

# RedHat OpenShift, développement niveau 1, applications de conteneurisation

Ahmed ZIAD



# Objectifs pédagogiques

- Déployer des applications sur un cluster OpenShift et gérer ces applications
- Concevoir et construire des conteneurs d'applications assurant un déploiement réussi sur un cluster OpenShift
- Construire des applications conteneurisées à l'aide de la fonctionnalité source-to-image
- Créer des applications sur la base de modèles OpenShift
- Extraire un service d'une application monolithique et déployer ce service en tant que microservice dans le cluster
- Migrer des applications à exécuter sur un cluster OpenShift

# Plan du cours



Module 01: Rappels Kubernetes

---



Module 02: Déploiement et gestion d'applications dans un cluster OpenShift

---



Module 03: Conception d'applications conteneurisées pour OpenShift

---



Module 04: Publication d'images de conteneurs d'entreprise

---



Module 05: Build d'applications depuis OpenShift

---



Module 06: Personnalisation de versions source-to-image

---



Module 07: Création d'applications à partir de modèles OpenShift

---



Module 08: Gestion de déploiement d'applications

---

# Module 01:

## Rappels

## Kubernetes



Conteneurs et Architecture Kubernetes



Approche impérative vs. Approche déclarative (YAML)



Les objets de base K8s (Pods, Deployment, CronJob, StatefulSet, Services, ...)



La gestion des volumes avec les PVC

# Qu'est-ce que Docker

Docker est une plate-forme logicielle qui vous permet de créer, de tester et de déployer rapidement des applications, en regroupant les logiciels dans des unités standardisées appelées **conteneurs**.

# QU'EST-CE QUE LES CONTENANTS ?

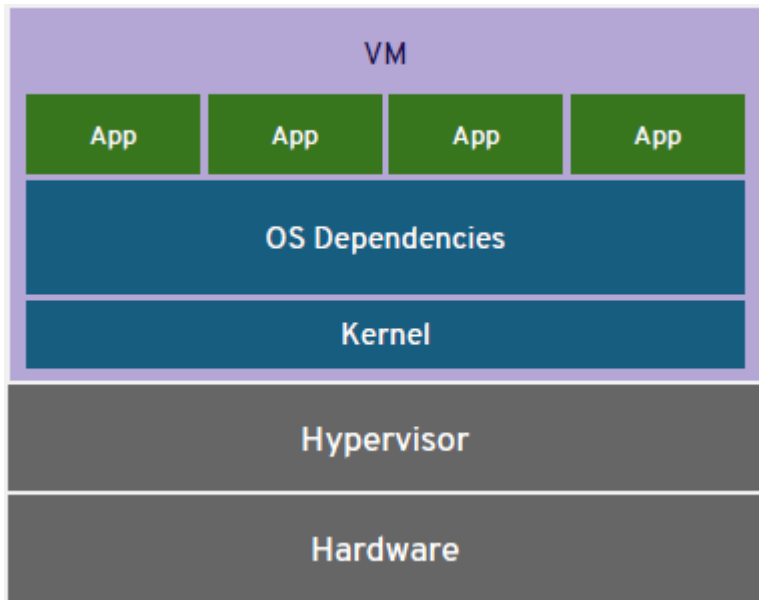
Cela dépend à qui vous demandez



- Processus d'application sur un noyau partagé
- Plus simple, plus léger et plus dense que les VM
- Portable dans différents environnements
- Empaqueter des applications avec toutes les dépendances
- Déploiement dans n'importe quel environnement en quelques secondes
- Facilement accessible et partagé

