POURQUOI KUBERNETES ?

#why

Les challenges du DevOps

L'opposition DevOps

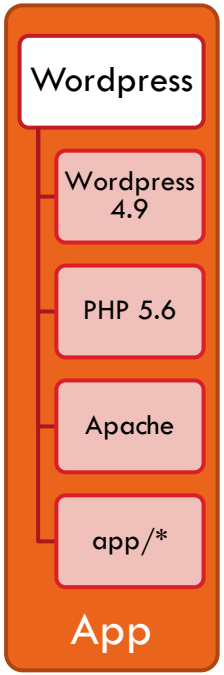
Les Devs veulent

- Déploiements et mise à jour fréquents
- Créer facilement des ressources

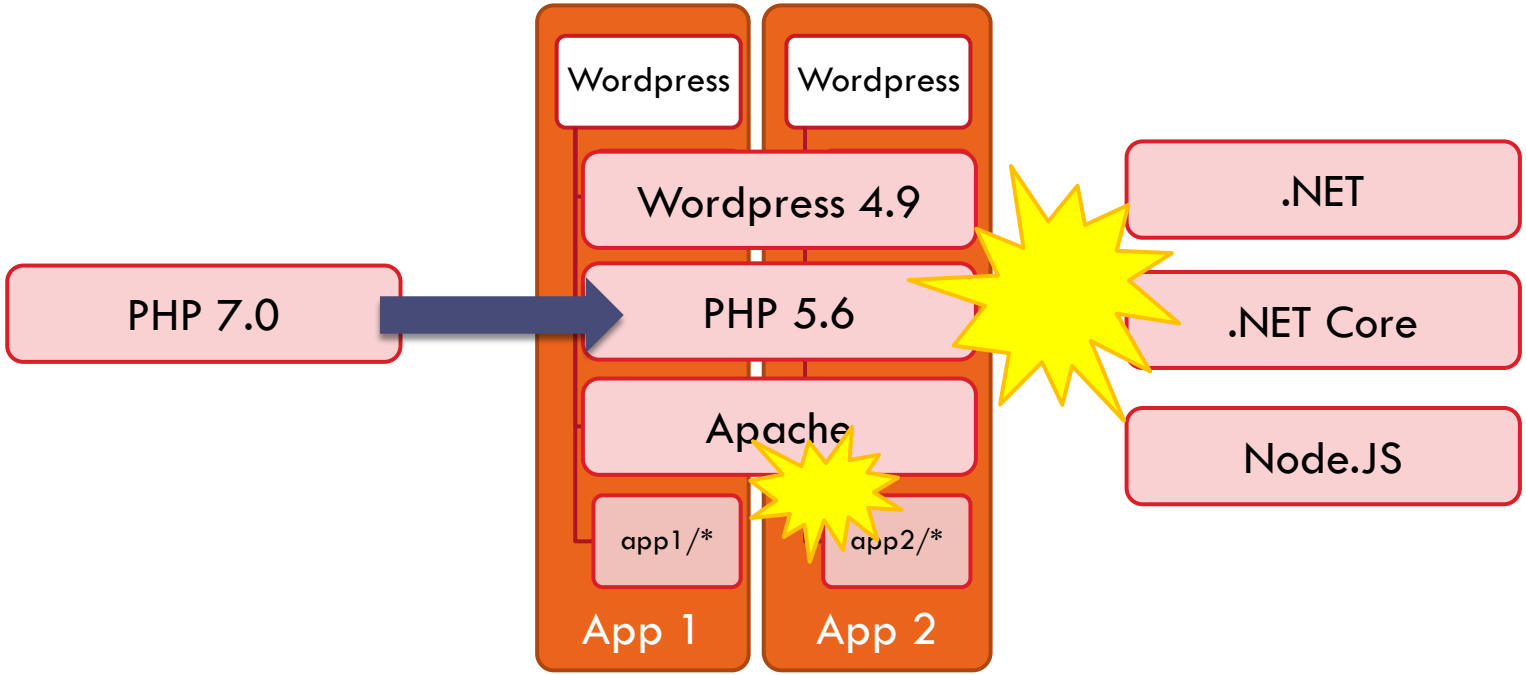
Les Ops veulent

- Stabilité des applications de production
- Supervision
- Contrôle
- Gerer l'infrastructure, pas des applications

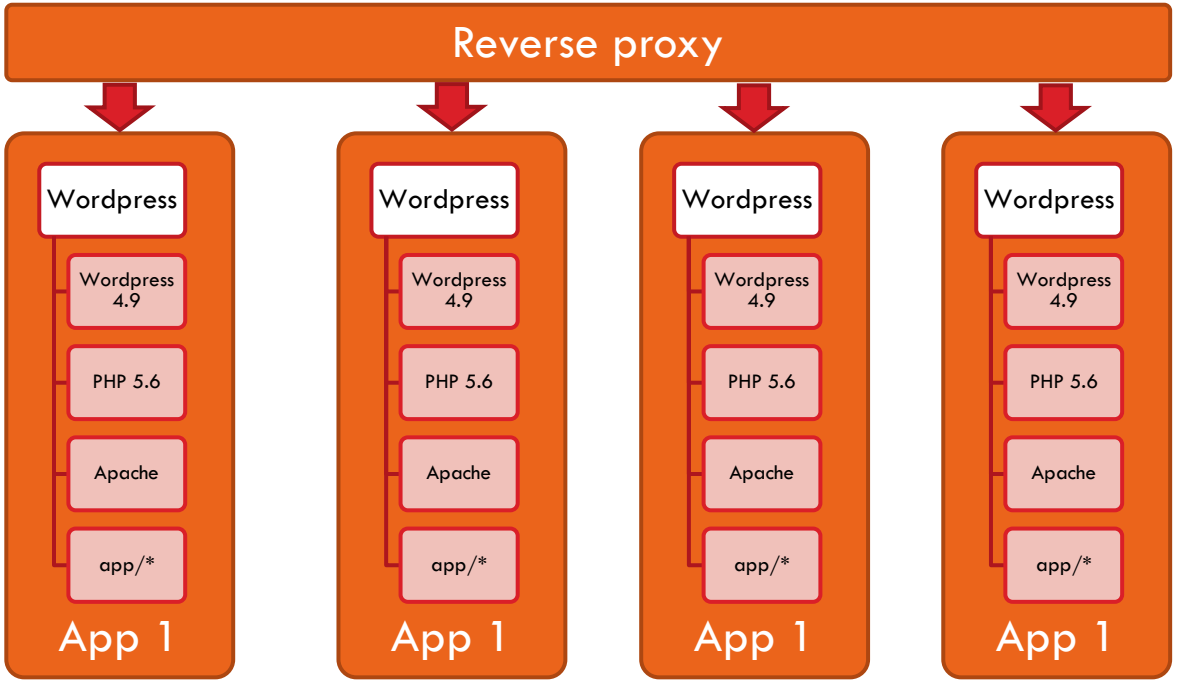
Application typique (simplifiée)



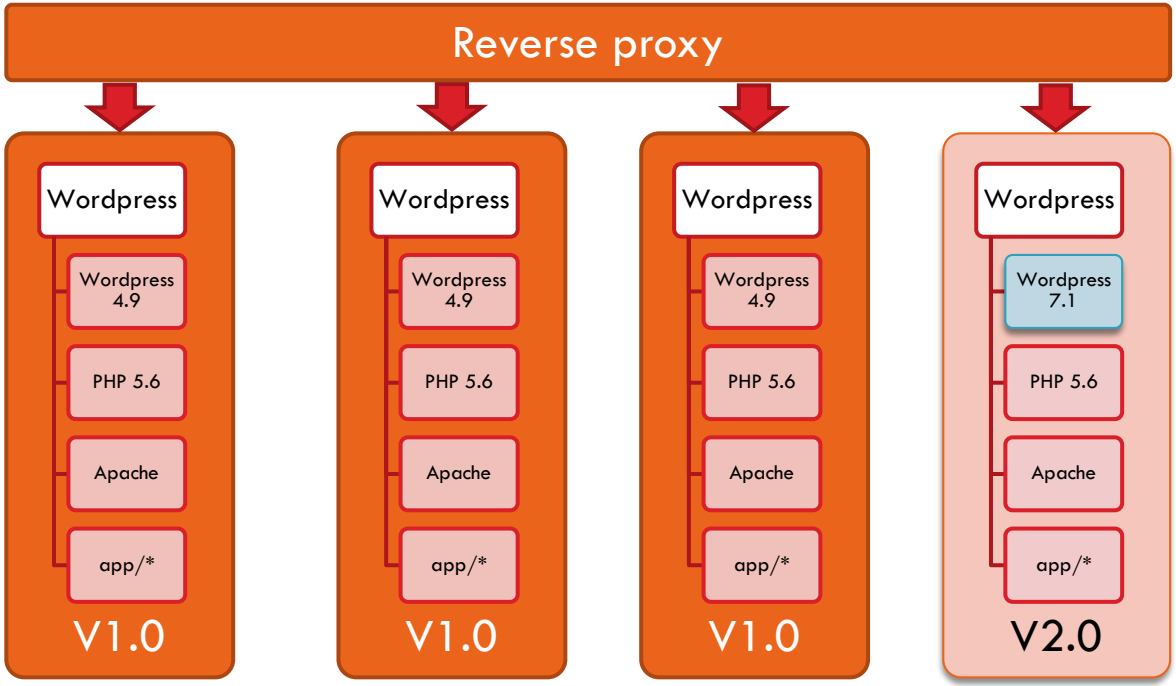
Exemple de serveur (simplifié)



Haute disponibilité



…et mise à jour



Les conteneurs

En bref

- Containers: virtualisation légère
  - Isolation
  - Reproductibilité
  - MAIS
    - Faibles ressources

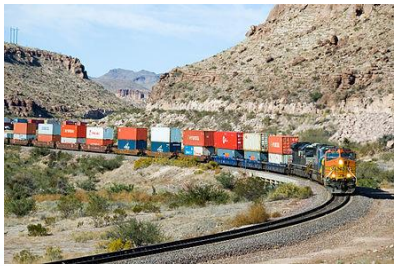
Livrer était déjà compliqué

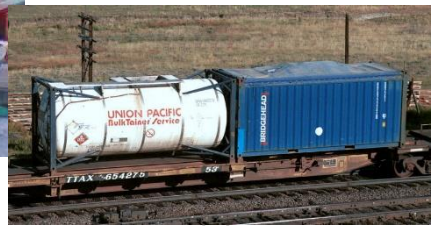
http://aweil.fr

Les conteneurs: le standard

http://aweil.fr

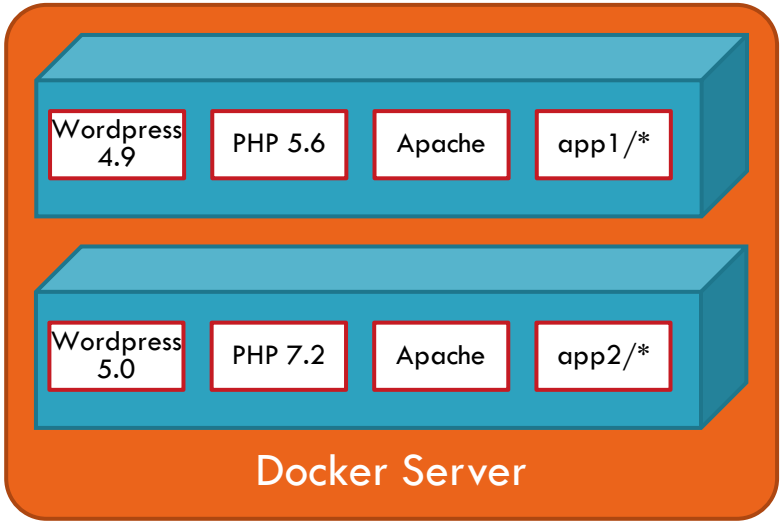
Homogénéité pour tous





?

Les conteneurs sont isolés



Démonstration

- Exécution locale

```
docker run -p 8085:80 nginx
```

Avez-vous remarqué ?

- Boîte noire pour les Ops
- Multi-techno: .NET Core, PHP, Node.JS, Ruby, Java, Python, ...
- Hébergement n'importe où: Hyper-V, Linux, Azure, AWS, Bluemix, hébergeur, cloud privé, ...

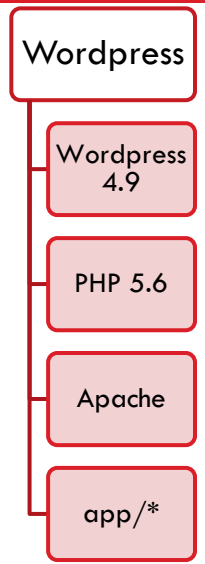
Conteneurs sous Docker

Docker

- Largement répandu
- Open source avec support niveau entreprise
- Linux, Windows, Mac

Images

Une application…



…devient une image

mon-wordpress:1.0

Images

Une image…

mon-wordpress:1.0

Images

Une image…

mon-wordpress:1.0

…produit de nombreux containers

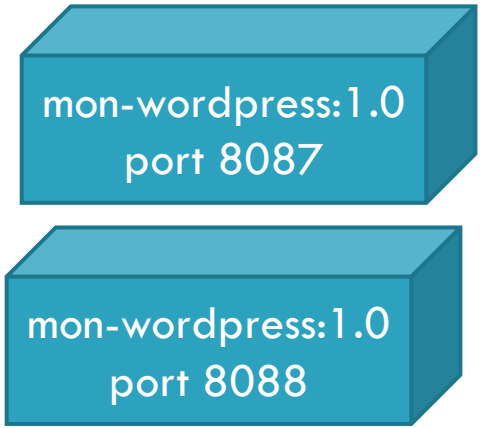
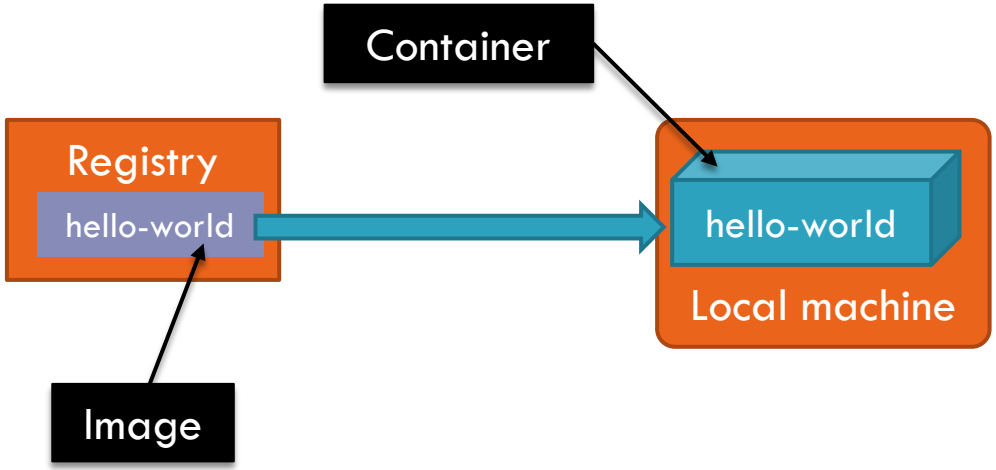


Schéma de principe

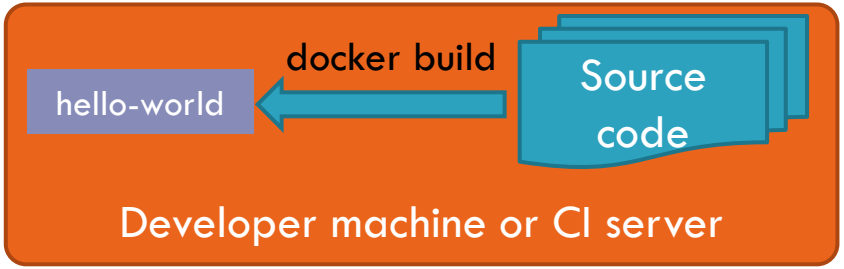


Pour créer une image: fichier Dockerfile

```
FROM wordpress:4.9.4-php5.6-apache

COPY ./sources /var/www/html
```

- …qui référence une autre image

Schéma de principe

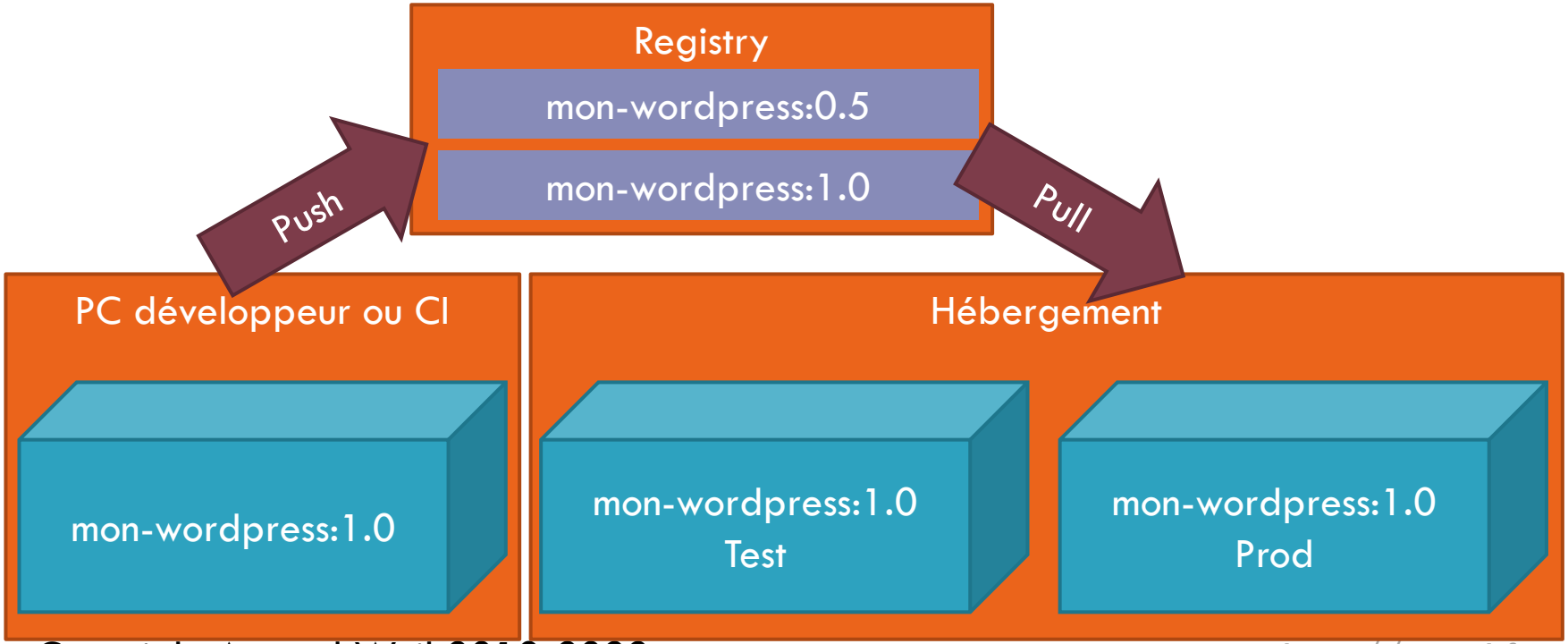


Démo

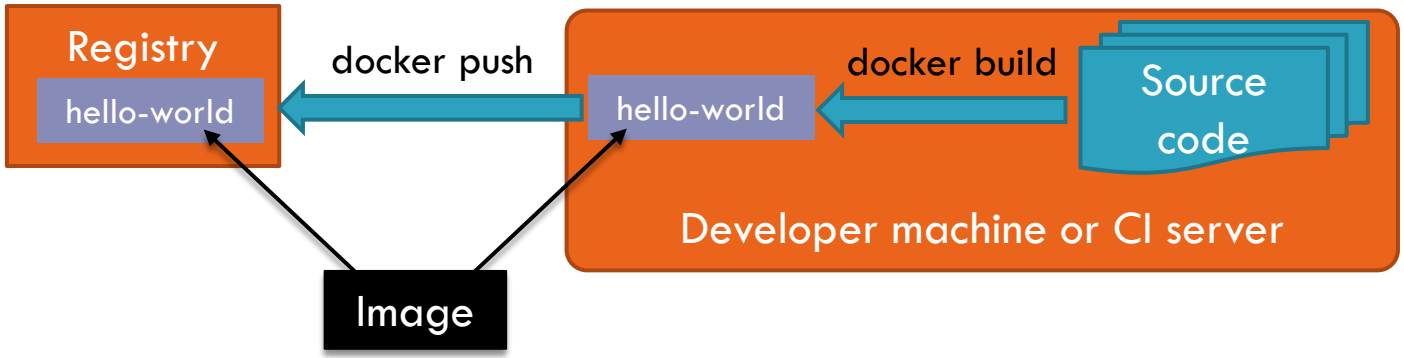
- Création d'une image NGINX avec contenu statique
- Création d'une image Node.JS avec sortie texte

