- Augmentation du nombre d'instances du back

# COMMENT GÉRER LE ROUTAGE/LOAD BALANCING ?

- Utilisation d'une brique dédiée (F5, HaProxy, apache en revers proxy, ...)
- Gestion des certificats https

# APPLICATION COMPLEXE

- Plusieurs fronts
- Trentaine de microservices
- Plusieurs bases de données
- Un broker de message

# **SOLUTION → ORCHESTRATION**

- Gérer le cycle de vie des conteneurs.
- Gérer les pics d'activité (augmentation simple du nombre de conteneurs).
- Gérer le routage au sein du cluster.
- Gérer le renouvellement des certificats https.

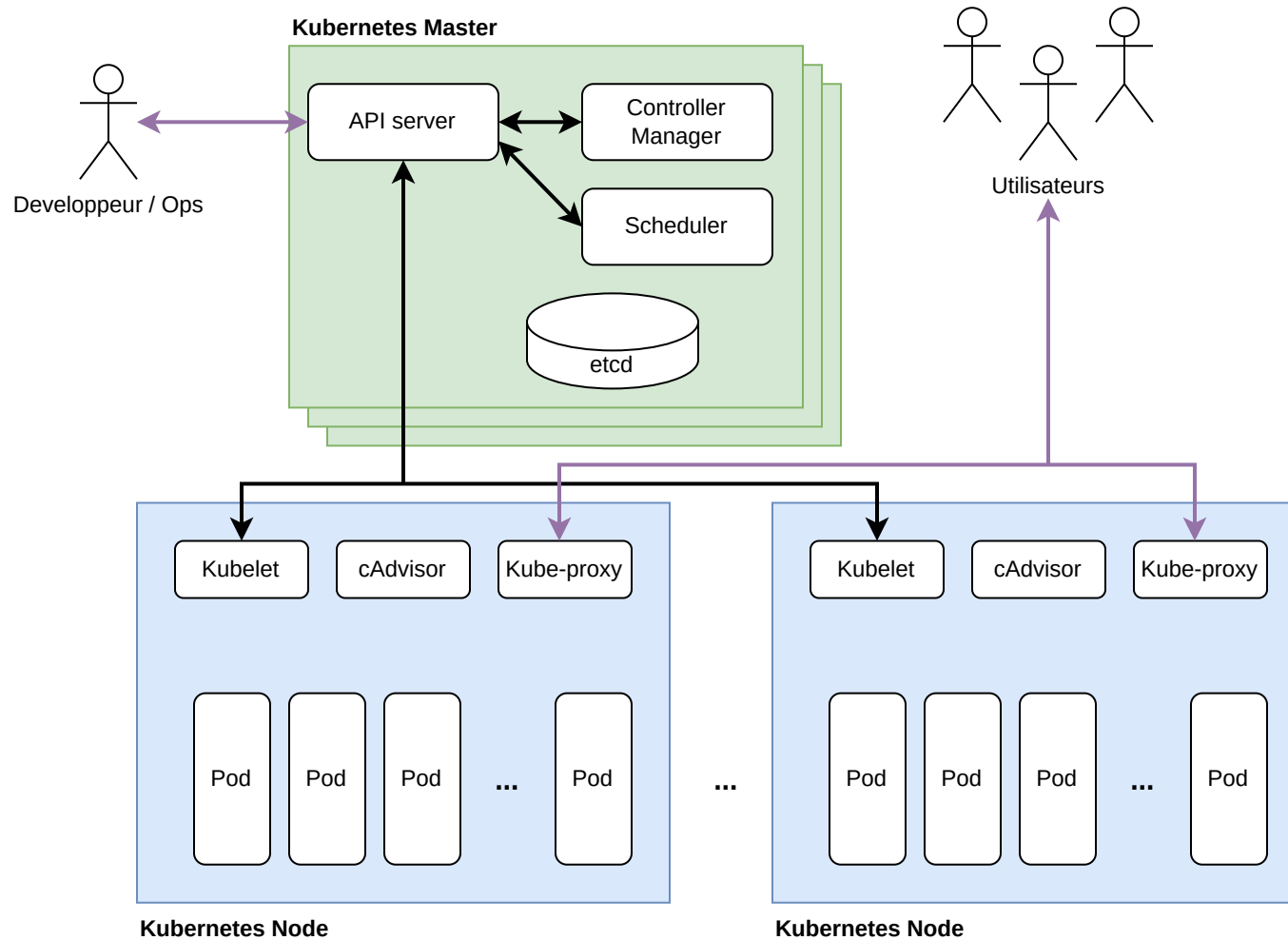