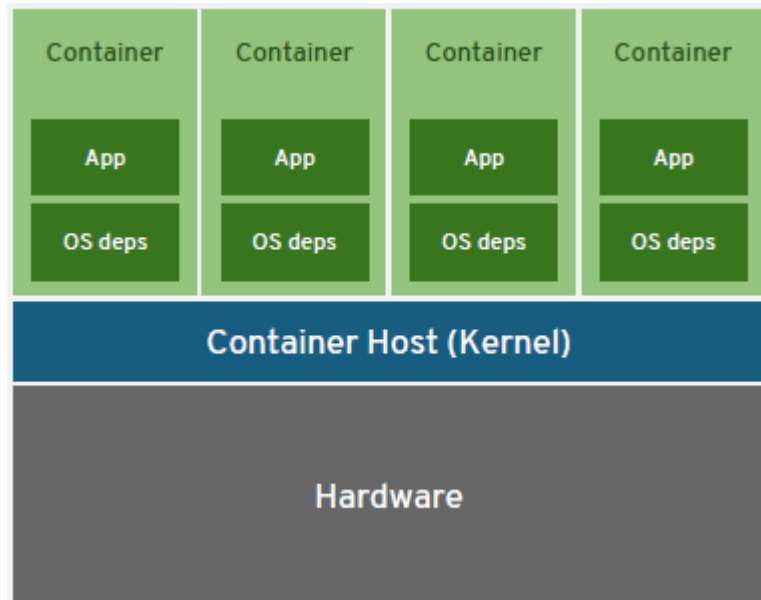
# MACHINES VIRTUELLES ET CONTENEURS

## Machines Virtuelles



les machines virtuelles sont isolées les applications ne le sont pas.

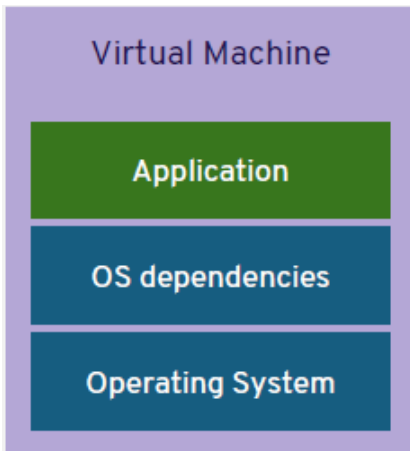
## Conteneurs



les conteneurs sont isolés les applications aussi

# MACHINES VIRTUELLES ET CONTENEURS

## Machines Virtuelles



Isolement de VM



Système d'exploitation complet



Calcul statique

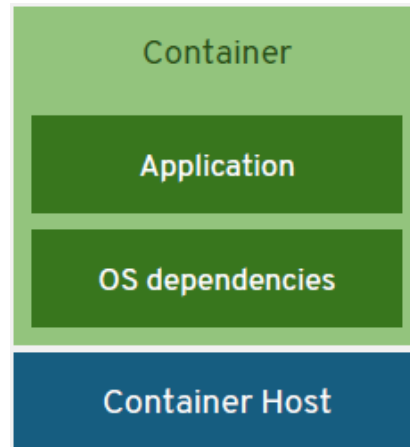


Mémoire statique



Utilisation élevée des ressources

## Conteneurs



Isolement des conteneurs



Noyau partagé



Calcul extensible



Mémoire extensible



Faible utilisation des ressources



# Docker Hub



## Explore Official Repositories

 <b>nginx</b> official	6.1K STARS	10M+ PULLS	<a href="#">&gt;</a> DETAILS
 <b>redis</b> official	3.8K STARS	10M+ PULLS	<a href="#">&gt;</a> DETAILS
 <b>busybox</b> official	1.0K STARS	10M+ PULLS	<a href="#">&gt;</a> DETAILS
 <b>ubuntu</b> official	6.1K STARS	10M+ PULLS	<a href="#">&gt;</a> DETAILS
 <b>registry</b> official	1.5K STARS	10M+ PULLS	<a href="#">&gt;</a> DETAILS
 <b>alpine</b> official	2.3K STARS	10M+ PULLS	<a href="#">&gt;</a> DETAILS
 <b>mysql</b> official	4.4K STARS	10M+ PULLS	<a href="#">&gt;</a> DETAILS
 <b>mongo</b> official	3.3K STARS	10M+ PULLS	<a href="#">&gt;</a> DETAILS