Déployer ses images ailleurs

Déploiement via une Registry



En pratique



Démonstration

- Publication dans une Registry

Orchestration de conteneurs

Challenges

- Applications sur plusieurs conteneurs
- Applications sur plusieurs machines
- Gestion des resources
- Scaling
- Mises à jour
- Changement d'hébergeur

…résolus par l'orchestration

- Kubernetes: orchestration de conteneurs

  "platform for automating deployment, scaling, and operations of application containers across clusters of hosts"

  https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Documentations pour démarrer

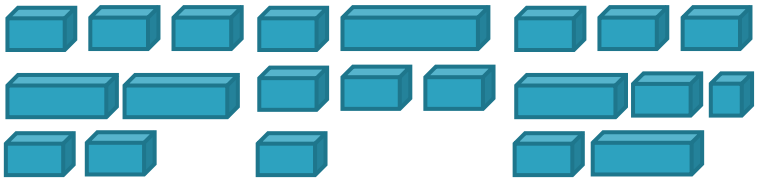
- Glossaire
  - https://kubernetes.io/docs/reference/glossary
- Référence
  - https://kubernetes.io/docs/reference/kubernetes-api/

CLUSTER KUBERNETES

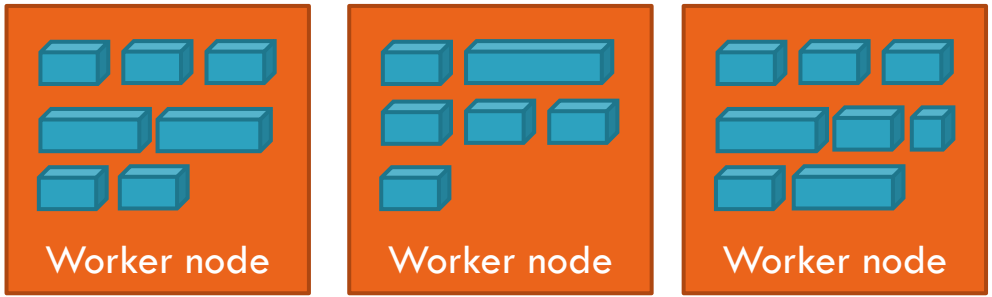
#cluster

Principe

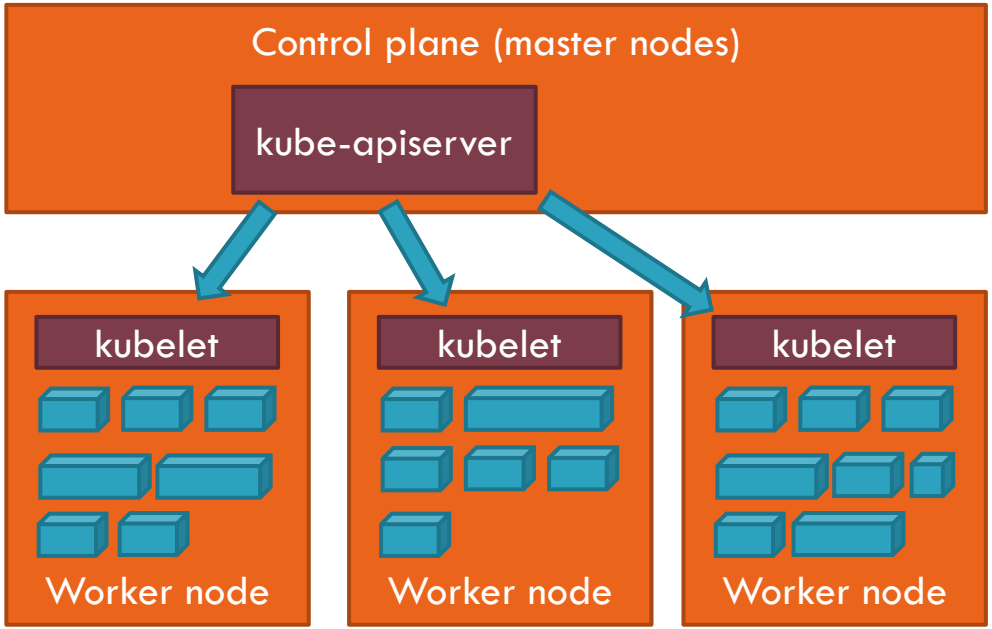
Exécuter des containers (pods)



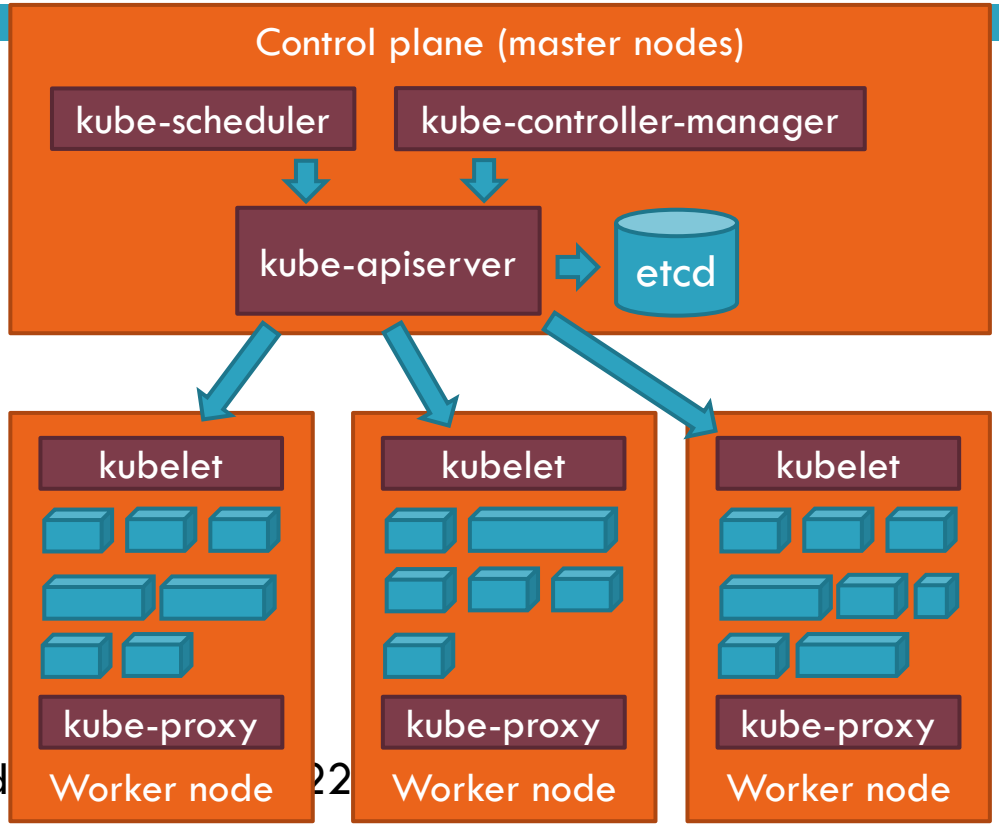
Sur des machines physiques



Supervision



Big picture


Control plane (master nodes)
kube-scheduler
kube-controller-manager
kube-apiserver
etcd
kubelet
kube-proxy
Worker node
kubelet
kube-proxy
Worker node
kubelet
kube-proxy
Worker node

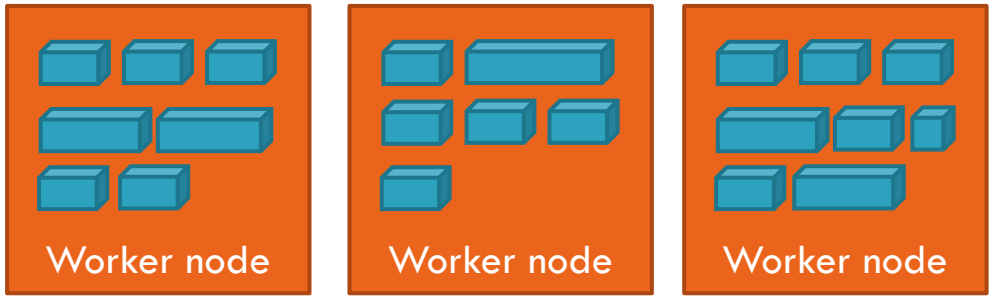
Control plane

- Exécute les composants
- Plusieurs nœuds pour la résilience
  - etcd a besoin de la moitié des master nodes
  - Trois minimum en production

Worker nodes

- Non critiques
  - Le scheduler peut réassigner des pods
- Nombre
  - Suffisant pour que la perte de l'un soit acceptable
  - Limité pour que la charge utile soit correcte

Virtualisation des ressources



- Pods
  - besoin de CPU et de RAM
- Nodes
  - offrent CPU et RAM

Création d'un cluster

Types

- Poste de développement
- Cloud PaaS
- Privé

Poste de développement

- Docker Desktop (one-stop)
- Minikube (choix de la version K8s, du runtime)

Cloud PaaS

- Amazon EKS
  - Facture les master nodes
- Azure AKS
  - Master nodes gratuits
  - Auto provisioning des worker nodes
  - Auto scaling des worker nodes
- Google GKE
  - Master nodes gratuits
  - Auto provisioning des worker nodes
- RedHat OpenShift, IBM CKS, Heptio Kubernetes Subscription

Privé

- Kubeadm
  - kubeadm init, join, upgrade, reset
  - Composable
- Kubespray
  - Clouds ou bare metal
  - Utilise Ansible ou Vagrant
- Kops
  - AWS, GCE

Attention

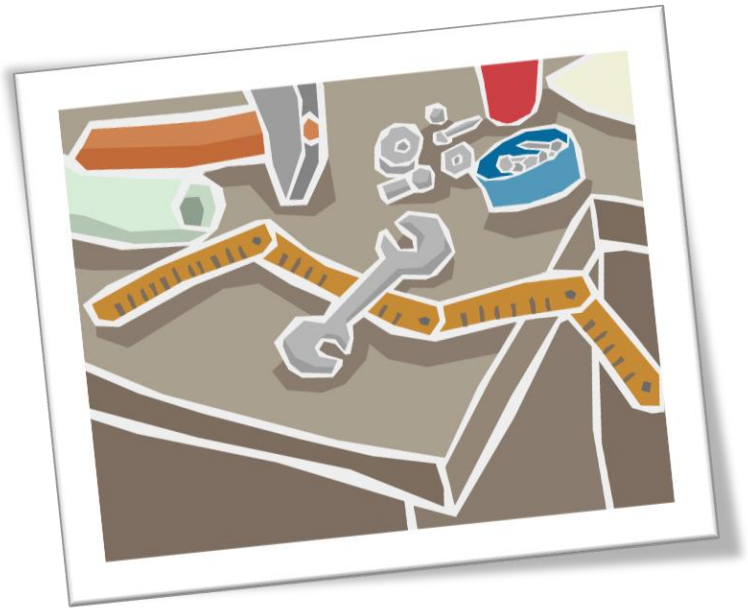
- Gérer un cluster privé n'est pas facile
  - Haute disponibilité du control plane
  - Extensibilité / réparation automatique des worker nodes
  - Sécurisation (communication inter nœuds et avec etcd)
  - Mises à niveau
  - Sauvegarder etcd
  - Sauvegarde des volumes

Démonstration

- Docker Desktop
- Azure AKS

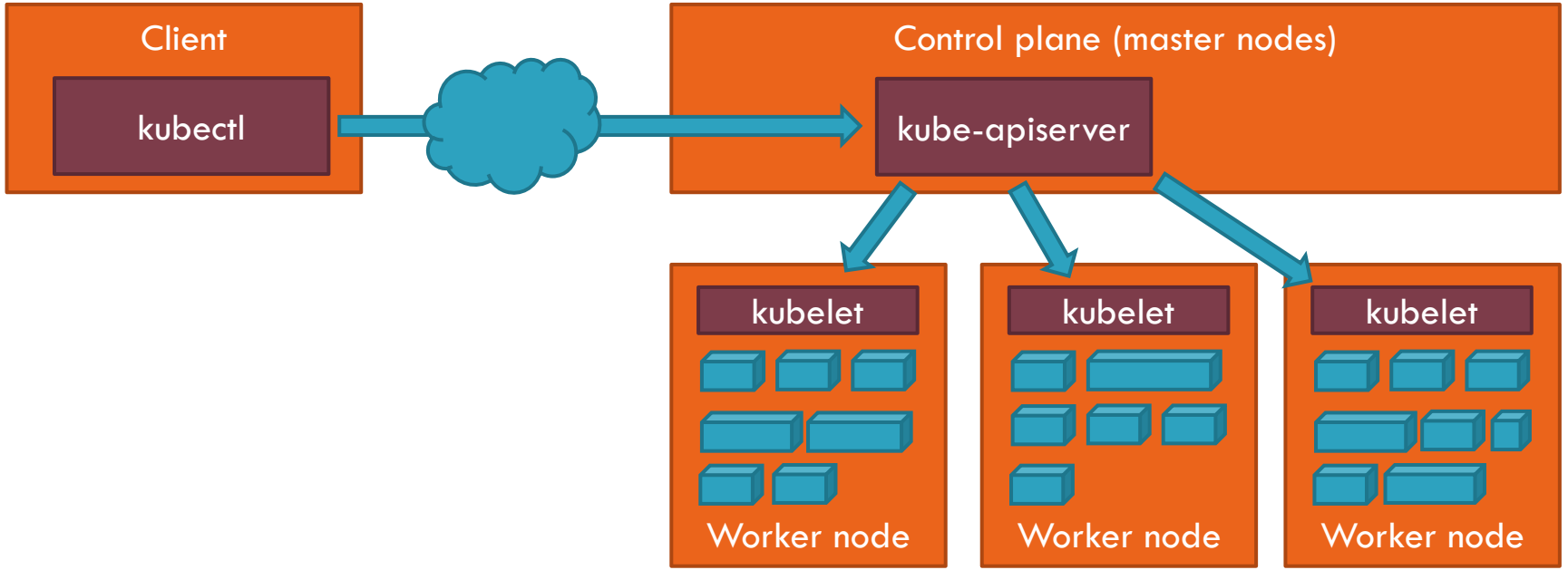
Travaux pratiques

- Objectif : avoir un cluster à disposition.
- #cluster / setup



Connexion au cluster

kubectl



Obtenir kubectl

- Exécutable:
  - https://kubernetes.io/docs/tasks/tools/install-kubectl

```
$ kubectl version

Client Version: version.Info{Major:"1", Minor:"14", ...}
Server Version: version.Info{Major:"1", Minor:"14", ...}
```

Contextes

- Contexte : cluster + credentials

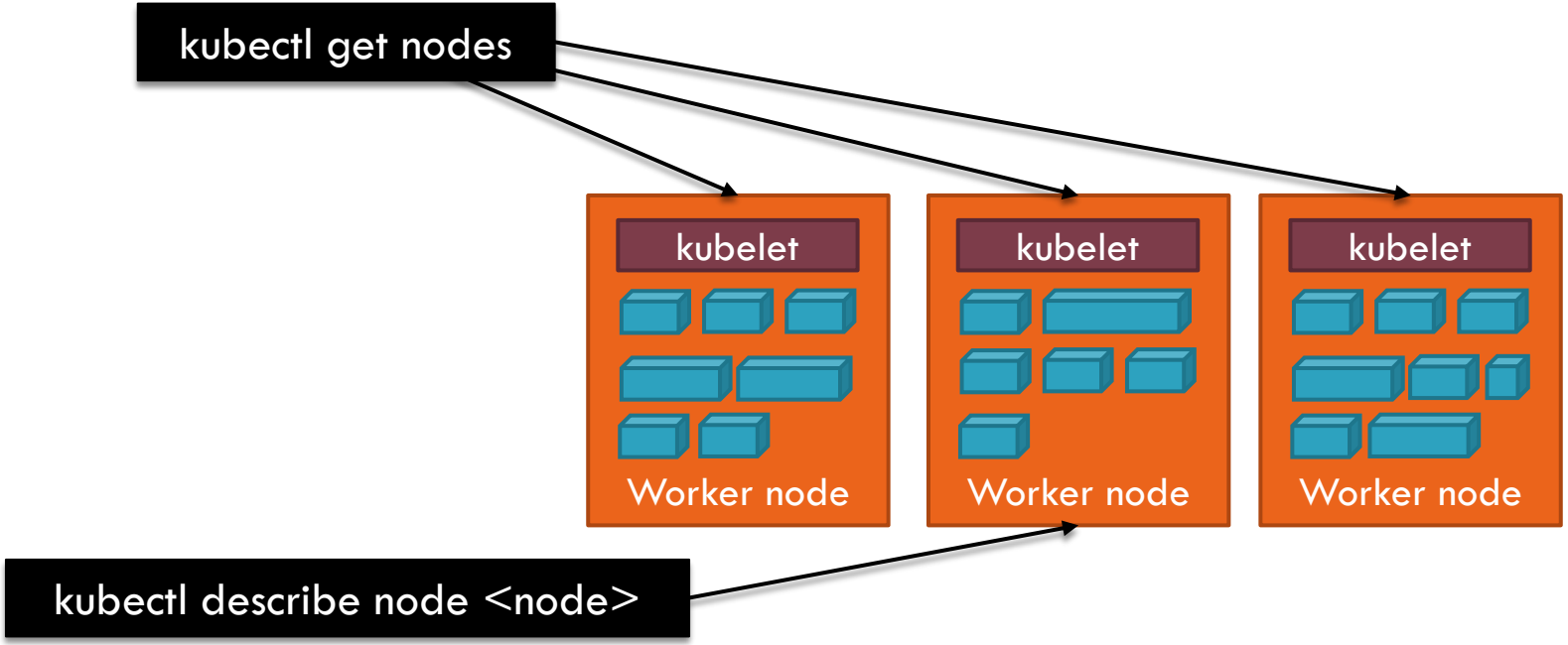
```
> kubectl config get-contexts

CURRENT   NAME
          some-cluster
          anotherOne
*         docker-desktop
          cloudk8s

> kubectl config use-context anotherOne
```

Contextes

- Récupérer un contexte AKS

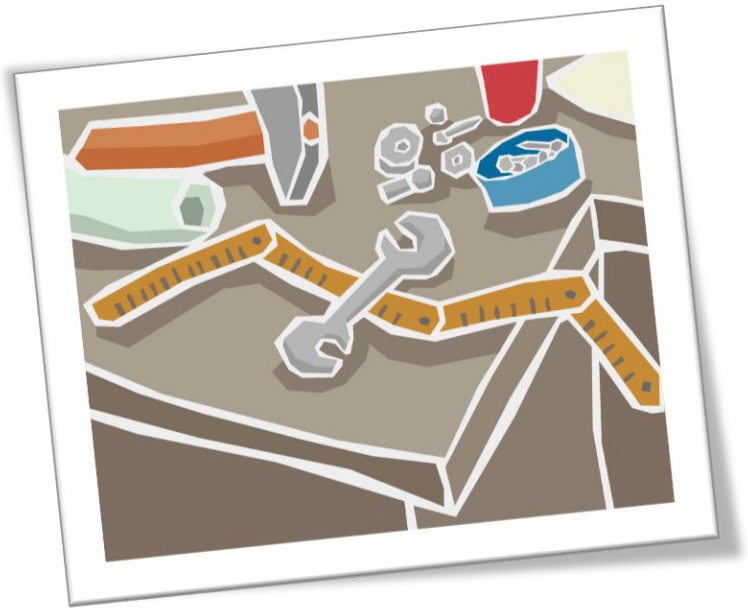
```
> az login

> az aks install-cli

> az aks get-credentials --resource-group
myResourceGroup --name myAKSCluster
```