# **SOLUTION → ORCHESTRATION**

- Facilite les déploiements.
- S'intègre très bien dans une CI.
- Permet de gérer facilement plusieurs environnements (dev, recette, prod).
- Permet de mieux gérer les restrictions sur les applications (mem, cpu, ...).

# HISTORIQUE

- À l'origine : Google Borg — Projet interne ( → 2015)
- v1 Kubernetes (2014)
- Google a donné Kube à la CNCF (Cloud Native Computing Foundation)

# FONCTIONNEMENT





# PRINCIPES

# PODS

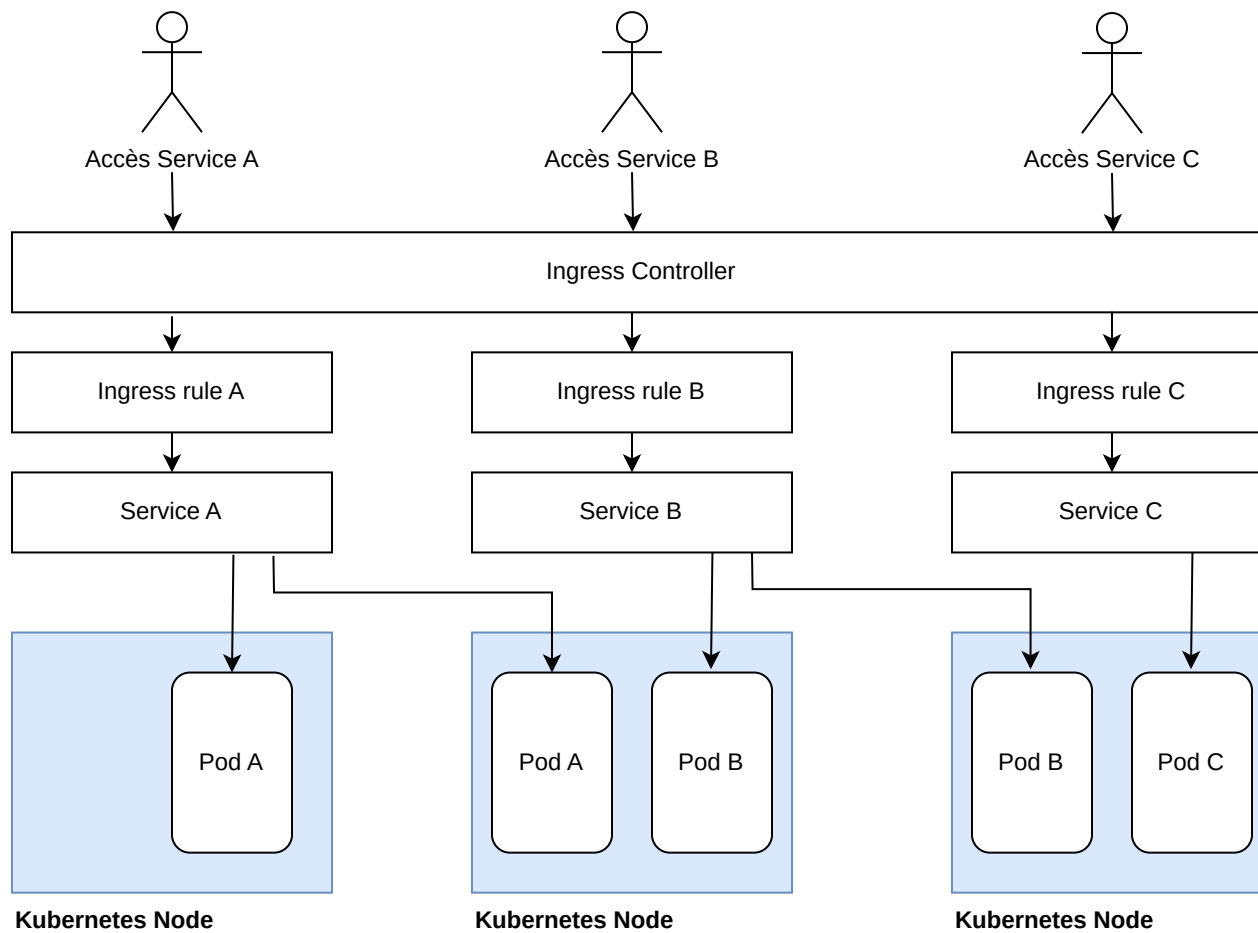
- Unité de déploiement
- Peut contenir 1 ou plusieurs conteneurs