# Dockerfile

- Il est possible de construire vos propres images en lisant les instructions d'un Dockerfile

```
1  # Use a container with Go pre-installed
2  FROM quay.io/projectquay/golang:1.17
3
4  # Copy our source file into the container
5  COPY src/hello-world.go /go/hello-world.go
6
7  # Set the default environment variables
8  ENV MESSAGE "Welcome! You can change this message by editing the MESSAGE environment variable."
9  ENV HOME /go
10
11 # Set permissions to the /go folder (for OpenShift)
12 RUN chgrp -R 0 /go && chmod -R g+rwX /go
13
14 # Just documentation.
15 # This container needs Docker or OpenShift to help with networking
16 EXPOSE 8080
17
18 # OpenShift picks up this label and creates a service
19 LABEL io.openshift.expose-services 8080/http
20
21 # OpenShift uses root group instead of root user
22 USER 1001
23
24 # Command to run when container starts up
25 CMD go run hello-world.go
26
```

# docker-compose

- Permet d'exécuter des applications Docker multi-conteneurs en lisant les instructions d'un fichier **docker-compose.yml**

```
version: "2"
services:
  my-application:
    build: ./
    ports:
      - "8000:8000"
    environment:
      - CONFIG_FILE
  db:
    image: postgres
  redis:
    image: redis
    command: redis-server --save "" --appendonly no
    ports:
      - "6379"
```

**“Kubernetes est une plate-forme portable,  
extensible et open source pour la gestion des  
charges de travail et des services conteneurisés,  
qui facilite à la fois la configuration déclarative et  
l'automatisation.»**

**”** Kubernetes Official  
Documentation



# KUBERNETES : K8s

K8s (Kate's) est simplement l'abréviation de Kubernetes.  
Le 8 représente les 8 lettres entre K et S !



# KUBERNETES

- COE développé par Google, devenu open source en 2014
- Adapté à tout type d'environnement
- Devenu populaire en très peu de temps
- Premier projet de la CNCF



# COMPOSANTS

- Kubernetes est écrit en Go, compilé statiquement.
- Un ensemble de binaires sans dépendance
- Faciles à conteneuriser et à packager
- Peut se déployer uniquement avec des conteneurs sans dépendance d'OS

# K8s : Fonctionnalités

## **Orchestration de conteneurs**

L'objectif principal de Kubernetes est de gérer dynamiquement les conteneurs sur plusieurs systèmes hôtes.

## **Fiabilité des applications**

Kubernetes facilite la création d'applications fiables, autoréparateurs et évolutifs.

## **Automatisation**

Kubernetes offre une variété de fonctionnalités permettant d'automatiser la gestion de vos applications de conteneur.



# Kubernetes cluster



Kubernetes Cluster

Server

Container

Container

Container

Server

Container

Container

Server

Container


Container

# K8s : COMPOSANTS DU CONTROL PLANE

- **etcd**: Base de données.
- **kube-apiserver** : API server qui permet la configuration d'objets Kubernetes (Pod, Service, Deployment, etc.)
- **kube-proxy** : est responsable de la maintenance des règles réseau dans les nœuds Il stocke les valeurs de mise en réseau des nœuds dans un IPTABLE virtuel au sein du nœud et permet la connexion aux pods à l'intérieur et à l'extérieur du cluster.
- **kube-scheduler** : Implémente les fonctionnalités de scheduling.
- **kube-controller-manager** : Responsable de l'état du cluster, boucle infinie qui régule l'état du cluster afin d'atteindre un état désiré.

# K8s : COMPOSANTS DU CONTROL PLANE

## Control Plane



Control plan est une collection de plusieurs composants responsables de la gestion du cluster dans sa globalité.

Essentiellement, le Control Plan gère le cluster.

Les composants d'un Control plan peuvent s'exécuter

sur n'importe quelle machine du cluster, mais s'exécutent généralement sur des machines contrôleurs dédiées.

# K8s : COMPOSANTS DU CONTROL PLANE

## Control Plane



The diagram shows a large rounded rectangle with a dashed orange border representing the Control Plane. Inside this rectangle, on the right side, is a smaller rectangle with a solid orange border containing the text 'Kube-api-server'.