Utilisation de base



Travaux pratiques

- Objectif : voir les éléments constituant le cluster.
- #cluster / inspect

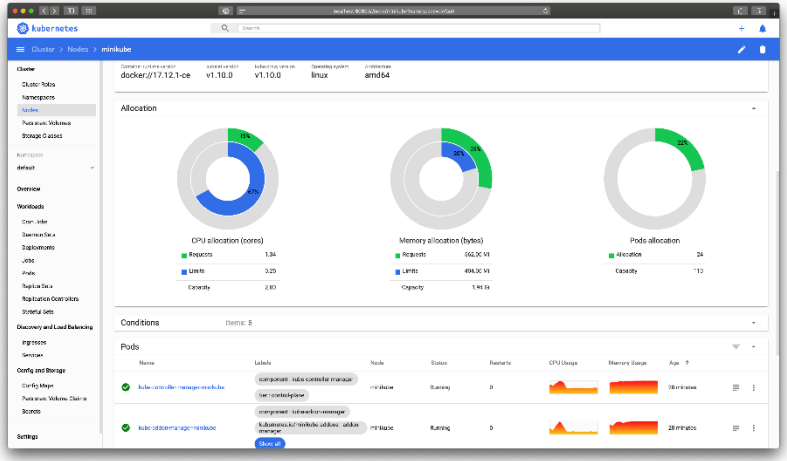


OUTILLAGE

#tooling

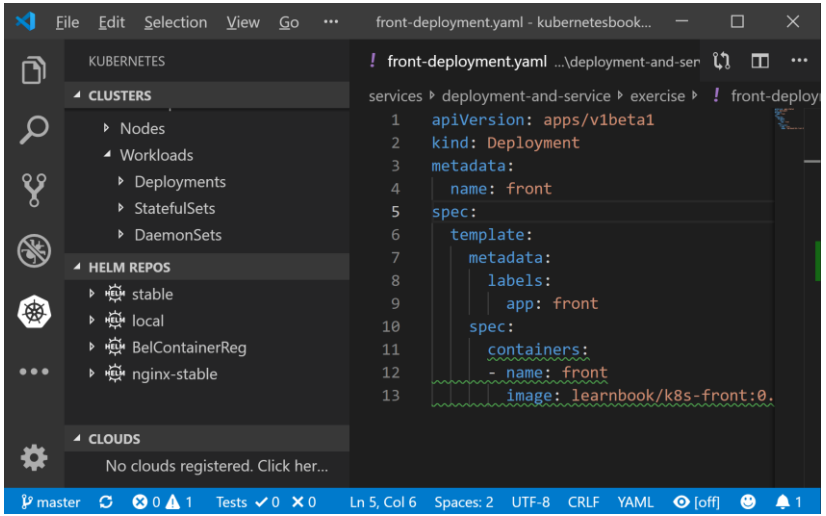
Dashboard

- Equivalent GUI de kubectl
- https://github.com/kubernetes/dashboard/



Visual Studio Code

- Extension « Kubernetes »
  - Autocomplétion
  - Visualisation des ressources

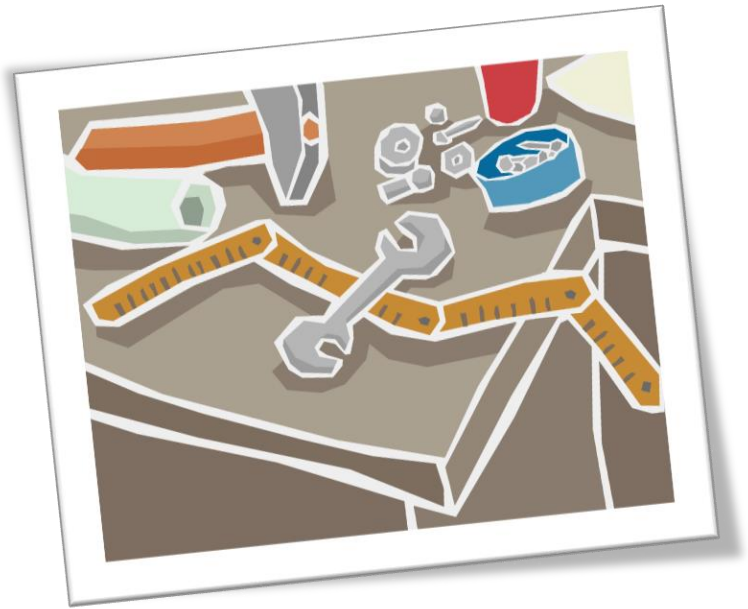


Démonstration

- Extension Kubernetes

Travaux pratiques

- Objectif : installer Visual Studio Code et son extension Kubernetes (optionnel).
- #tooling / vs-code

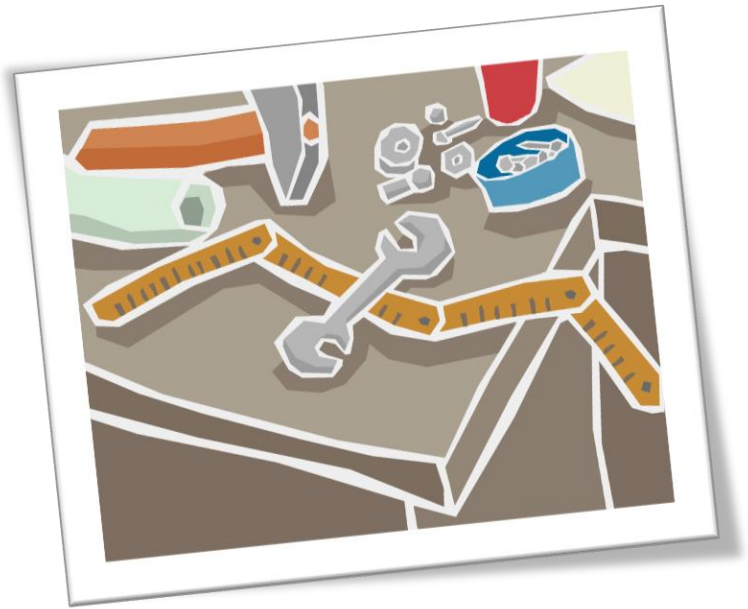


Helm

- Déploiement simultané de plusieurs ressources
- Déploiement paramétrable
- Retour en arrière simple

Travaux pratiques

- Objectif: installer Helm.
- #tooling / helm-install



EXÉCUTION DE PODS

#pods

Démonstration

- Solution des exercices

https://bitbucket.org/epobb/kubernetes-exercises

Schéma de principe

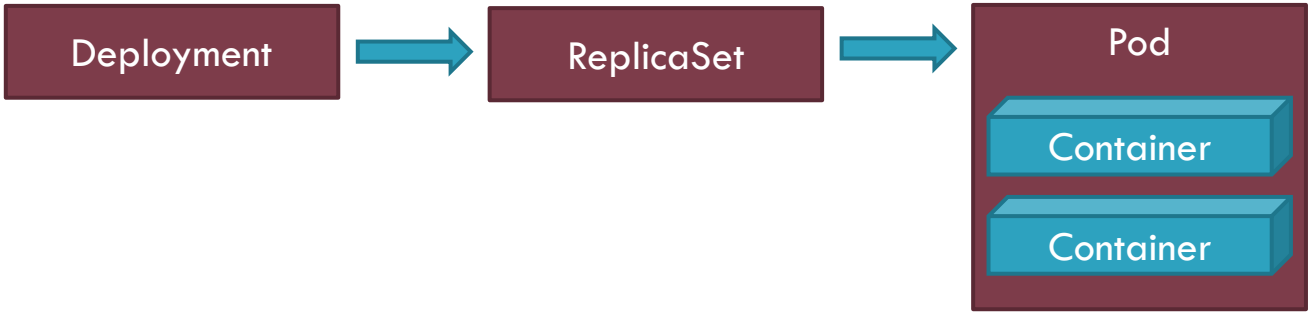
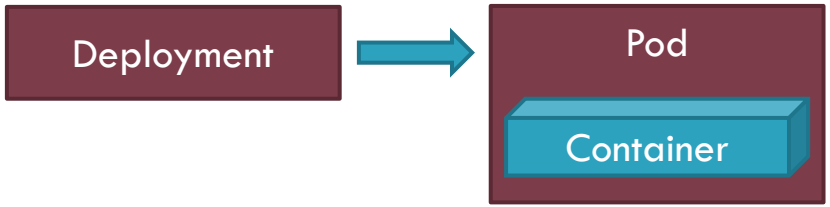


Schéma de principe simplifié



Exécuter un pod

- Deux façons
  - Impérative
  - Déclarative

Exécution impérative

Exécution impérative

- Simple

```
kubectl create deployment hello --image=hello-world
```

- Plantage si le deployment existe déjà

Surveiller un pod

- Voir les pods

```
kubectl get pods -l app=hello
```

- Voir les logs, sélection par le label

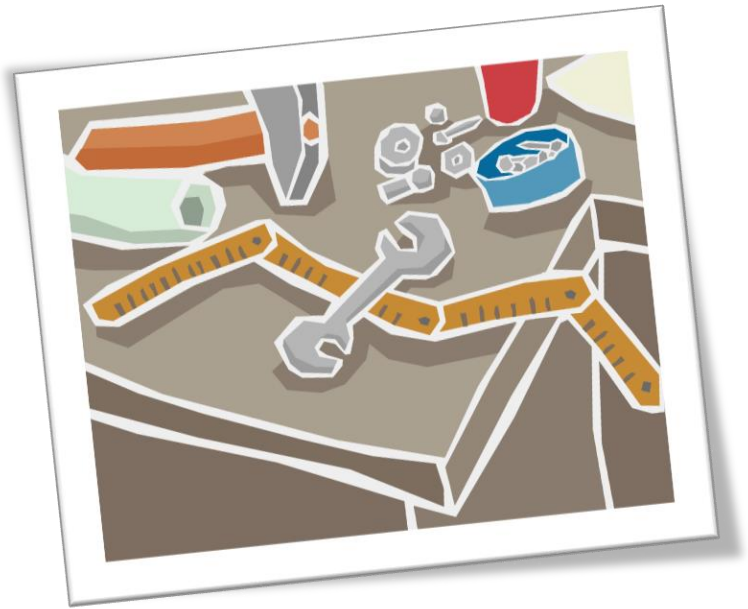
```
kubectl logs -l app=hello
```

Démonstration

- kubectl create
- kubectl get pods
- kubectl logs

Travaux pratiques

- Objectif : prendre en main la création impérative de ressources.
- #pods / imperative-pod-run



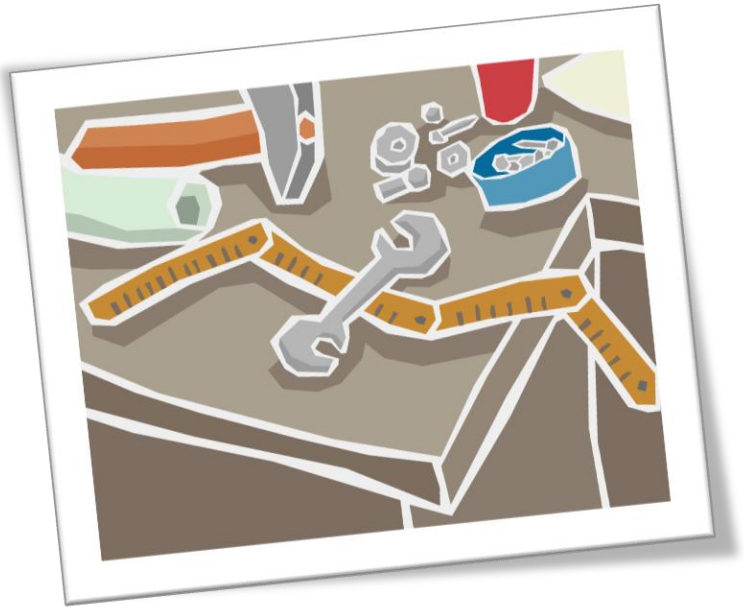
Voir un Deployment

- Current: replicas actuelles
- Up-to-date: replicas reflétant les derniers changements
- Available: disponibles pour utilisateurs

```
$ kubectl get deployments

NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello     4         4         4            0           3s
```

Travaux pratiques

- Objectif : vérifier ce qui se passe en cas de défaillance d'un pod.
- #pods / imperative-deployment-check



Attention aux noms

- Attention aux noms des objets Kubernetes:
  - Minuscules et tirets
  - Moins de 64 caractères

Exécution déclarative

Exécution déclarative

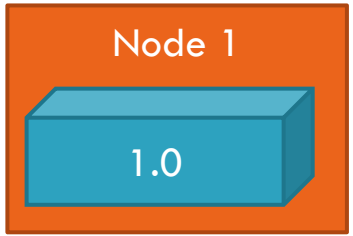
- Semble complexe à la création
- Plus simple en maintenance
- Applique les bonnes pratiques :
  - Infrastructure As Code
  - Single Source Of Truth
- Simple à appliquer dans CI/CD

Deploiement

- Description de l'état souhaité

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-front
spec:
  replicas: 1
  template:
    spec:
      containers:
      - name: mon-front
        image: mon-wordpress:1.0
        ports:
        - containerPort: 80
```

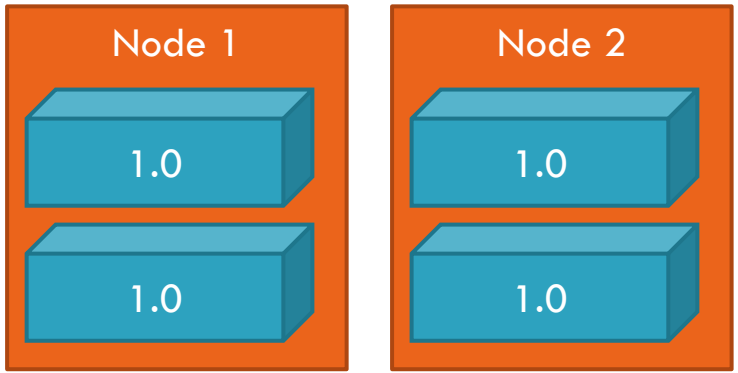
```
kubectl apply -f deploy.yaml
```



Montée en charge

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-front
spec:
  replicas: ~~1~~ 4
  template:
    spec:
      containers:
      - name: mon-front
        image: mon-wordpress:1.0
        ports:
        - containerPort: 80
```

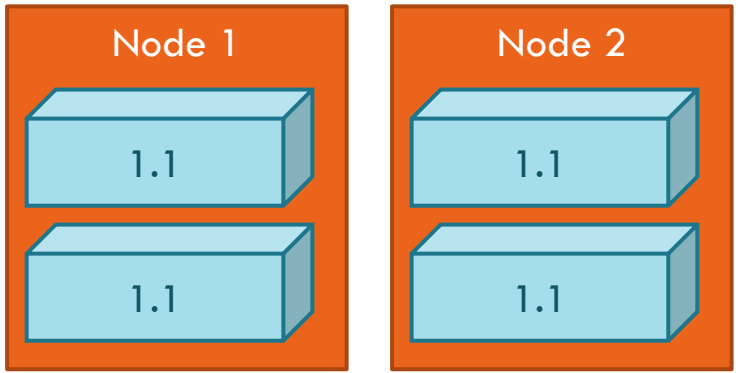
```
kubectl apply -f deploy.yaml
```



Mise à jour

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-front
spec:
  replicas: 4
  template:
    spec:
      containers:
      - name: mon-front
        image: mon-wordpress:~~1.0~~ 1.1
        ports:
        - containerPort: 80
```

kubectl apply -f deploy.yaml



Commandes déclaratives

- Création

```
kubectl apply -f dep.yaml
```

- Mise à jour

```
kubectl apply -f dep.yaml
```

- Suppression

```
kubectl delete -f dep.yaml
```

Astuce

- Sur un répertoire entier

```
kubectl apply -f .
```

- Y compris en suppression

```
kubectl delete -f .
```

Se faire écrire le YAML ?

- Par kubectl:

```
kubectl create <objet> ... --dry-run=client -o yaml
```

- En cas de doute sur les versions

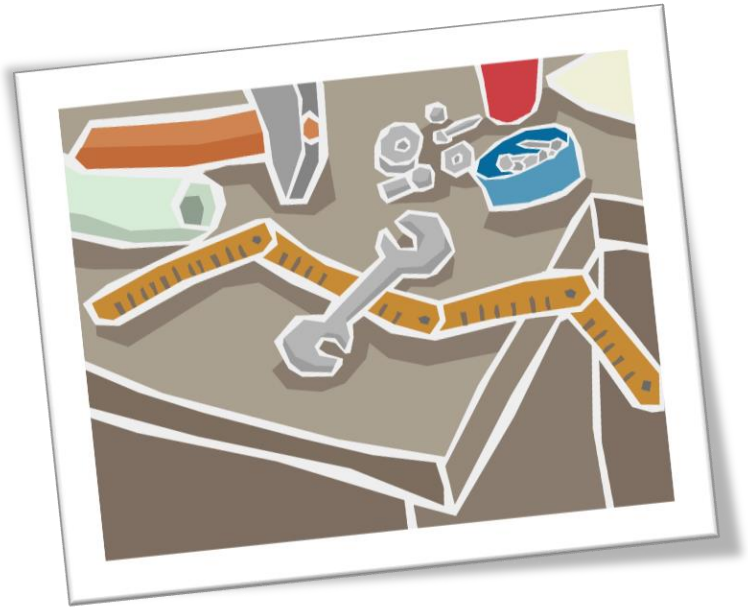
```
kubectl api-versions
```

Démonstration

- Création déclarative d'un déploiement
- Modification
- Suppression

Travaux pratiques

- Objectif : créer des ressources de manière déclarative.
- #pods / declarative-deployment



Passage de variable d'environnement

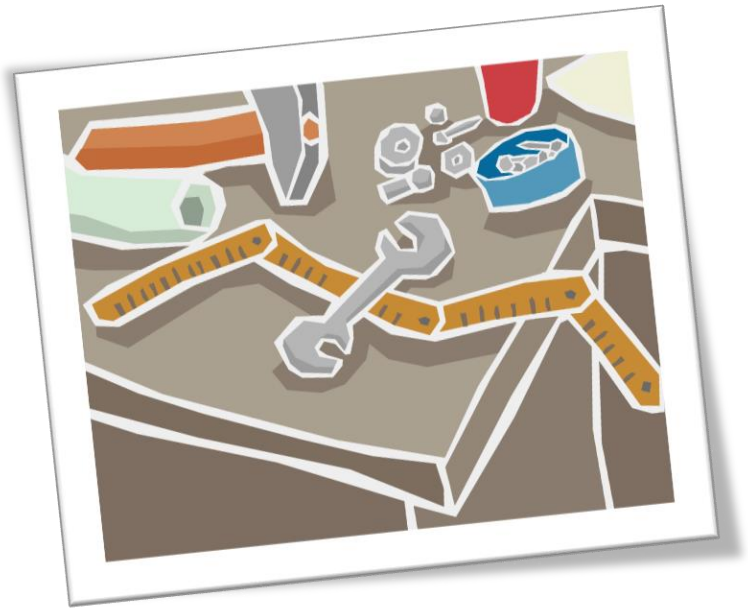
```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: mon-front
spec:
template:
    spec:
      containers:
      - name: mon-front
        image: mon-wordpress:1.0
        env:
        - name: nom
          value: "valeur"
        - name: nom2
          value: "valeur2"
```

Démonstration

- Modification de notre image pour accepter une variable d'environnement
- Passage de la variable d'environnement

Travaux pratiques

- Objectif : mettre à jour des ressources créées de manière déclarative.
- #pods / environment-vars



Controllers

- 5 types pour lancer des pods:
  - ReplicaSet / Deployment: ceux qui ne se terminent pas
  - StatefulSet: idem mais ordre et unicité
  - Job: pour ceux qui se terminent
  - CronJob: idem mais exécution périodique
  - DaemonSet: un par node

Débogage

« Ce qui peut aller mal ira mal »

- loi de Murphy

Erreurs courantes pour un pod

- Image introuvable (pas encore publiée, erronée)
- Plantage du code du conteneur (code de retour en erreur)
- Dans ces cas, le controller recommence en boucle avec des pauses croissantes

Comprendre ce qui se passe

- Inspecter le pod et son historique:

```
kubectl describe pod <name>
```

- Examiner les logs

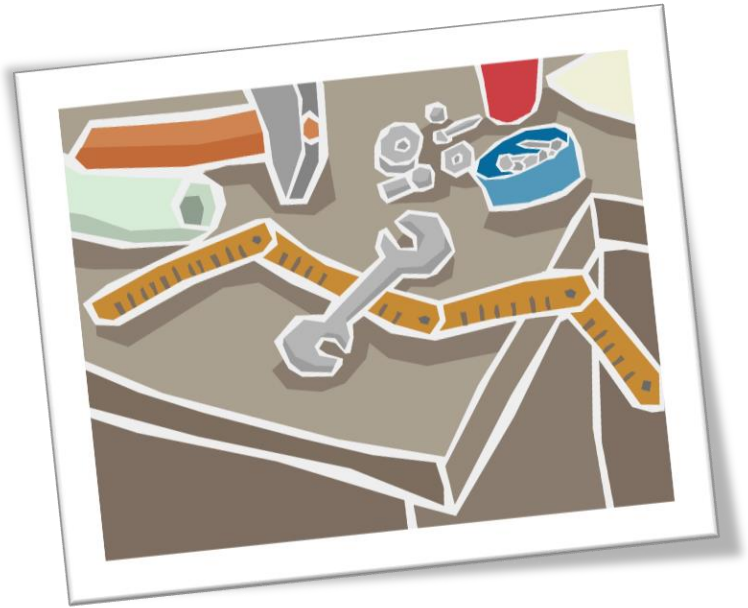
```
kubectl logs -l app=hello
```

- Exécuter un pod de débogage interactif