# SERVICES

- Expose le contenu des pods sur le réseau
- Permet d'avoir une représentation du ou des pods d'un même type
- Donne une vision des ports exposés par ces pods

# INGRESS

- Permet de décrire les règles de routage



# NAMESPACE

- Partitionnement de ressources de Kubernetes
- Typiquement utilisé pour regrouper les ressources d'un environnement / application
- Correspond à une sorte de cluster virtuel

# **PERSISTENT VOLUME (PV)**

- Une représentation d'un espace de stockage
- Généralement provisionné par un admin

# PERSISTENT VOLUME CLAIM (PVC)

- Correspond à une requête sur un PV
- On va donc dire que le PVC consomme le PV, en l'occurrence de l'espace disque
- Un PVC va avoir des modes d'accès
  - Read Write
  - Read Only
  - Multiple Read Only : plusieurs pods y accèdent



# CONFIGMAPS

- Permet de stocker des configurations applicatives
- Permet de factoriser la configuration de plusieurs pods (credential bdd)
- Cela peut se matérialiser sous plusieurs formes
  - Une variable d'environnement
  - Un fichier de configuration

# SECRET

- Semblable au ConfigMaps mais stocké dans une sorte de coffre-fort
- Idéal pour stocker des mots de passe ou des certificats

# DEPLOYMENTS

- Pilote la création/suppression des Pods
- Permet de décrire l'état souhaité
  - Image docker
  - Ports
  - ConfigMaps/Secret
  - Nombre de répliques
  - Volumes
  - Stratégie de mise à jour

# OUTILS

# KUBECTL

- Téléchargeable ici :  
<https://kubernetes.io/docs/tasks/tools/>
- Outil de base pour dialoguer avec un cluster Kubernetes
- Permet de consulter l'état du cluster, d'un pod, d'un déploiement.
- Permet de créer des ressources (un pod, un déploiement, ...)
- Permet de supprimer un pod, ...
- Permet de "décrire" des pods, déploiement, service, ...

# ALIAS KUBECTL

- Pour ceux qui sont sur Linux, Mac ou sur Windows avec WSL
- L'utilisation de kubectl peut devenir très verbeuse et rébarbative à la longue
- Il faut très rapidement passer sur des alias

# ALIAS - LA BASE

- Premier alias → remplacer la commande kubectl par la version "gros flemmard" 🤔
- kubectl → k

```
alias k="kubectl"
```

# ALIAS - RÉCUPÉRATION D'INFO

- Pour récupérer des informations sur un élément de Kube, on passe par la commande `kubectl get [...]`
- `kubectl get` → `kg`

```
alias kg="kubectl get"
```