Kube-api-server

Authentifie l'utilisateur

Valide la requete

Récupère les données

Mis à jour etcd

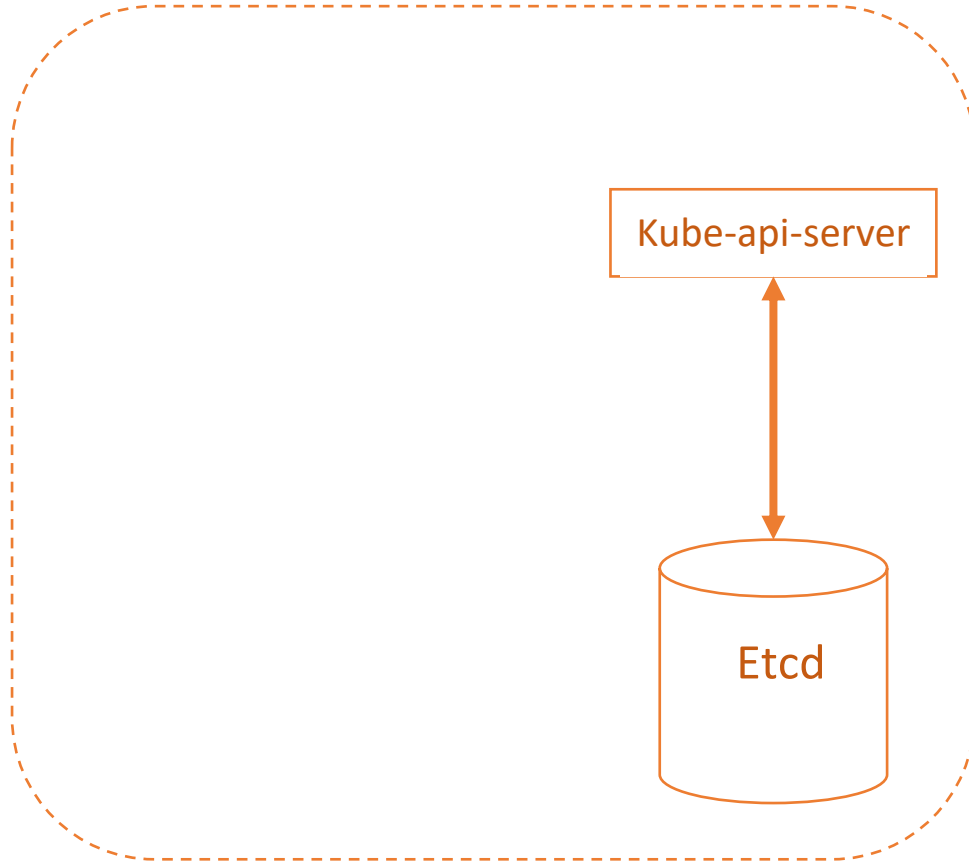
Lance le scheduler

Communique avec kubelet

```
sudo cat /etc/kubernetes/manifests/kube-apiserver.yaml
```