```
kubectl run -i --tty busybox --image=busybox -- sh
```

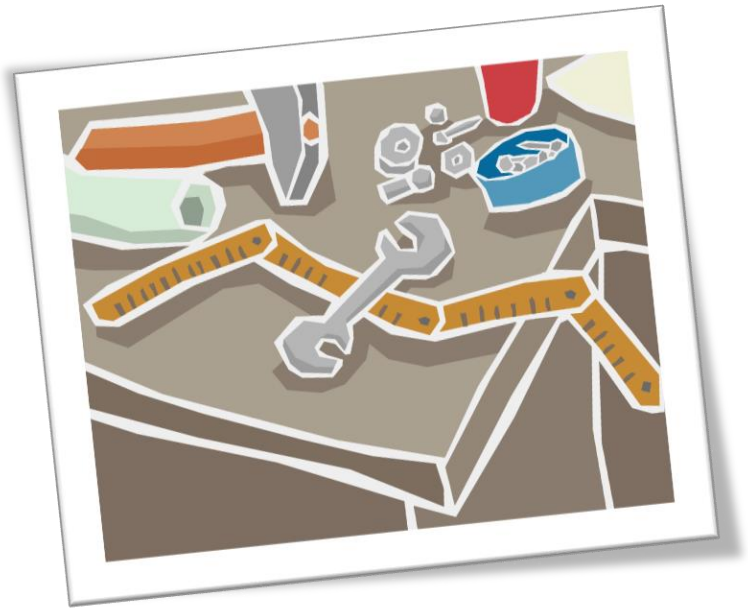
Travaux pratiques

- Objectif : vérifier le comportement de Kubernetes en cas d'erreur et apprendre à identifier la source des erreurs.
- #pods / debugging



Travaux pratiques

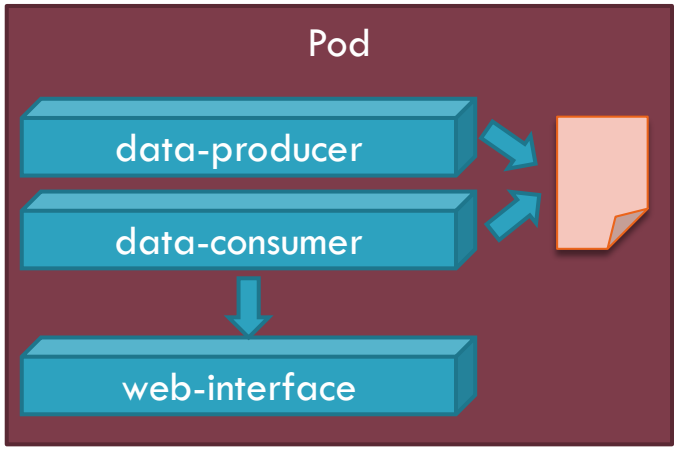
- Objectif : libérer les ressources créées.
- #pods / cleanup



Pod = Container ?

- Souvent oui.
- En fait un pod regroupe des containers liés.

  Exemple:



Gestion des ressources

Requests

- Le minimum nécessaire sur un node pour y scheduler le pod
- Par conteneur
- Des valeurs trop élevées empêchent le scheduling

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
      - name: front
        image: learnbook/k8s-front:0.4
        resources:
          requests:
            cpu: "100m"
            memory: "10Mi"
```

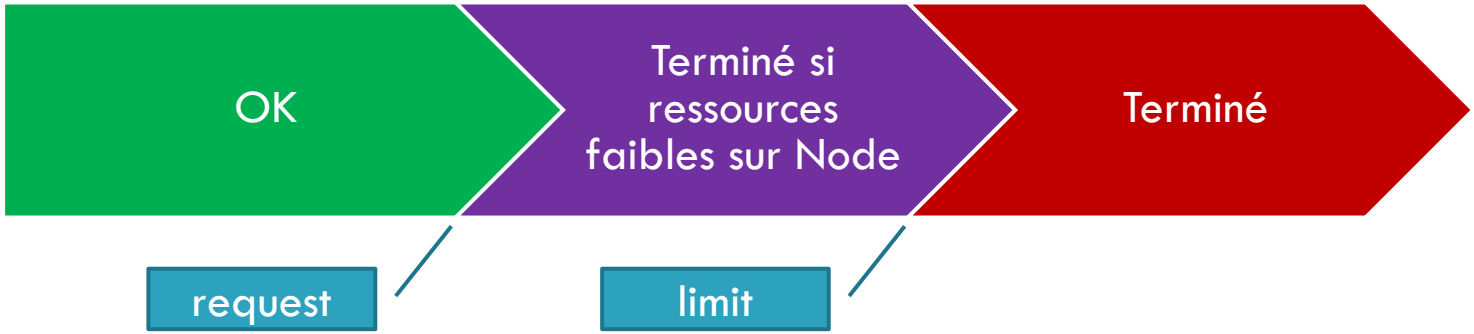
Limits

- Le maximum autorisé
- Quand un Pod les dépasse, il est terminé

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
      - name: front
        image: learnbook/k8s-front:0.4
        resources:
          limits:
            cpu: "200m"
            memory: "50Mi"
```

Libération des ressources

- Lorsque les ressources sont faibles sur un Node, candidats à la suppression
  - Ceux qui dépassent le plus leurs requests



Gérer l’état d’un Pod

Probes

- Liveness : le Pod doit-il être redémarré ?
- Readyness: le Pod est-il prêt à recevoir des requêtes ?
- Probes possibles : command, http, tcp

```
apiVersion: apps/v1beta1
kind: Deployment
spec:
  template:
    spec:
      containers:
      - name: front
        image: learnbook/k8s-front:0.4
        livenessProbe:
        exec:
          command:
          - cat
          - /tmp/healthy
        initialDelaySeconds: 10
        periodSeconds: 5
```

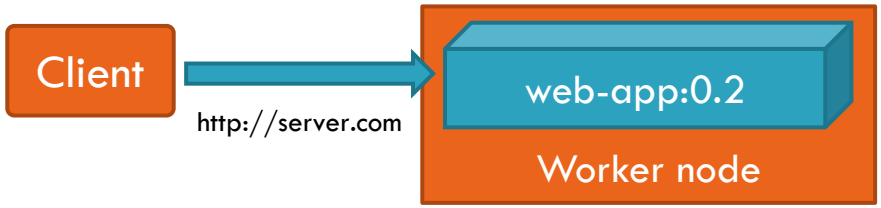
SERVICES

#services

Problématique

Problématique 1

- Comment permettre au monde extérieur d'accéder au réseau de mes pods ?



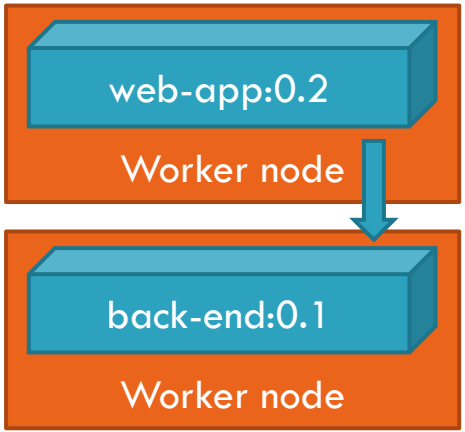
Solution impérative

- Dans un environnement de développement

```
kubectl port-forward deployment/nom 8085:8080
```

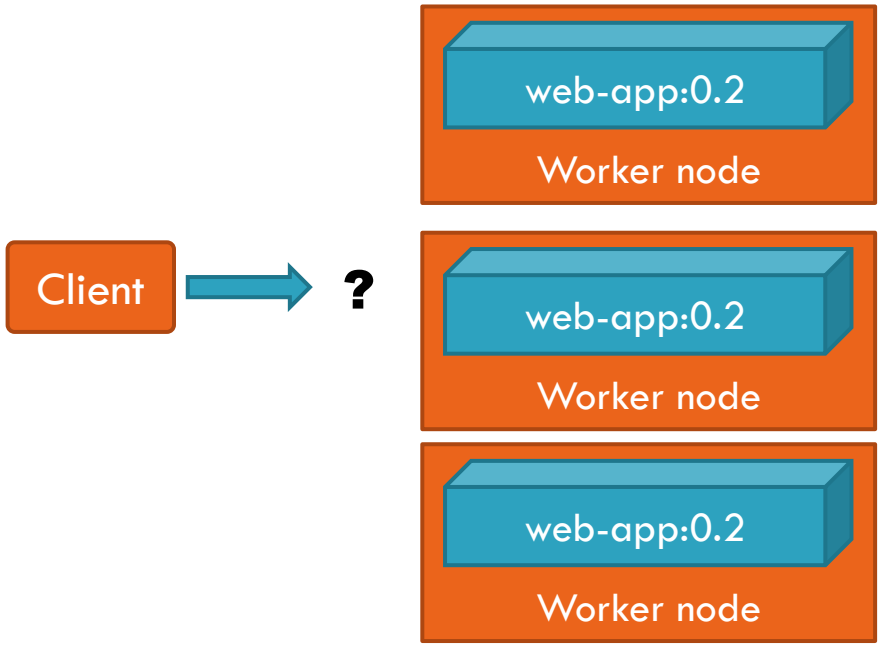
Problématique 2

- Comment permettre aux pods sur des nodes différents de communiquer ?



Problématique 3

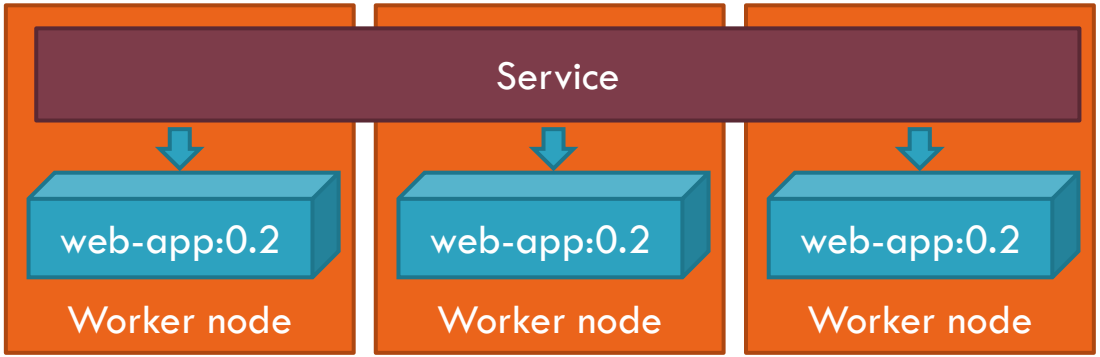
- Comment faire quand on a plusieurs instances réparties sur des nodes différents ?



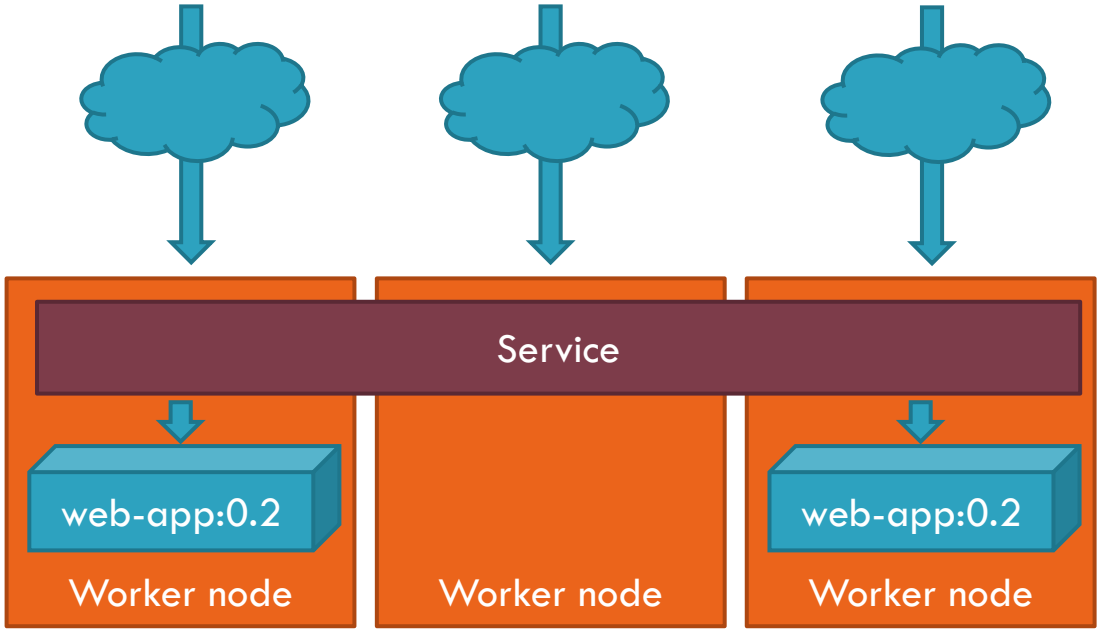
Services

Service

- Effectue le load-balancing
- Sélectionne d'après des labels

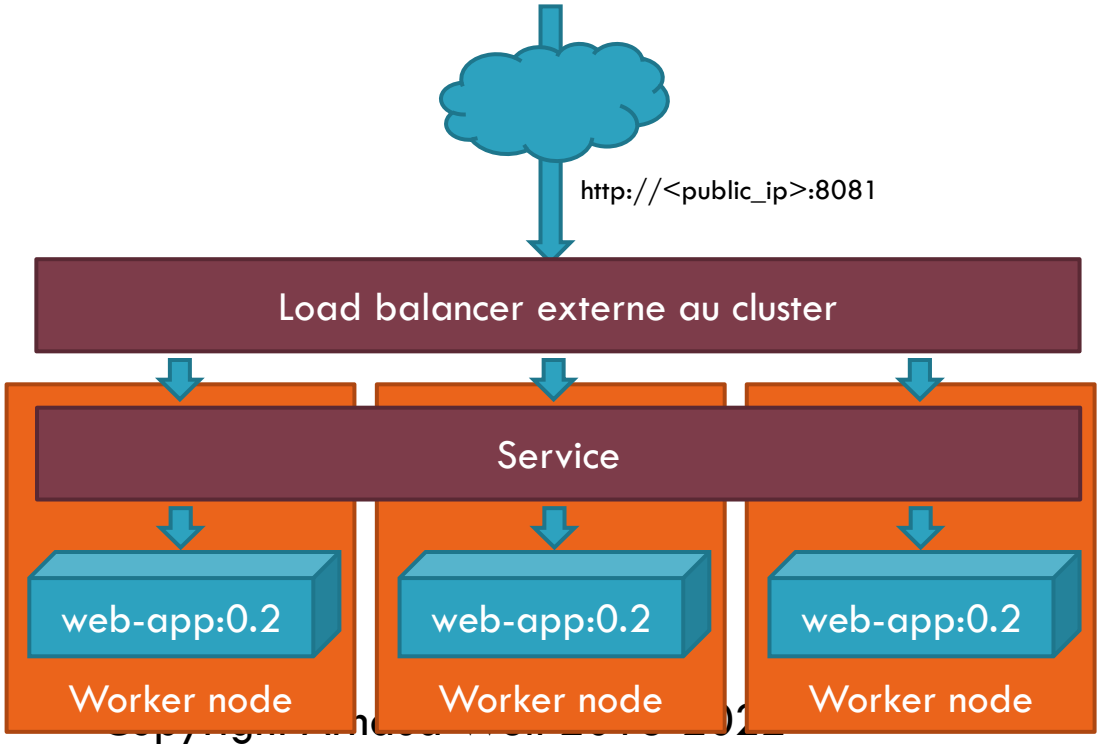


Ouvert sur l'extérieur




```
apiVersion: v1
kind: Service
metadata:
  name: mon-front-service
spec:
  type: NodePort
  ports:
  - port: 80
    targetPort: 80
    nodePort: 30000 # ou assigné
  selector:
    app: web-app
```

Ouvert sur l'extérieur




```
apiVersion: v1
kind: Service
metadata:
  name: mon-front-service
spec:
  type: LoadBalancer
  ports:
  - port: 8081
    targetPort: 80
  selector:
    app: web-app
```

Démonstration

- Déploiement de adminer avec un service LoadBalancer

Minikube

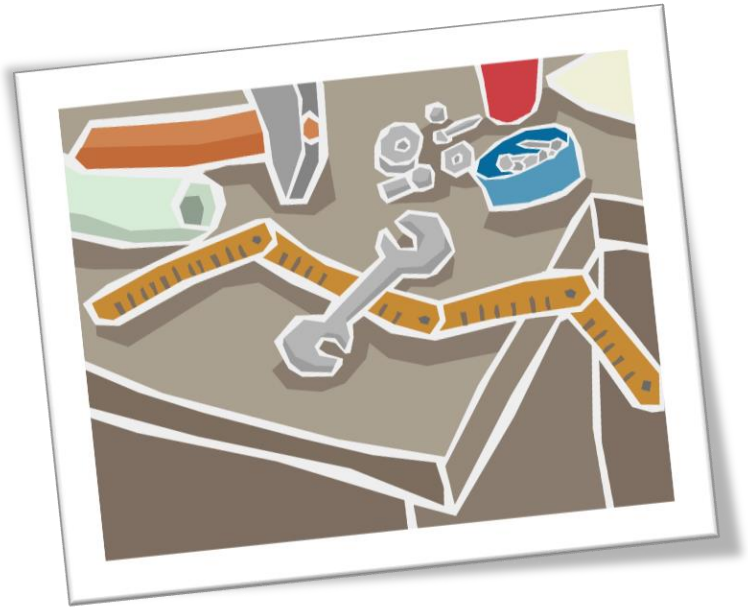
- Besoin de faire

```
minikube service mon-front-service --url

minikube tunnel # pour l'accès extérieur
```

- https://minikube.sigs.k8s.io/docs/handbook/accessing/

Travaux pratiques

- Objectif : exposer une application web à l'extérieur.
- #services / deployment-and-service



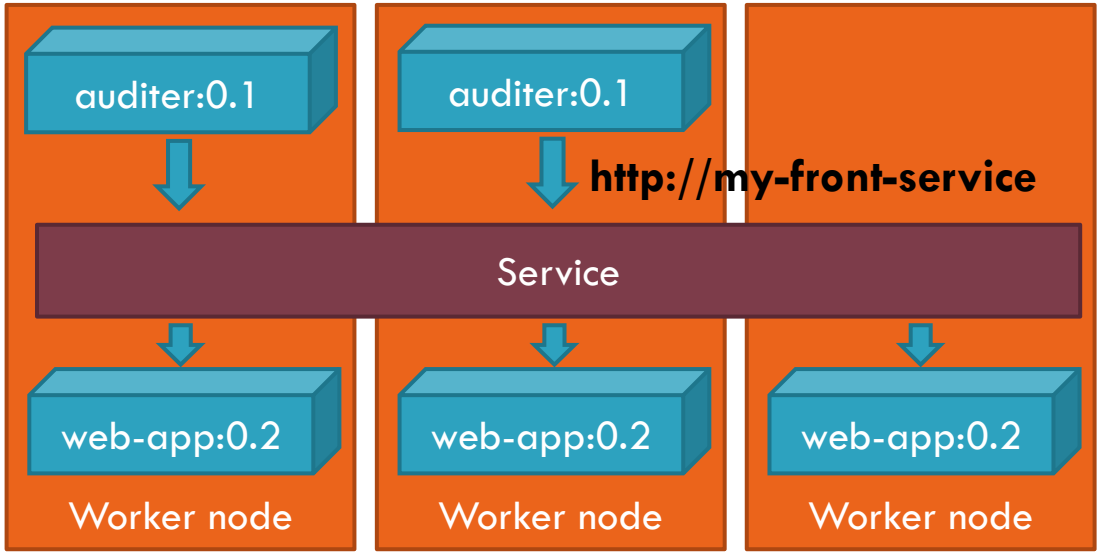
En cas de souci

- https://kubernetes.io/docs/tasks/debug-application-cluster/debug-service

Démonstration

- LoadBalancer sur cloud public

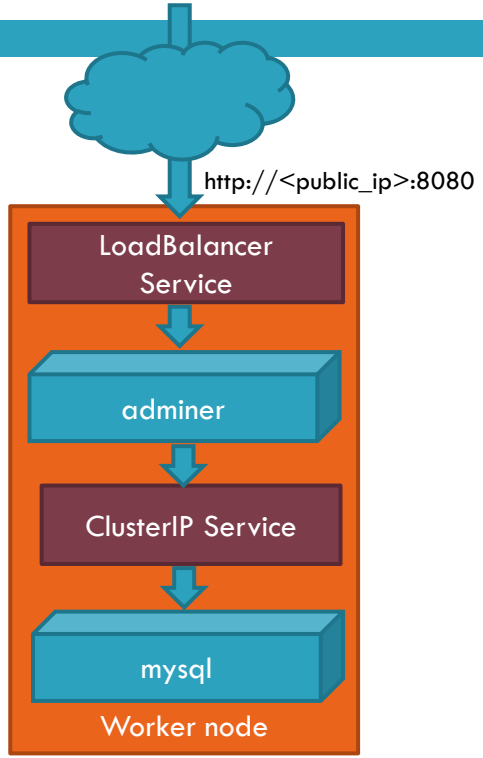
Interne au cluster