# ALIAS - RÉCUPÉRATION TOUTES LES INFOS DU CLUSTER

- Pour récupérer tout ce que l'on peut voir sur un cluster Kube, on passe par la commande `kubectl get all --all-namespaces`
- `kubectl get all --all-namespaces` → kga

```
alias kga="kg all --all-namespaces"
```

# ALIAS - VOIR L'ÉTAT COMPLET DE TOUS LES PODS D'UN NAMESPACE

- Pour récupérer l'état complet de tous les pods d'un namespace, on passe par la commande `kubectl get pods -o wide`
- `kubectl get pods -o wide → kgp`

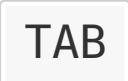
```
alias kgp="watch kubectl get pods -o wide"
```

# ALIAS - VOIR LES SERVICES D'UN NAMESPACE

- Pour récupérer tous les services d'un namespace, on passe par la commande `kubectl get services`
- `kubectl get services → kgs`

```
alias kgs="kg services"
```

# AUTO-COMPLÉTION

- Pour ceux qui sont sur Linux, Mac ou sur Windows avec WSL
- Doc ici : <https://kubernetes.io/fr/docs/tasks/tools/install-kubectl/#configurations-kubectl-optionnelles>
- Permet avec un double  d'avoir des suggestions

# LENS

- Outil complet pour voir toutes les informations et agir sur un cluster
- Permet de se connecter sur les conteneurs des pods très facilement
- Permet de consulter les logs d'un conteneur en un clic
- Téléchargeable ici : <https://k8slens.dev/>

# K9S

- Outil permettant de naviguer facilement dans un cluster, tout ceci dans un terminal
- Utilise la philosophie de VIM (commandes en :, exemple : q pour sortir)
- Intègre un outil permettant de valider les bonnes pratiques
- Téléchargeable ici : <https://github.com/derailed/k9s>
- À déployer sur le cluster ou sur une machine sur le même réseau.