# K8s : COMPOSANTS DU CONTROL PLANE

## Control Plane

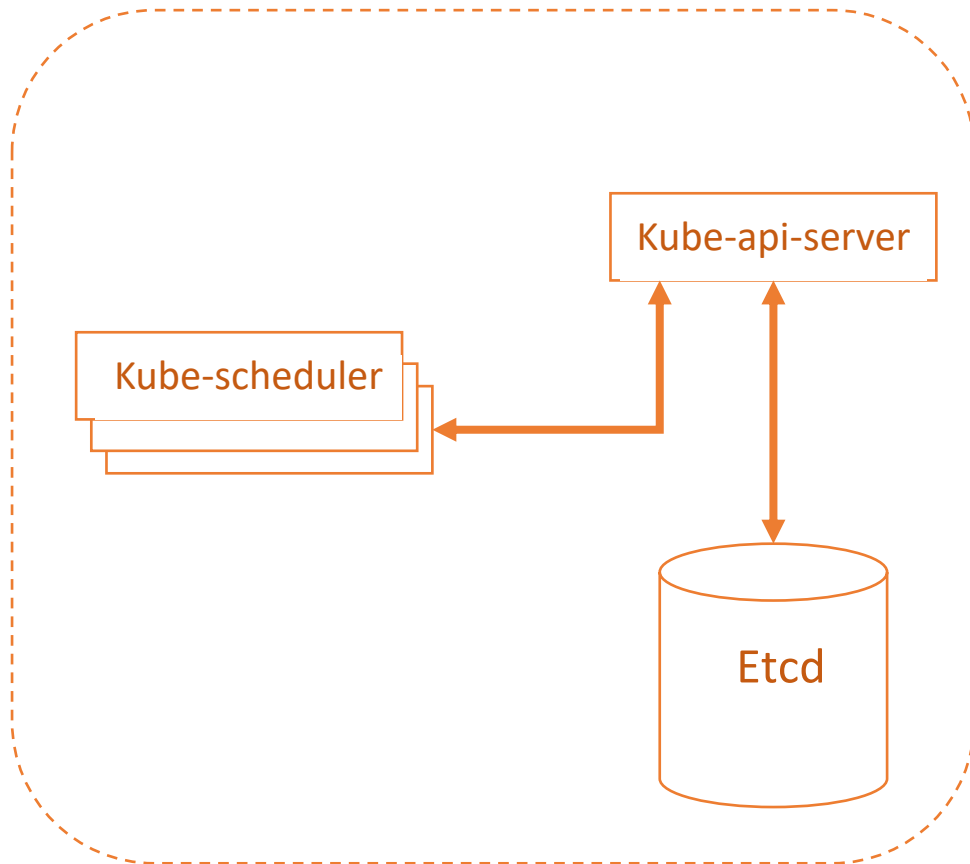


Etcd est le magasin de données principal pour le cluster Kubernetes.

Il offre un stockage à haute disponibilité de toutes les données relatives à l'état du cluster.