```
apiVersion: v1
kind: Service
metadata:
  name: my-front-service
spec:
  type: ClusterIP
  ports:
  - port: 80
selector:
    app: web-app
```

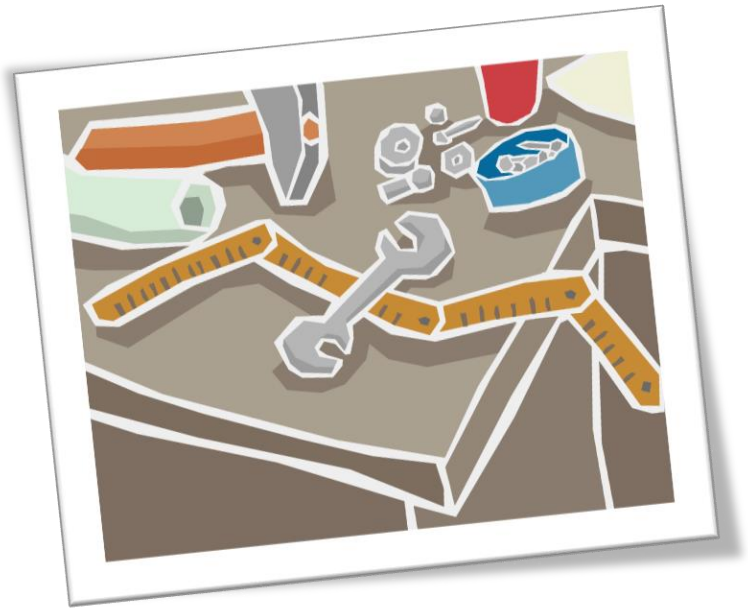
Démonstration

- Déploiement MySQL et Adminer



Travaux pratiques

- Objectif : exposer une API de manière interne au cluster.
- #services / back-and-front

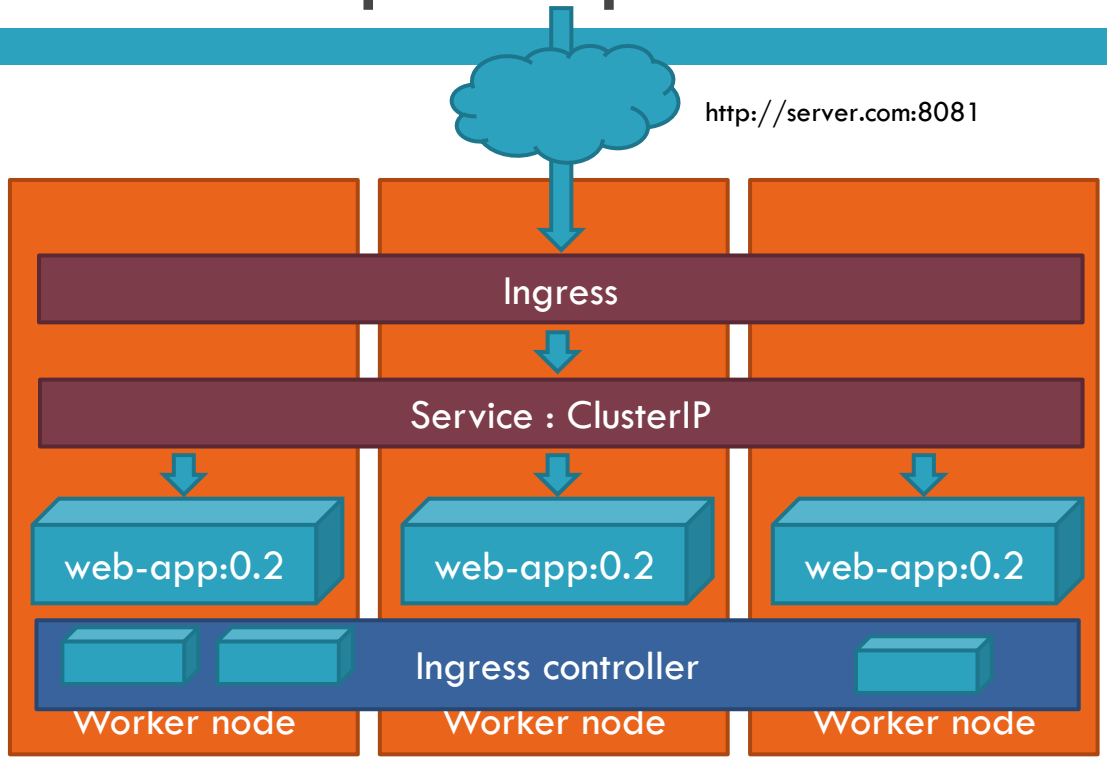


Ingress

Ingress, mais pourquoi ?

- Equivalent au LoadBalancer mais
  - le load balancing est fait par des pods **dans** le cluster
  - paramétrage dans K8s : objets Ingress
- Router vers des services selon la DNS ou le chemin d'arrivée:
  - http://front/store vers store:1.0
  - http://front/old/store vers store:0.8
- TLS (HTTPS)

Schéma de principe



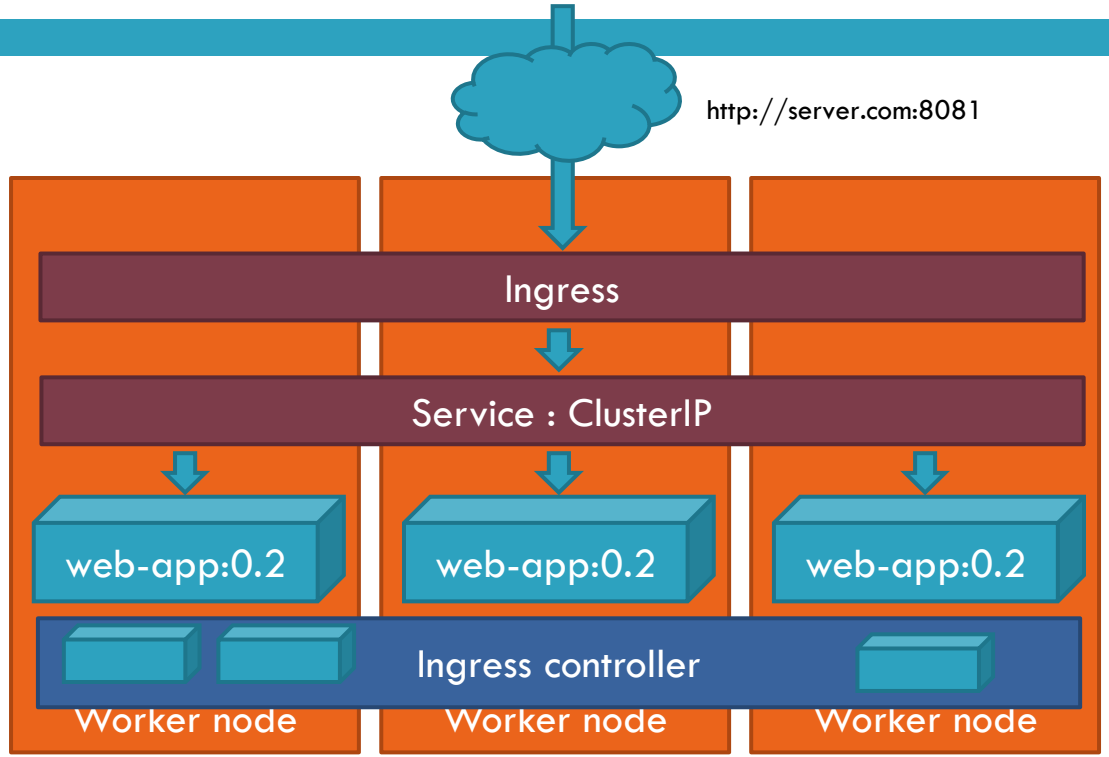
Exemple

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontandback-ingress
spec:
  ingressClassName: nginx
  rules:
    - host: server.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web-app
                port:
                  number: 80
```

- Il faut en plus avoir un contrôleur ingress dans le cluster :
  - celui du prestataire
  - nginx-ingress
  - contour
  - traefik

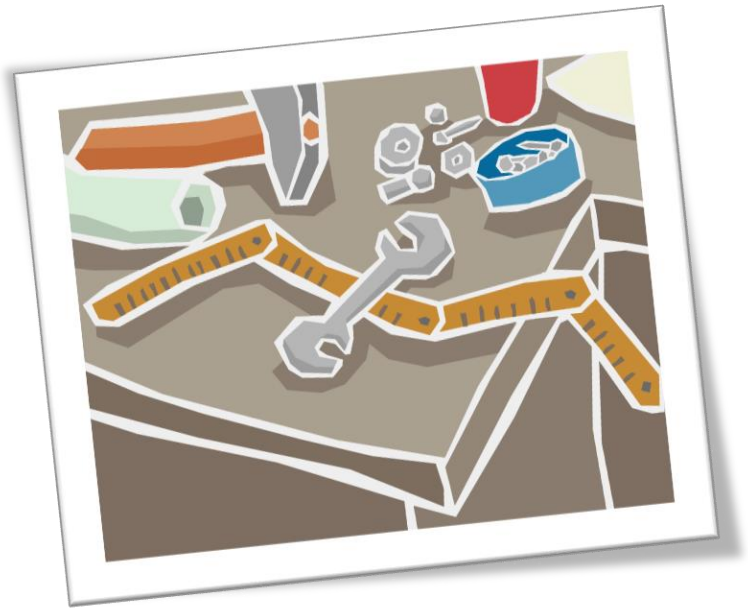
En résumé

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontandback-ingress
spec:
  ingressClassName: nginx
  rules:
    - host: server.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web-app
                port:
                  number: 80
```

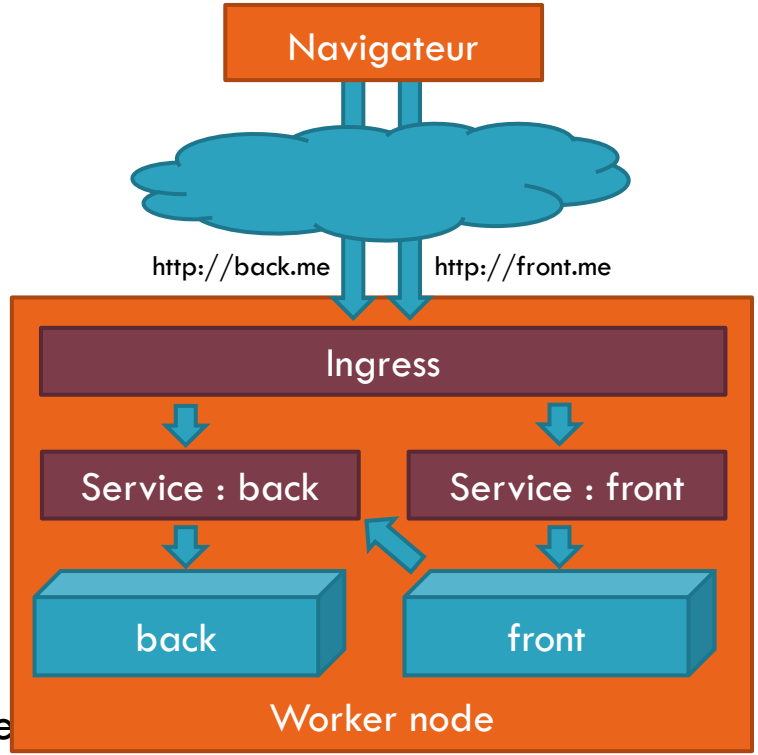


Travaux pratiques

- (optionnel) Objectif : exposer le front et le back chacun sur le port par défaut (80) et sur sa propre DNS.
- #services / ingress



Architecture du TP



VOLUMES

#storage

Problématique

Fichiers du conteneur

- Ils sont éphémères

Besoin de persistance

- Temporaire

  ou durable

- Interne à
  - un pod,
  - un node

  ou externe au cluster

Conserver le modèle d'abstraction

- Les ressources des pods sont fournies par des worker nodes interchangeables
- Qui fournit la persistence des pods ?
- Quels critères pour demander du stockage ?

Deux solutions

- Volume: durée liée au pod
- PersistentVolumeClaim: durée indépendante des pods

Jetable: le volume

Volume

- Propre à un pod
- Meurt en même temps que le pod
  - Mais pratiquement tous persistants
- Visible par les containers du pod

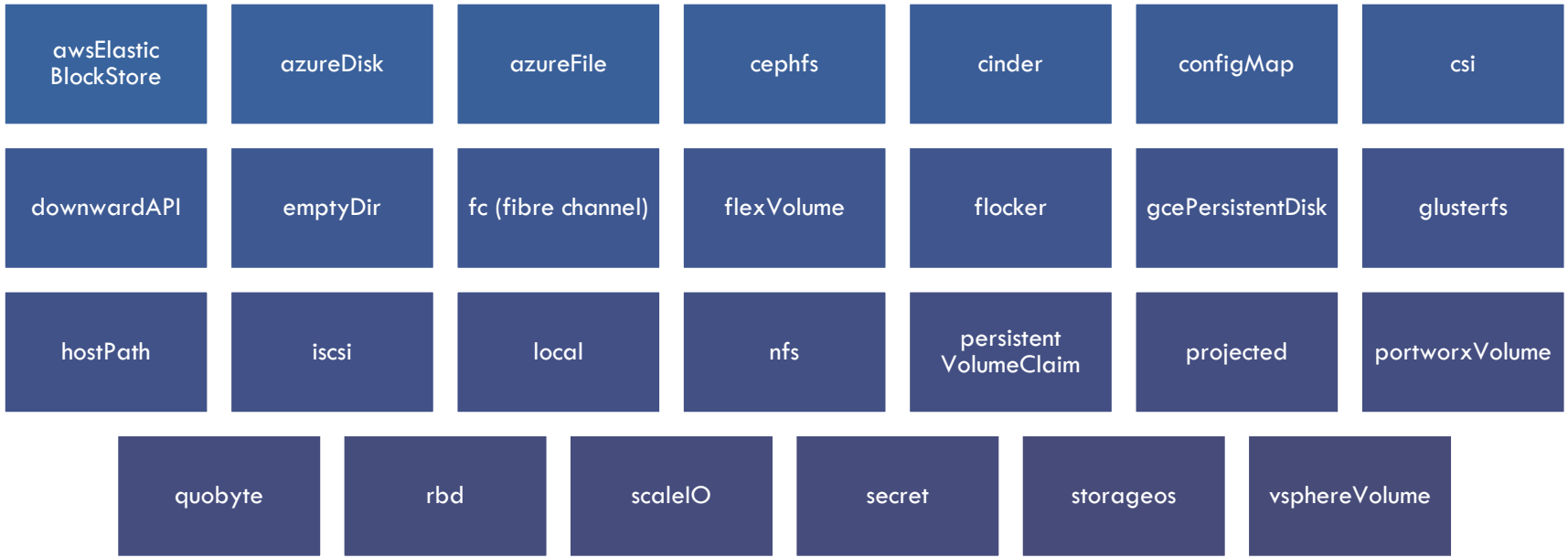
Déclaration

- Déclaré sur le Deployment
- Monté sur les containers
- Un conteneur voit son système de fichiers plus les volumes
- Ici le type est emptyDir

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      volumes:
        - name: persistent-data
          emptyDir: {}
      containers:
      - name: front
        image: my-image:1.0
        volumeMounts:
          - mountPath: /external
            name: persistent-data
```

Types de volumes

https://kubernetes.io/docs/concepts/storage/volumes/#types-of-volumes



Déclaration

- Selon les types, des options supplémentaires sont à passer

```
apiVersion: apps/v1beta1
kind: Deployment
spec:
  template:
    spec:
      volumes:
        - name: persistent-data
          azureDisk:
            diskName: test.vhd
            diskURI: https://...microsoft.net/vhds/test.vhd
      containers:
      - name: front
        image: my-image:1.0
        volumeMounts:
          - mountPath: /external
            name: persistent-data
```

Démonstration

- #volumes / hostpath

emptyDir

- Ecrit sur le node ou en RAM (medium: "Memory")
- Meurt avec le pod (pas si crash du container)

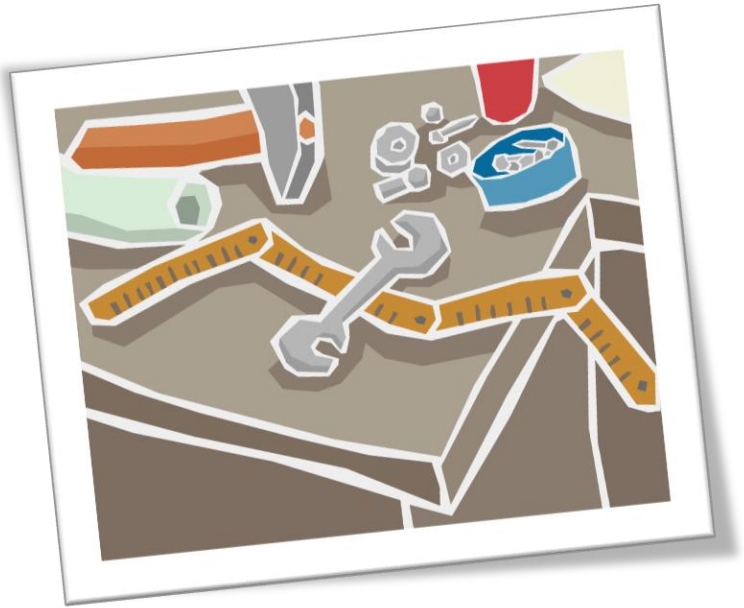
```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      volumes:
        - name: persistent-data
          emptyDir: {}
      containers:
      - name: front
        image: my-image:1.0
        volumeMounts:
          - mountPath: /external
            name: persistent-data
```

persistentVolumeClaim

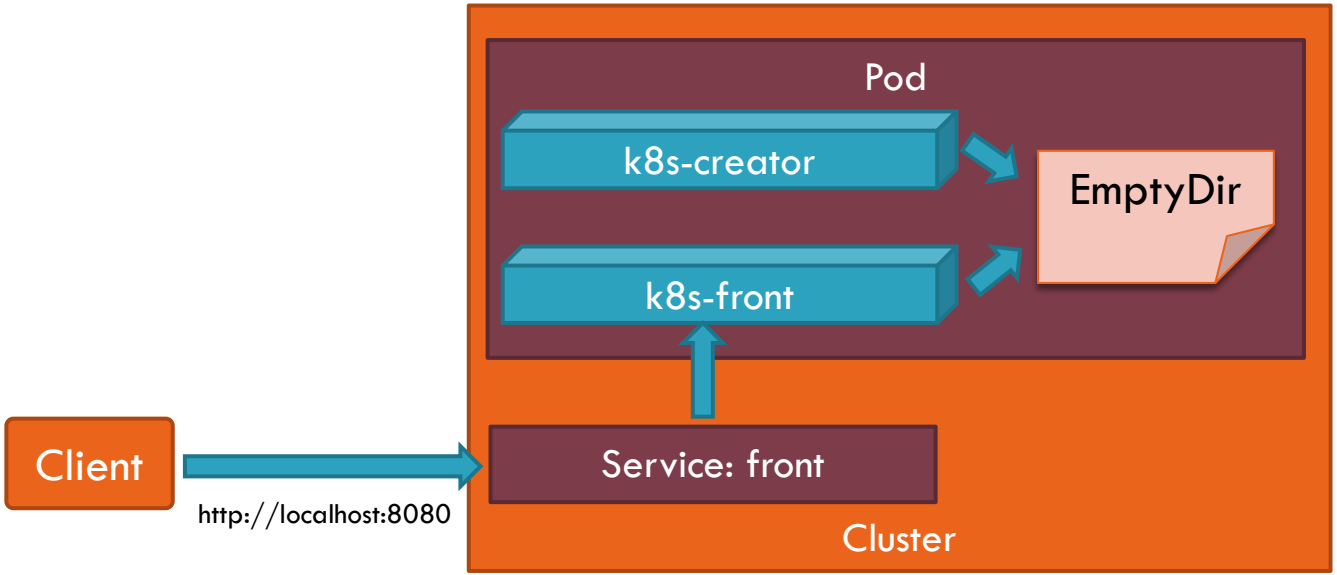
- Permet de faire une demande indirecte de stockage persistant