**TP/DEMO**

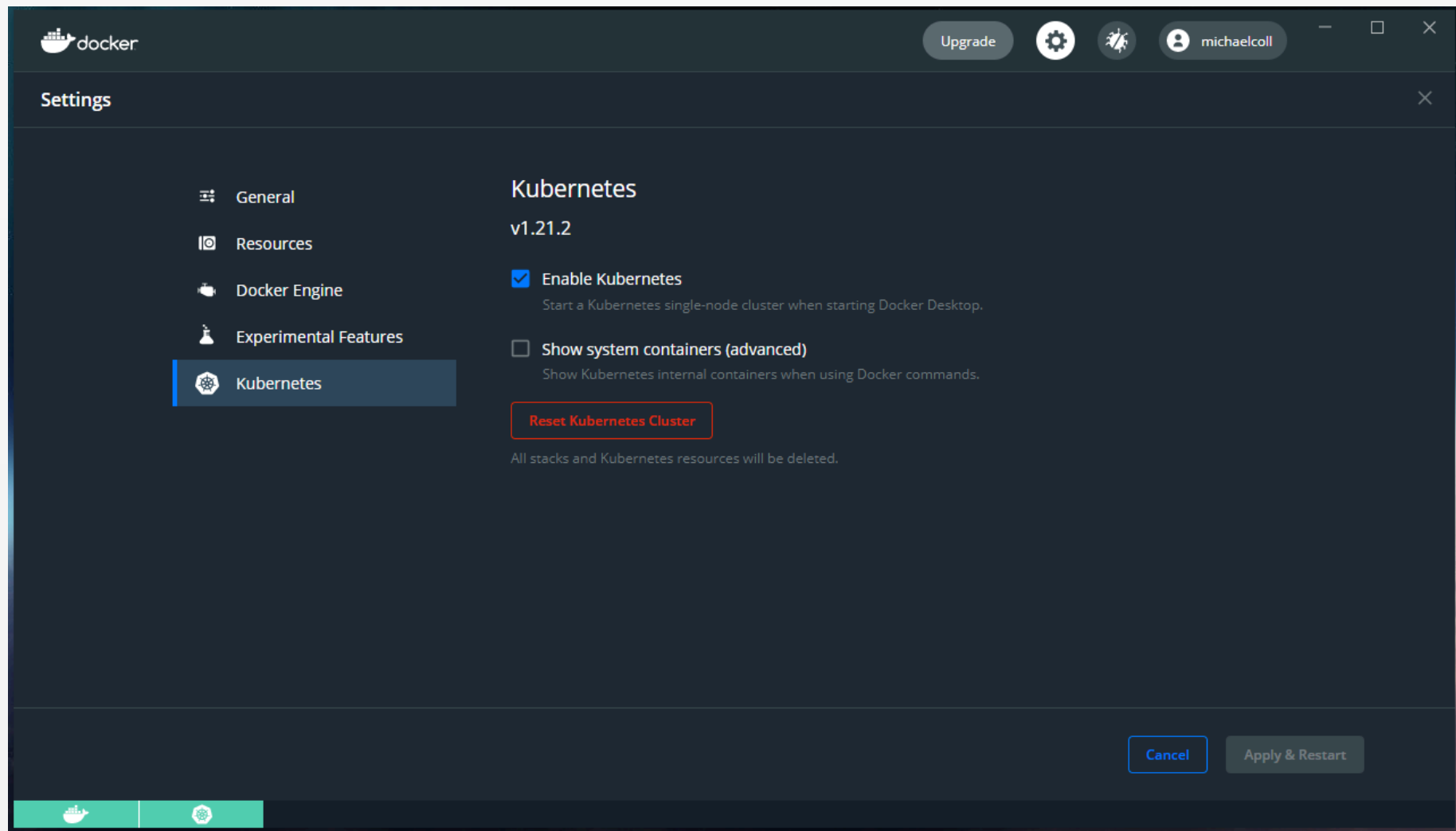
# INSTALL DOCKER

- Suivre les explications de cette doc :  
<https://docs.docker.com/get-docker/>



# LANCEMENT CLUSTER KUBERNETES

- Double cliquer sur l'icône docker à côté de l'heure
- Cliquer sur l'engrenage en haut à droite
- Et dans les settings, aller dans la section Kubernetes
- Cliquer sur la case à cocher : ***Enable Kubernetes***



# LANCEMENT D'UN CLUSTER KUBERNETES SOUS LINUX

- Installer minikube :  
<https://kubernetes.io/fr/docs/tasks/tools/install-minikube/>
- Démarrer le cluster avec la commande

```
$ minikube start
```

# PACKAGING D'UNE API

- Aller dans le dossier tp12-demo-kube
- Construire l'app avec mvn

```
$ mvn package
```

# CRÉATION DE L'IMAGE DOCKER AVEC DOCKERFILE

- Avec ce Dockerfile

```
FROM azul/zulu-openjdk-alpine:17-jre-headless

ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar

ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- Construire une image docker

```
$ docker build . -t tp12:latest
```

- Pour les utilisateurs de minikube

```
$ minikube image build . -t tp12:latest
```

# CRÉATION DE L'IMAGE DOCKER AVEC SPRING BOOT

```
$ mvn spring-boot:build-image
```

# LANCEMENT DE NOTRE NOUVELLE IMAGE

```
$ docker run --rm -ti -p 8080:8080 tp12
```

# DÉPLOIEMENT SUR KUBERNETES

```
$ k apply -f kubernetes.yml
```



# TESTER LE LOAD BALANCER

- Modifier le fichier `kubernetes.yml` comme indiqué

```
1 spec:
2   selector:
3     matchLabels:
4       app: tp12-service
5   replicas: 2
6   template:
7     metadata:
8       labels:
9         app: tp12-service
```

- Ré-appliquer le fichier `kubernetes.yml`

```
$ k apply -f kubernetes.yml
```

# TESTER LE LOAD BALANCER

```
$ while true; do curl `minikube service tp12-service --url -n tp12`/hello/random ; sleep 1;
```

**QUESTIONS ?**