Travaux pratiques

- Objectif : créer et utiliser un volume de type emptydir afin de partager des fichiers entre conteneurs dans un pod.
- #storage / emptydirs

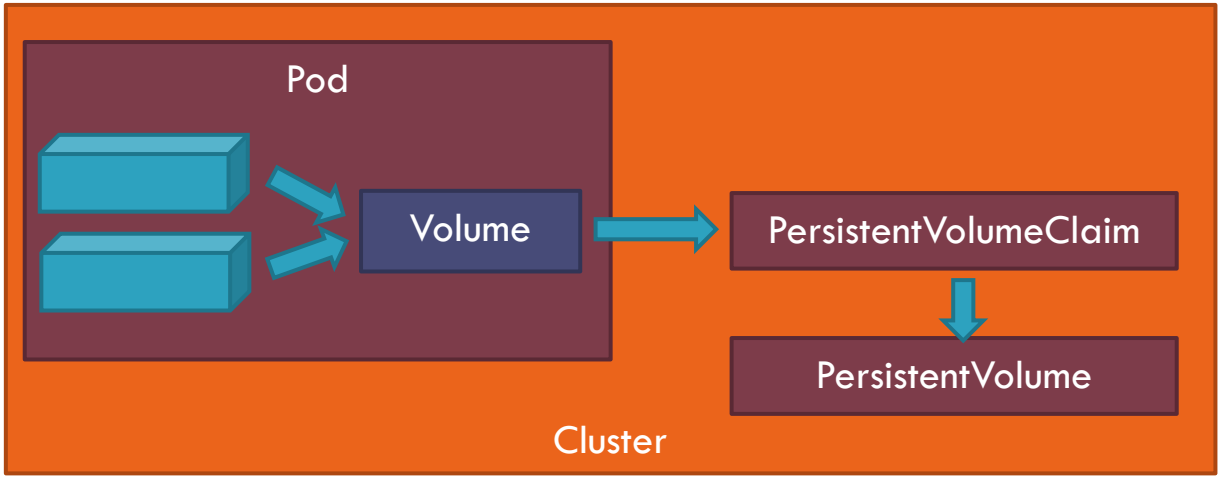


Architecture du TP



Persistant: PersistentVolumeClaim

Schéma de principe



Principe

- PersistentVolumes: ressources attribuées
  - par l'administrateur

    - ou -
  - Dynamiquement sur demande des PersistentVolumeClaims
- Virtualisation des ressources disque
  - Les Pods consomment des Nodes
  - Les PersistentVolumeClaims consomment des PersistentVolumes

Demande

- Le pod consomme un PVC comme si c'était un volume
- La PVC est un objet créé séparément

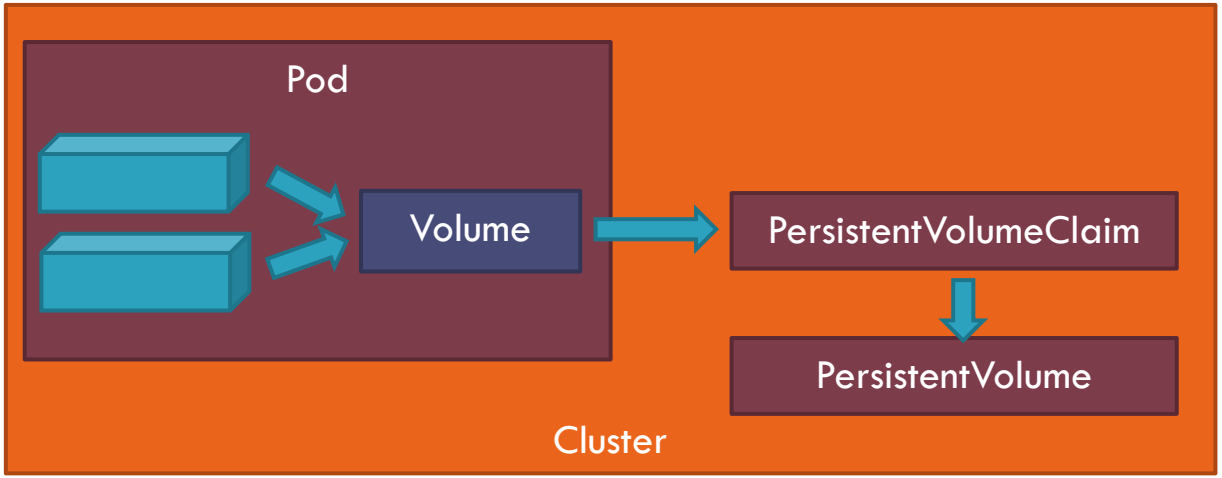
```
apiVersion: apps/v1beta1
kind: Deployment
spec:
  template:
    spec:
      volumes:
       - name: persistent-data
         persistentVolumeClaim:
           claimName: claiming-disk
      containers:
      - name: front
        image: my-image:1.0
        volumeMounts:
          - mountPath: /external
            name: persistent-data
```

Déclaration d'un PVC

- Demande d'un morceau de PersistentVolume
- Cycle de vie indépendant des pods

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: claiming-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Rappel: schéma de principe



Binding PVC-PV

- Binding statique: attribution d'un PersistentVolume préalablement créé par l'administrateur
- Binding dynamique: spécifier la StorageClass
  - L'admin a créé un StorageClass avant
  - Si on spécifie le volumeName, on peut en plus choisir un PV en particulier

Binding PVC-PV dynamique

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: foo-pvc
  namespace: foo
spec:
  storageClassName: "slow"
  resources:
    requests:
      storage: 1Gi
```

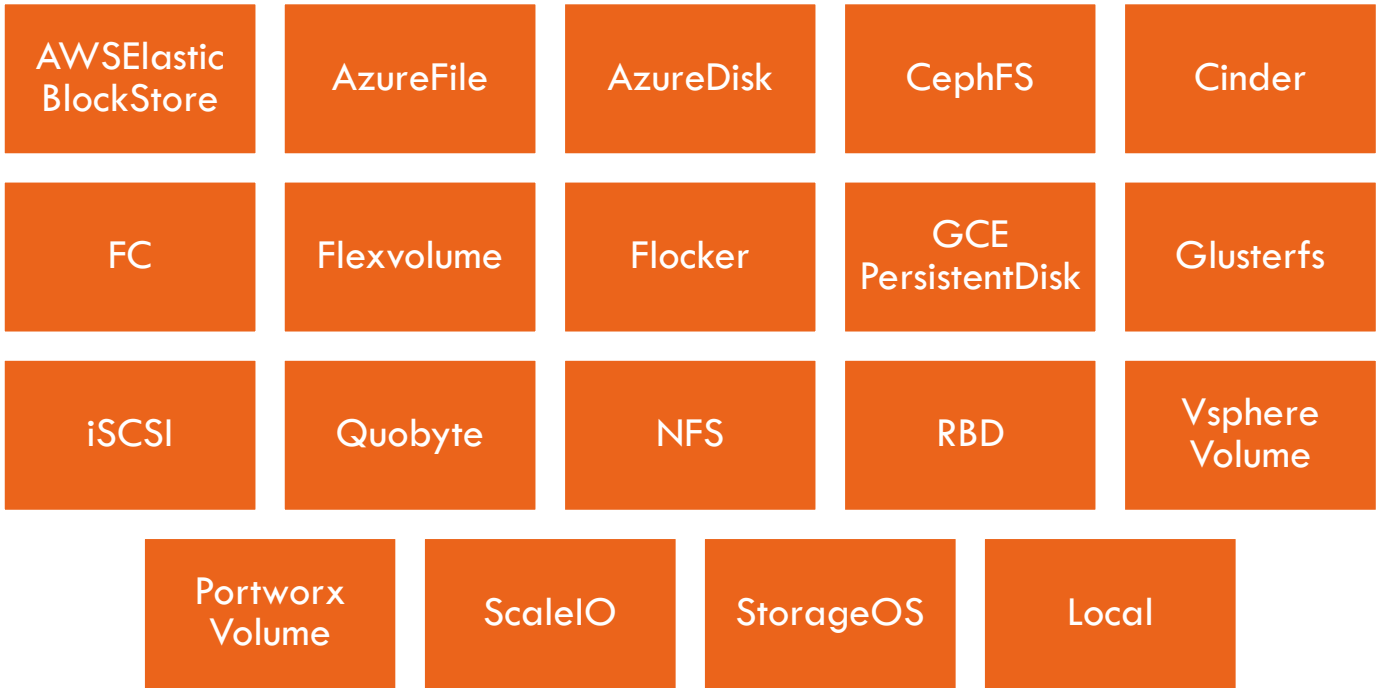
```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

Binding PVC-PV statique

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: foo-pvc
  namespace: foo
spec:
  storageClassName: "" # vide
  volumeName: pv0003
```

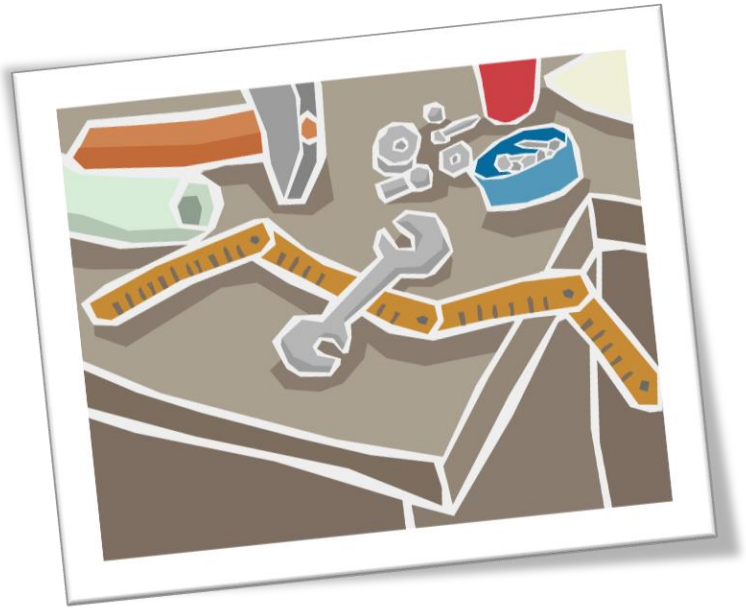
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Storage classes



Travaux pratiques

- Objectif : utiliser un PersistentVolume dynamique pour conserver les fichiers lorsque le pod meurt.
- #storage / persistentvolume



Rappel

```
apiVersion: apps/v1beta1
kind: Deployment
spec:
  template:
    spec:
      volumes:
        - name: persistent-data
          persistentVolumeClaim:
            claimName: claiming-disk
      containers:
      - name: front
        image: my-image:1.0
        volumeMounts:
          - mountPath: /external
            name: persistent-data
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: claiming-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

PARAMÉTRAGE

#configuration

ConfigMap

Principe

- ConfigMap permet de stocker des clés / valeurs utilisées par plusieurs pods
- Un ConfigMap peut être exposé au pod en tant que:
  - Variables d'environnement nommées
  - Variables d'environnement pour toutes les entrées
  - Fichiers

Création du ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-values
data:
  key1: "value1"
  key2: "value2"
```

Montage en tant que volume

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-values
data:
  key1: "value1"
  key2: "value2"
```

```
apiVersion: apps/v1beta1
kind: Deployment
spec:
  template:
    spec:
      volumes:
        - name: config-volume
          configMap:
            name: configmap-values
      containers:
      - name: front
        image: learnbook/k8s-front:0.4
        volumeMounts:
          - mountPath: /config
            name: config-volume
```

Variables d'environnement

- Toutes les clés:

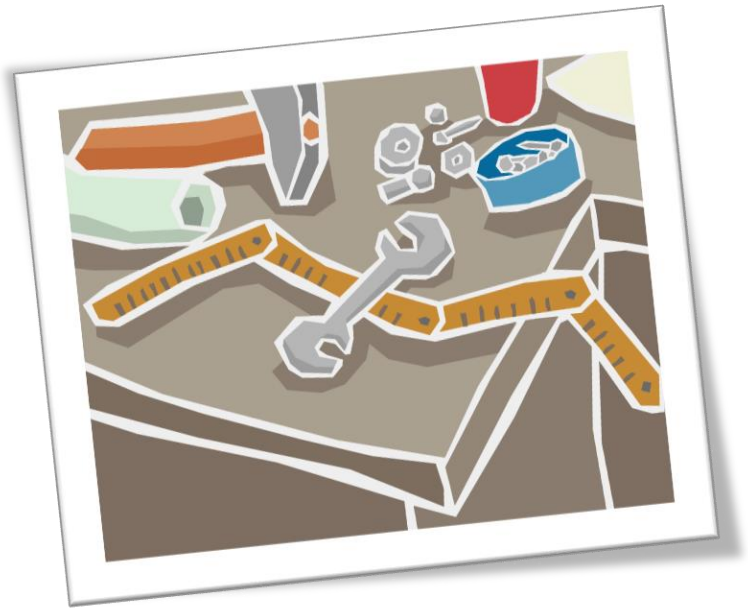
```
apiVersion: apps/v1beta1
kind: Deployment
spec:
  template:
    spec:
      containers:
      - name: front
        image: learnbook/k8s-front:0.4
        envFrom:
          - configMapRef:
              name: configmap-values
```

- Sélection de clés :

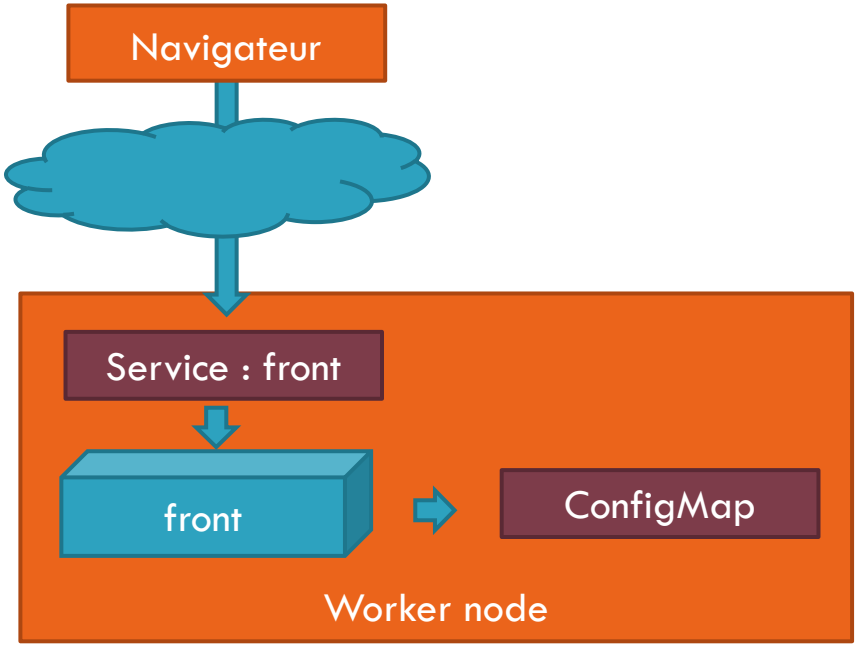
```
apiVersion: apps/v1beta1
kind: Deployment
spec:
  template:
    spec:
      containers:
      - name: front
        image: learnbook/k8s-front:0.4
        env:
        - name: VAR_NAME
          valueFrom:
            configMapKeyRef:
              name: configmap-values
              key: key2
```

Travaux pratiques

- Objectif : créer et utiliser un objet ConfigMap de différentes manières.
- #configuration / configmap



Aide




```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-values
data:
  key1: "value1"
  key2: "value2"
```

Secret

Principe

- Comme les ConfigMaps
- Appliquer des restrictions avec RBAC

Création du Secret

- Clés/valeurs passées en paramètres

```
kubectl create secret generic my-secret --from-literal=key1=supersecret --from-literal=key2=topsecret
```

- Clés : noms de fichiers, valeurs : contenu des fichiers

```
kubectl create secret generic my-secret --from-file=path/to/bar
```

- Certificat

```
kubectl create secret tls tls-secret --cert=path/to/tls.cert --key=path/to/tls.key
```

Montage en tant que volume

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-values
stringData:
  key1: "secret value1"
  key2: "secret value2"
```

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      volumes:
        - name: config-volume
          secret:
            secretName: secret-values
      containers:
      - name: front
        image: learnbook/k8s-front:0.4
        volumeMounts:
          - mountPath: /config
            name: config-volume
```

En tant que variables d'environnement

- Toutes les clés:

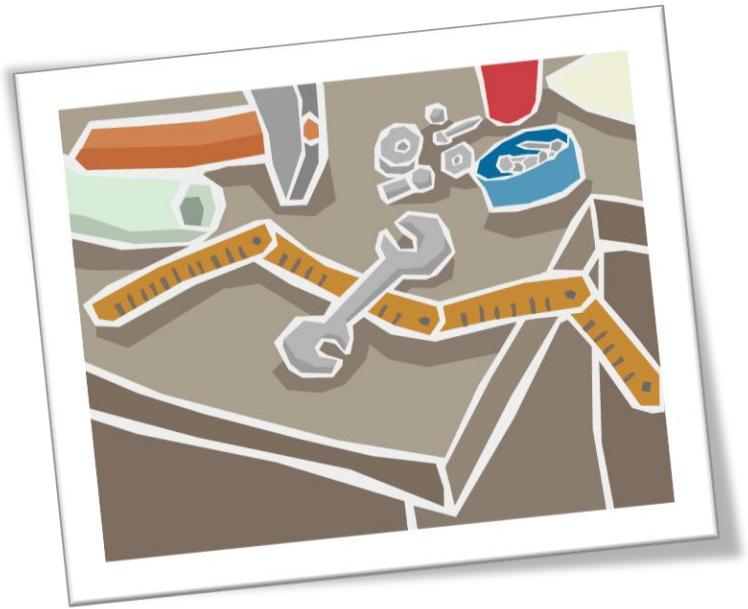
```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
      - name: front
        image: learnbook/k8s-front:0.4
        envFrom:
          - secretRef:
              name: secret-values
```

- Sélection de clés :

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      containers:
      - name: front
        image: learnbook/k8s-front:0.4
        env:
        - name: VAR_NAME
          valueFrom:
            secretKeyRef:
              name: secret-values
              key: key2
```

Travaux pratiques

- Objectif : créer et utiliser un objet Secret pour un paramétrage confidentiel.
- #configuration / secret

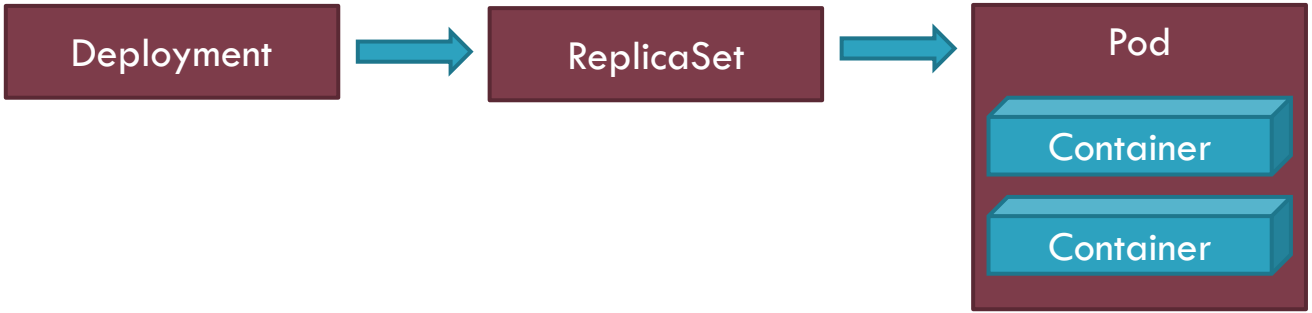


MONTÉE EN CHARGE ET MISE À JOUR

#updating

Montée en charge horizontale

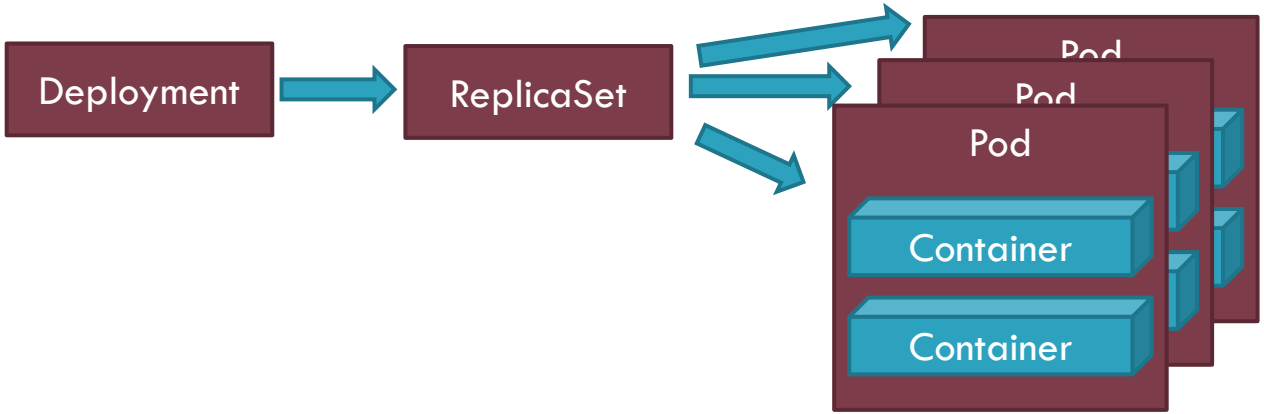
Rappel sur les deployments



Rappel sur les deployments

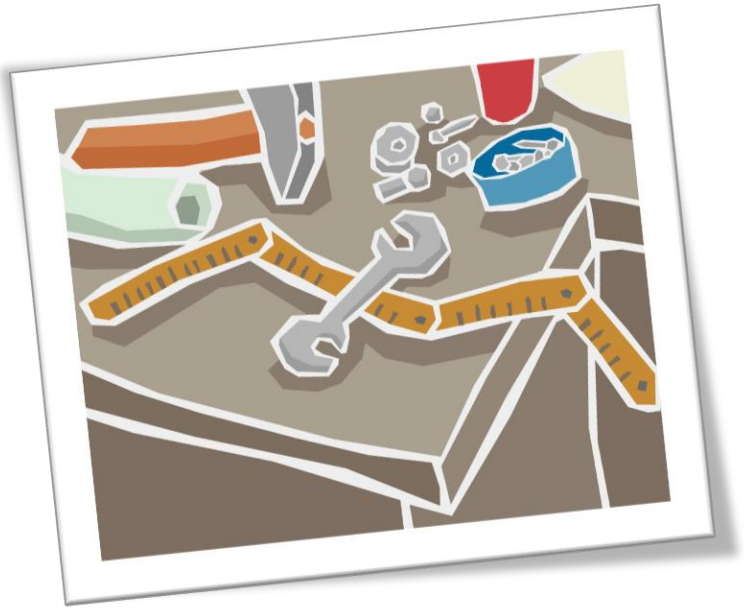
- ReplicaSet crée toutes les replicas

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: mon-front
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: mon-front
        image: mon-wordpress
```



Travaux pratiques

- Objectif : observer le comportement du ReplicaSet lorsque plusieurs replicas sont demandées.
- #updating / create-instances



Montée en charge automatique

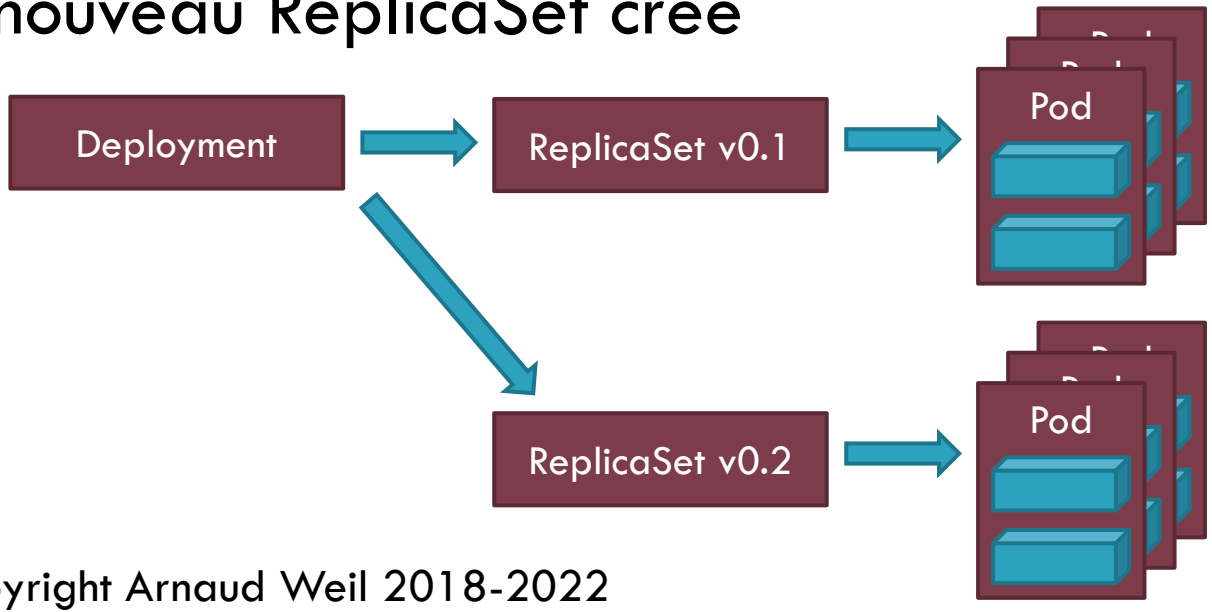
- Moyenne du pourcentage de la resource request utilisé par tous les pods de ce Deployment

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: captureorder
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: front
  minReplicas: 4
  maxReplicas: 10
  targetCPUUtilizationPercentage: 70
```

Stratégies de mise à jour

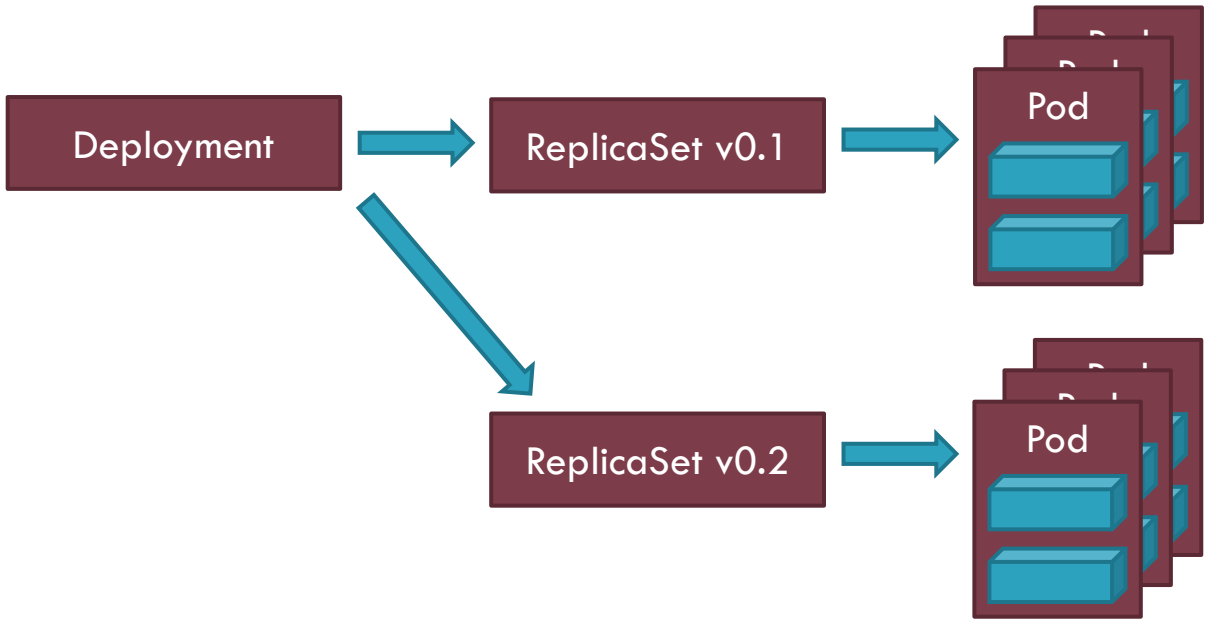
Mise à jour

- Lors d'une modification de la configuration du Pod, nouveau ReplicaSet créé



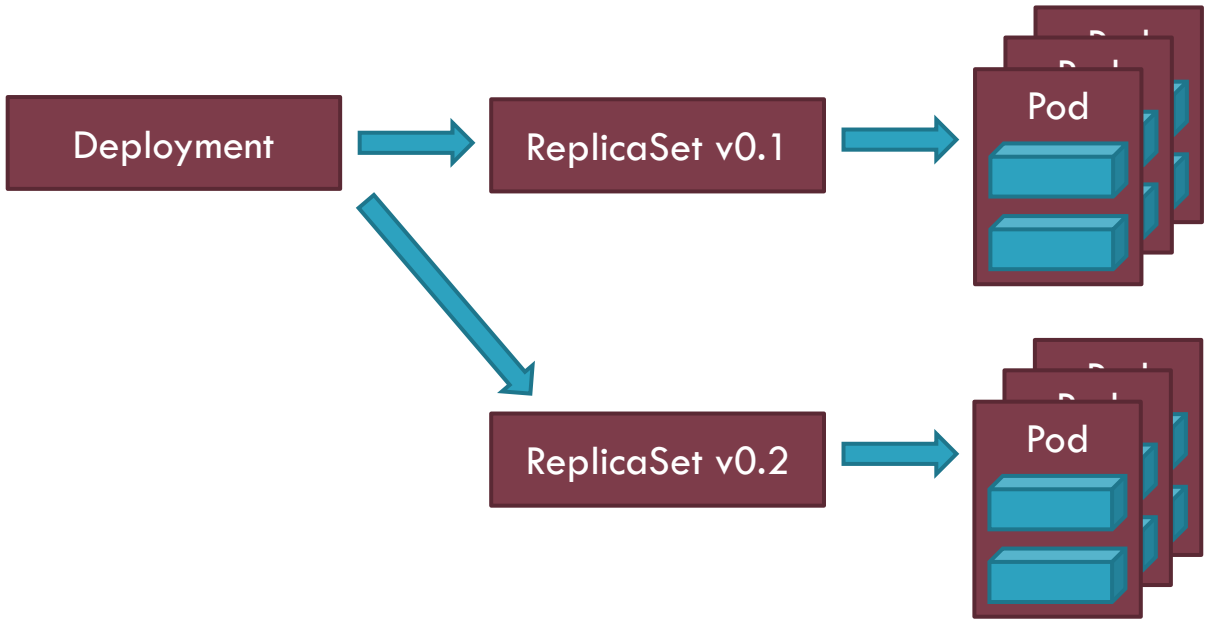
Recreate

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-front
spec:
  strategy:
    type: Recreate
  replicas: 3
  template:
    spec:
      containers:
      - name: mon-front
        image: mon-wordpress
```



Rolling update

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-front
spec:
  strategy:
    type: RollingUpdate
  replicas: 3
  template:
    spec:
      containers:
      - name: mon-front
        image: mon-wordpress
```



Paramétrage du rolling update

- maxSurge, maxUnavailable
  - Pourcentage
  - Entier
- Par défaut: 25%

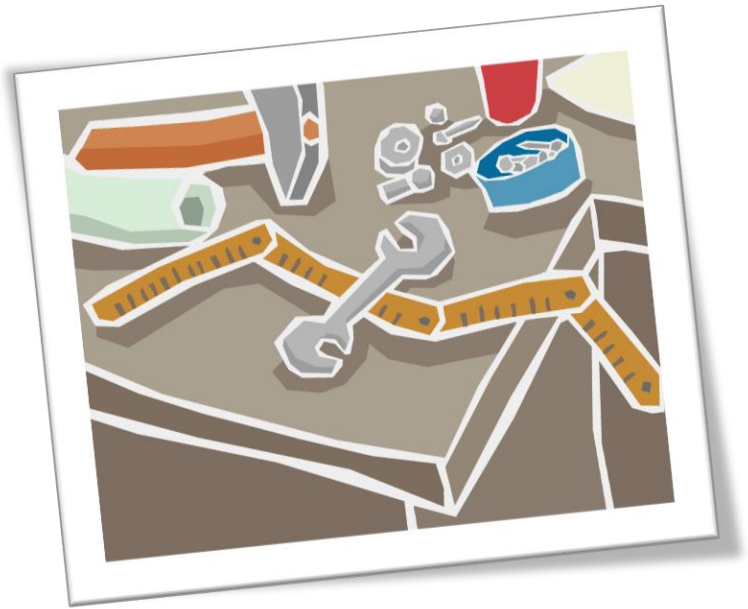
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mon-front
spec:
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
  replicas: 3
  template:
    spec:
      containers:
      - name: mon-front
        image: mon-wordpress
```

Autres stratégies

- Types
  - Blue / green
  - Canary
  - Rainbow
- Méthode
  - Routage via les services avec selecteurs

Travaux pratiques

- Objectif : procéder à des mises à jour de type rolling update en vérifiant le comportement du cluster.
- #updating / update



PARTAGER UN CLUSTER

#sharing

Namespaces

Principe

- Offre un certain degré de séparation:
  - Noms scopés
  - DNS utilisant le namespace
  - Autorisations liées aux namespaces (RBAC)
  - Limitation des ressources par namespace
- Utile pour séparer des équipes

Namespaces par défaut

- default: objets sans namespace
- kube-system: objets de Kubernetes

Création d'un namespace

- C'est un objet Kubernetes

```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
```

```
kubectl apply -f ns.yaml
```

Commandes

- Prennent --namespace

```
kubectl apply -f dep.yaml --namespace team1
```

```
kubectl get pods --namespace team1
```

- Possibilité de passer --all-namespaces pour des requêtes get

Pour plus de praticité

- On peut assigner un namespace par défaut au contexte

```
kubectl config set-context --current --namespace=team1
```

Noms scopés

- Unicité au sein d'un namespace

DNS

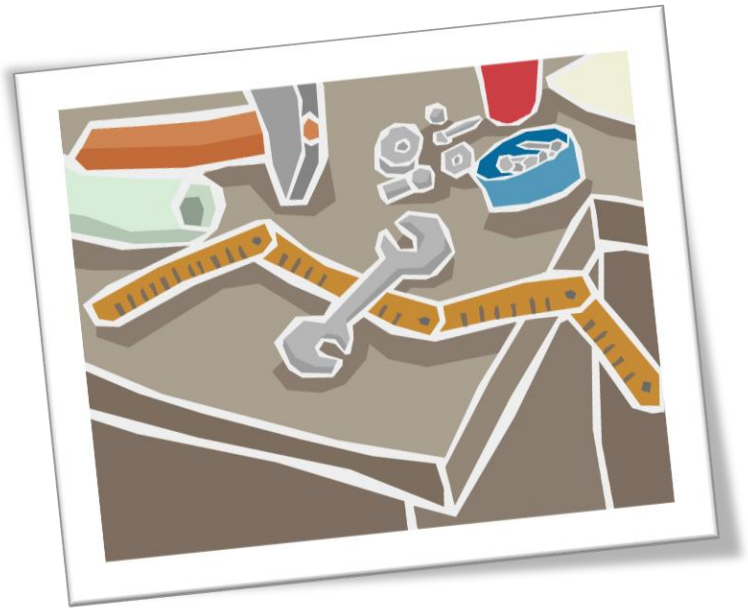
- CoreDNS (add-on avant la version 1.13) crée des entrées DNS pour chaque service
- Le service front du namespace team1 a comme DNS:
  - front.team1
- Accès depuis le même namespace:
  - front

Ce qui n'est pas dans un namespace

- Namespaces
- Nodes
- PersistentVolumes

Travaux pratiques

- Objectif : créer deux déploiements parallèles de l'application front
- #sharing / namespaces



Limitation des ressources

- Objet ResourceQuota permet de limiter le total sur tous les pods d'un namespace pour:
  - limits.cpu
  - limits.memory
  - requests.cpu
  - requests.memory

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
    requests.nvidia.com/gpu: 4
```

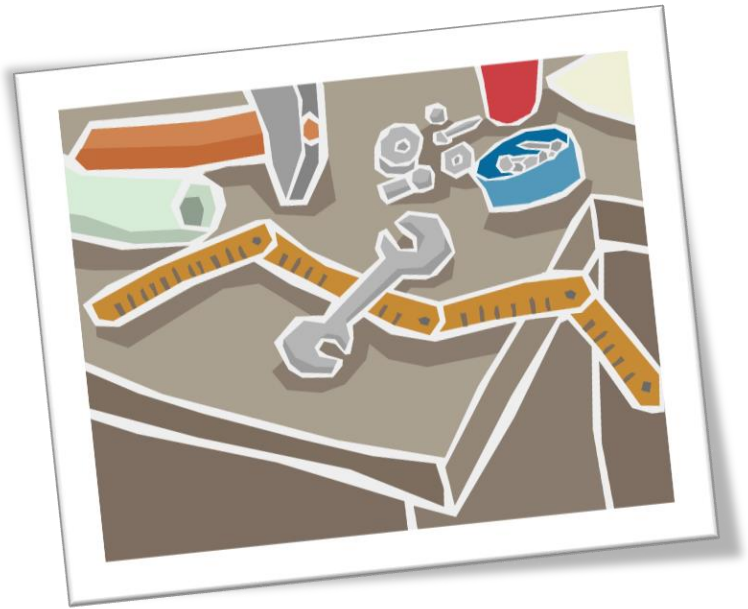
Application de requests/limits par défaut

- Objet LimitRange permet de définir les requests/limits par défaut dans un namespace

```
apiVersion: v1
kind: LimitRange
metadata:
  name: memory-limit
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256Mi
    type: Container
```

Travaux pratiques

- Objectif : limiter les ressources utilisables dans un namespace
- #sharing / quotas



Références

- Sécurisation

  https://kubernetes.io/docs/tasks/administer-cluster/securing-a-cluster/

RBAC

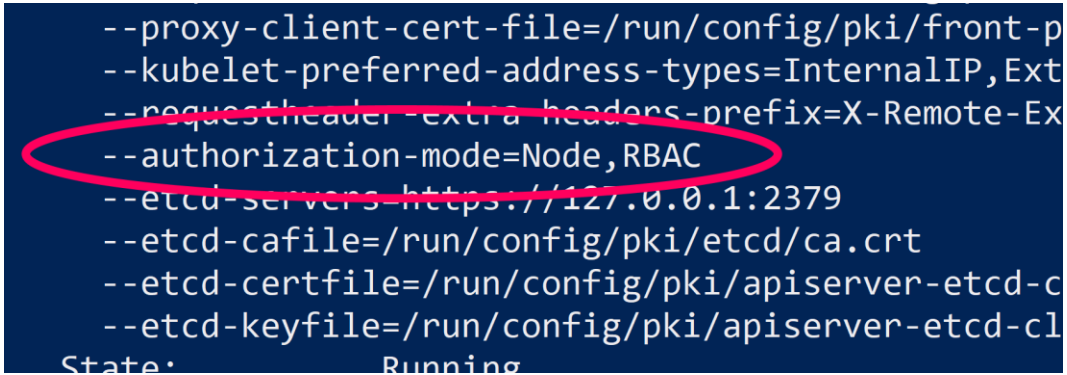
Pourquoi ?

- Limiter les accès à l'API Kubernetes
- Permissions sur l'API, pour les Pods et kubectl
- C'est un des modes d'autorisation (Node, ABAC, RBAC, WebHook)

Vérifier son activation

- C'est un module d'autorisation parmi d'autres

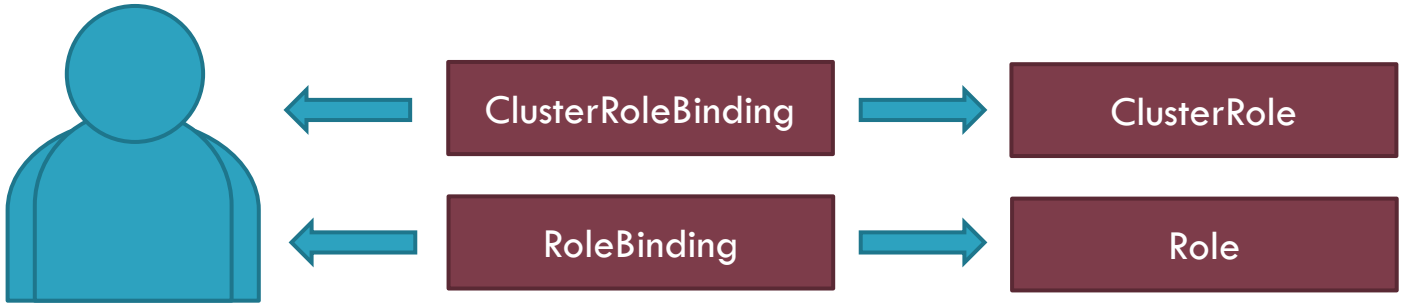
kubectl describe pod -n kube-system -l component=kube-apiserver



Roles

- Déterminent un ensemble de permissions
- ClusterRoles: tout le cluster
- Roles: limités à un namespace

Bindings



User / ServiceAccount

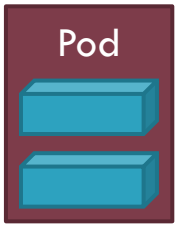
- User
  - Utilisateur externe
  - Plusieurs modules d'identification possibles, seul le nom est retenu
- ServiceAccount
  - Pour les Pods

ServiceAccount

- Pour chaque Pod
- Visible dans spec.serviceAccountName
- Assignation d'un ServiceAccount :

```
…
spec:
  containers:
  …
  serviceAccountName: default
  …
```

Relation complète



ServiceAccount

RoleBinding

Role

```
…
spec:
  containers:
  …
  serviceAccount: toto
  …
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: toto
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: list-secrets_gotty
  namespace: default
roleRef:
  kind: Role
  name: list-secrets
  apiGroup: rbac.authorization.k8s.io
subjects:
  - kind: ServiceAccount
    name: toto
    namespace: default
```

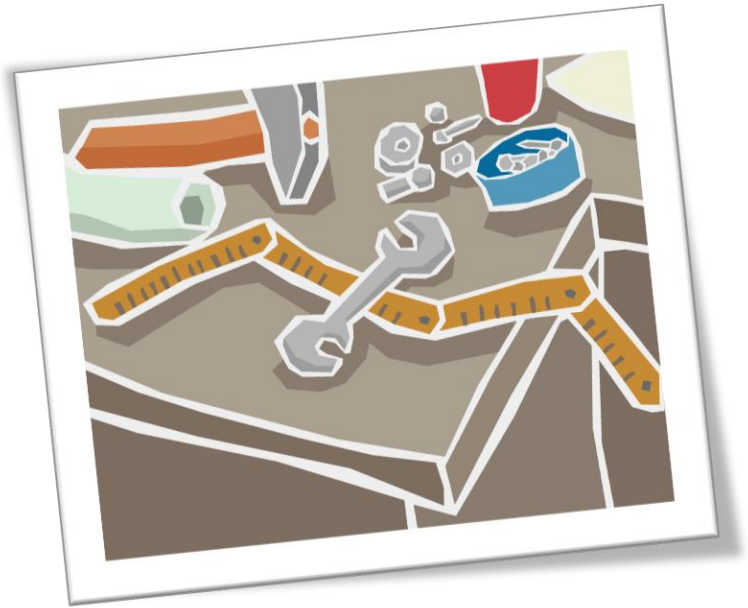
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: list-secrets
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["list"]
```

Démonstration

- Affichage détaillé de quelques ClusterRoles

Travaux pratiques

- Objectif : accéder à l'API Kubernetes depuis un Pod
- #sharing / rbac



HELM

#helm

Utilisation de Helm

Helm ?

- « Gestionnaire de package de Kubernetes »
- En pratique: fichiers YAML variabilisés et packagés

- Intérêt
  - Ajout de ressources au cluster comme avec kubectl apply mais paramétrable

En pratique

- Les charts sont dans des repositories (exemple: https://artifacthub.io/)
- Installation d'un chart:

```
helm install <name> --set version=1.0,key=value,… <chart>
```

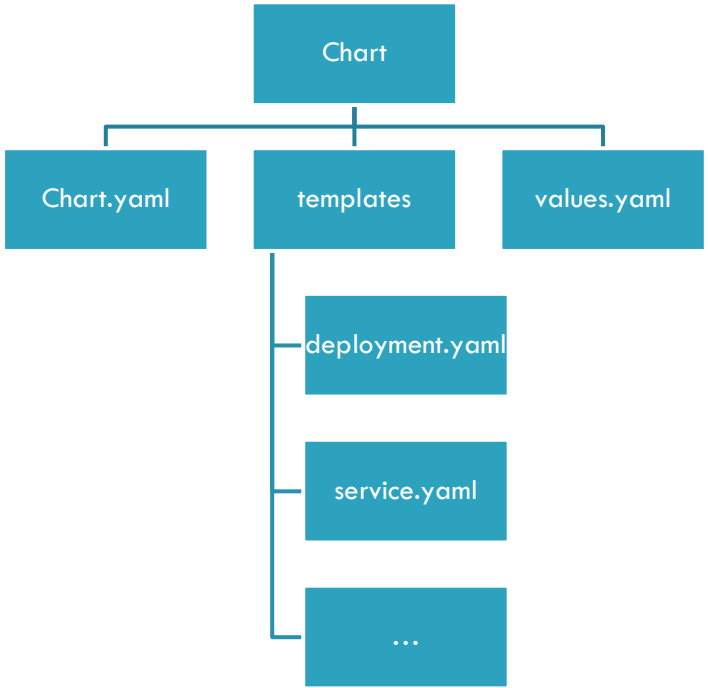
Démonstration

- Déploiement de Redis
- Déploiement de MySQL

Création de charts

Un chart c'est

- Des fichiers YAML d'objets Kubernetes
- Un fichier values.yaml (valeurs par défaut)
- Un fichier NOTES.txt
- Le tout dans un fichier .tgz



Syntaxe

- Syntaxe YAML standard
- Utilisation de templates Go

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: back-{{ .Values.config }}
```

Valeurs disponibles

```
version: 0.2
type: my-value
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-{{ .Release.Name }}
spec:
  template:
    spec:
      containers:
      - name: back-{{ .Release.Name }}
        image: learnbook/k8s-back:{{ .Values.version }}
```

Générer le chart

- Création initiale des fichiers

```
helm create <name>
```

- Vérification puis mise en tgz

```
helm lint <name>
helm package <name> --debug
```

- Vérification exécution

```
helm install <release> --dry-run --debug <name>-0.1.0.tgz
```

Pour installer un chart

- Au choix:
  - Le placer sur un repo Helm (serveur HTTP de fichiers)
  - Pointer directement un fichier tgz
- Puis comme tout autre chart:

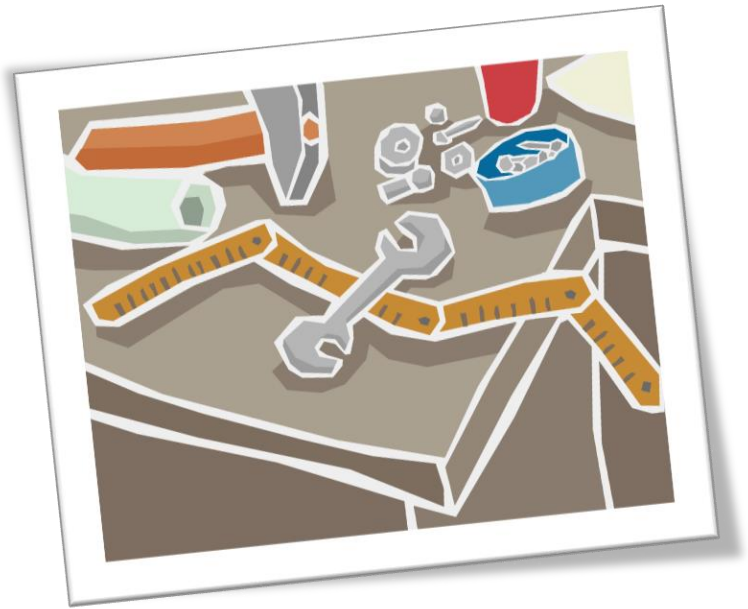
```
helm install <name> --set key=value,… <chart>-0.1.0.tgz
```

Démonstration

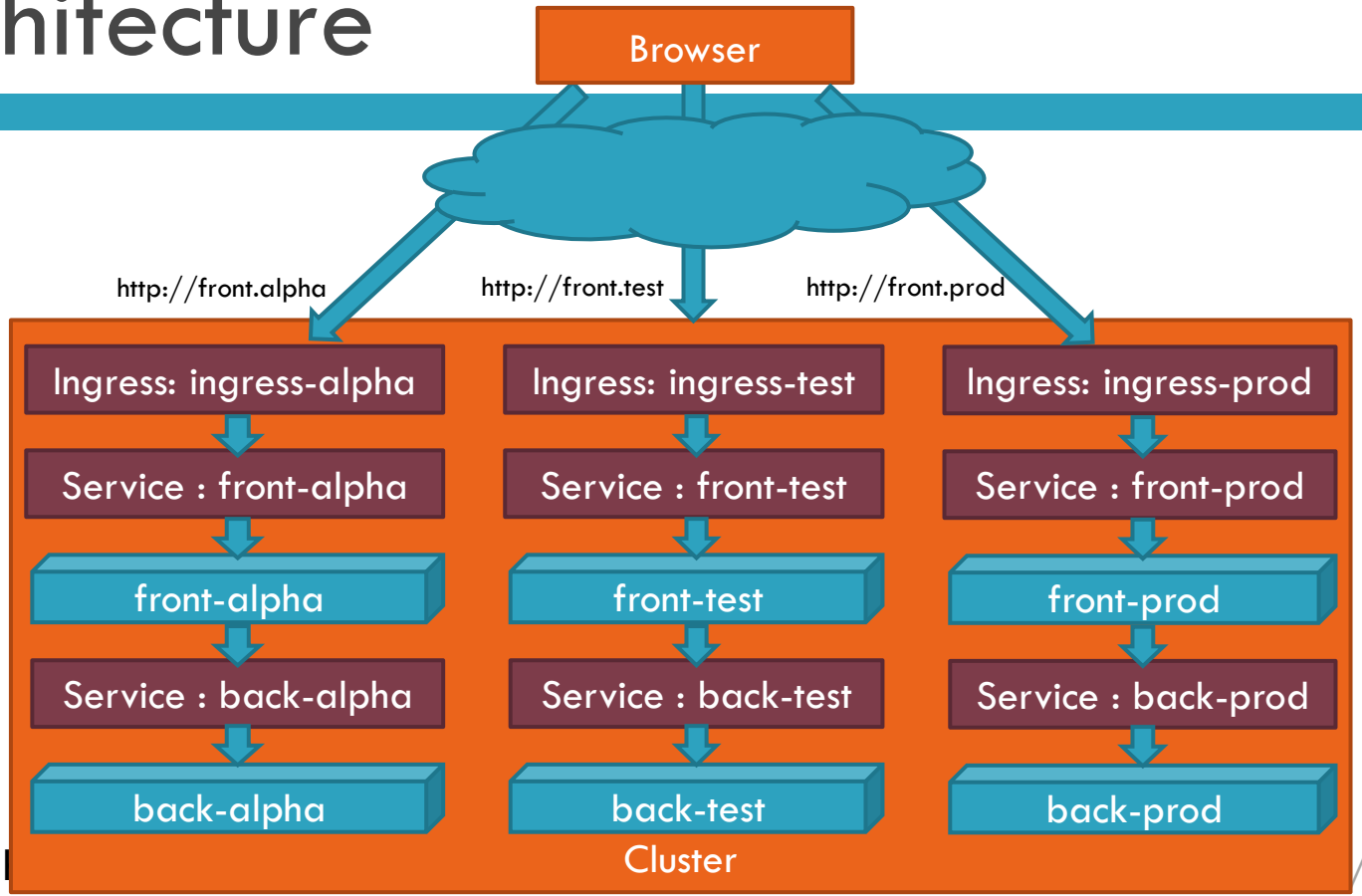
- Création de chart et déploiement

Travaux pratiques

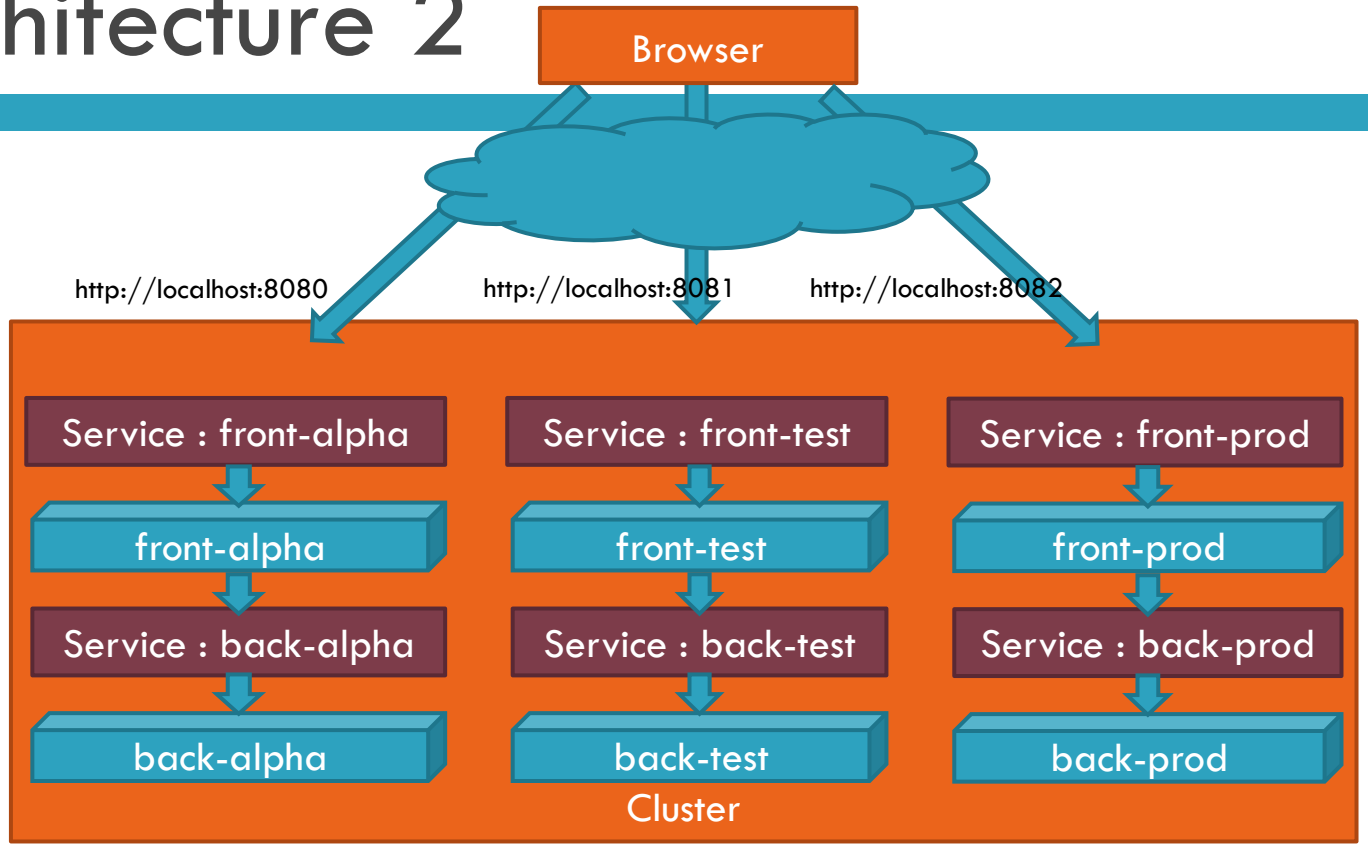
- Objectif: transformer la stack front-back en chart Helm, puis en exécuter trois instances.
- #helm / environments



Architecture



Architecture 2



BIBLIOGRAPHIE

Le Plan Copenhague

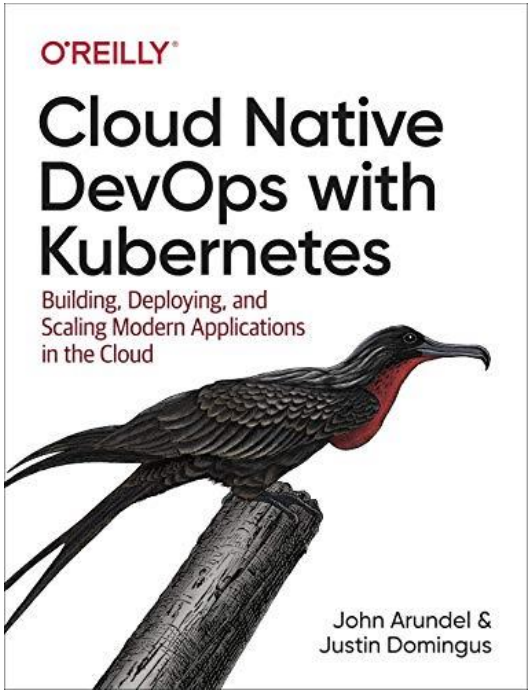
- Retour d'expérience

- https://leanpub.com/6cloud

Cloud Native DevOps

- Apprentissage et recommandations

- John Arundel, Justin Domingus

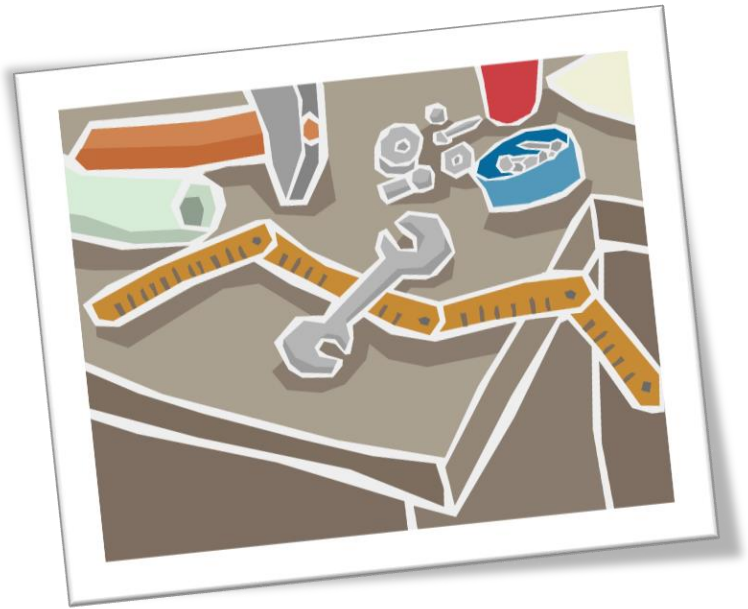


CLUSTER PAAS

#cloud

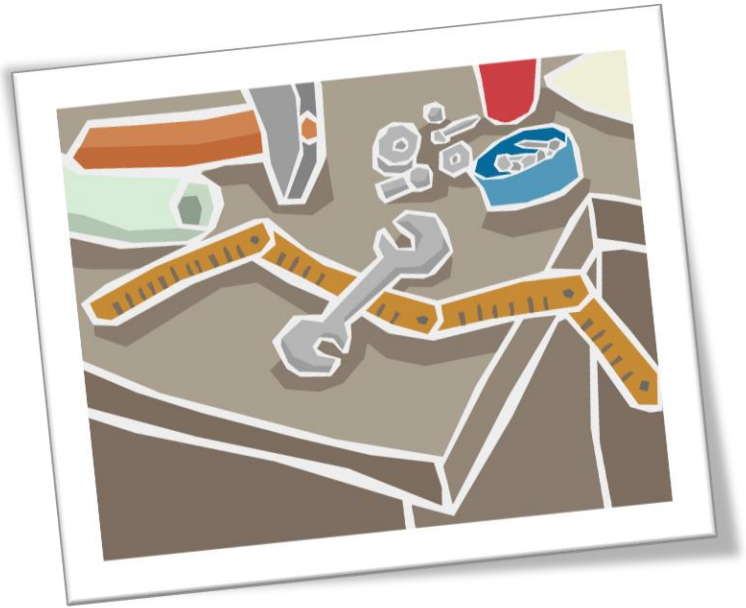
Travaux pratiques

- Objectif : créer et utiliser un cluster AKS.
- #cloud / azure



Travaux pratiques

- Objectif : voir les éléments constituant le cluster. Ce TP est le même que celui effectué au début.
- #cloud / inspect



Autres clusters

- Développement
  - https://computingforgeeks.com/deploy-production-kubernetes-cluster-with-ansible/
- Production
  - https://www.infoworld.com/article/3265059/10-kubernetes-distributions-leading-the-container-revolution.html
  - https://medium.com/better-programming/kubernetes-distributions-what-are-they-be2c438c8706
  - Exemple: Rancher https://rancher.com/docs/rancher/v2.x/en/quick-start-guide/deployment/

Démonstration

- https://github.com/SundeepPaluru/kubernetes-hyperv-vagrant