# K8s : COMPOSANTS DU CONTROL PLANE

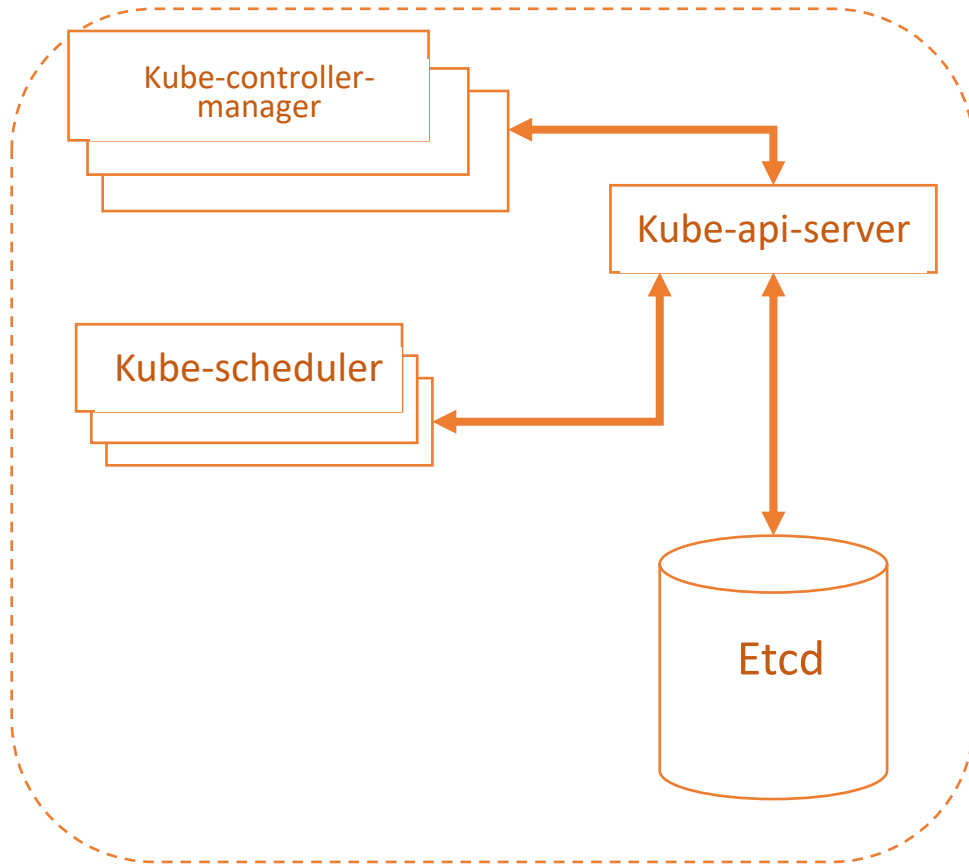
## Control Plane



kube-scheduler gère la planification, le processus de sélection d'un nœud disponible dans le cluster sur lequel exécuter les conteneurs.

# K8s : COMPOSANTS DU CONTROL PLANE

## Control Plan



Vérifie le status des objets

Remédier à la situation

Node Monitor Period = 5 secondes

Node Monitor Grace Period = 40 secondes

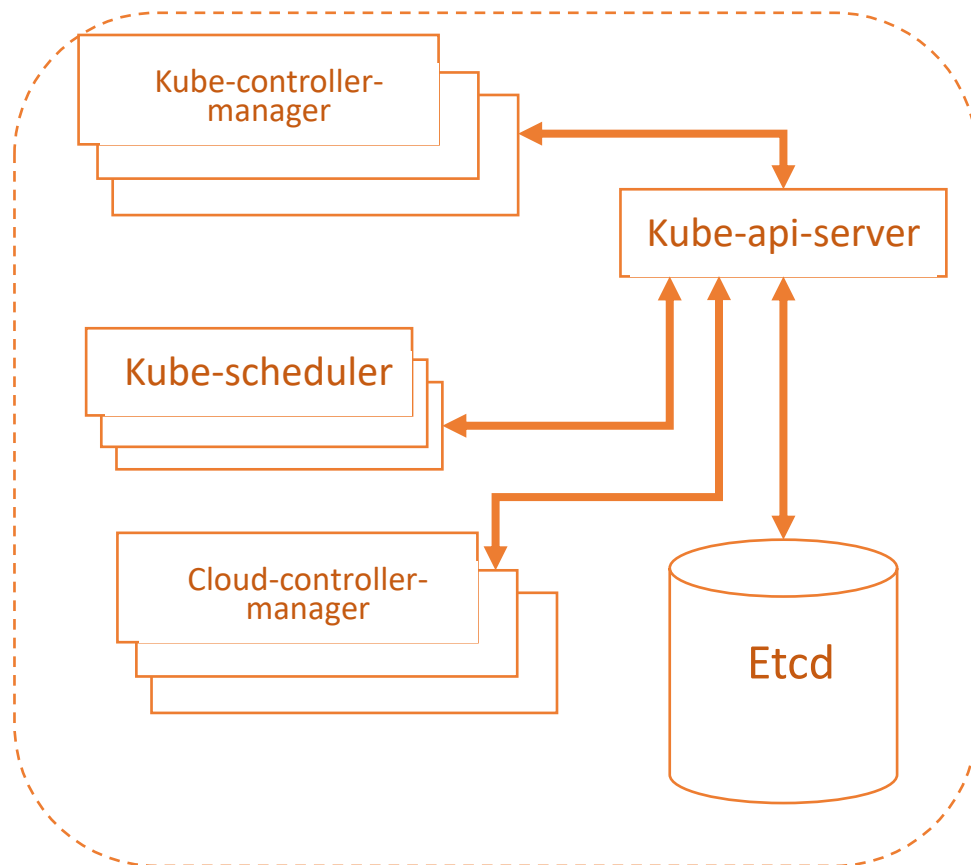
Pod Eviction Timeout = 5 m

kube-controller-manager exécute une collection de plusieurs utilitaires de contrôleur dans un seul processus.

Ces contrôleurs effectuent une variété de tâches liées à l'automatisation au sein du cluster Kubernetes.

# K8s : COMPOSANTS DU CONTROL PLANE

## Control Plane



cloud-controller-manager fournit une interface entre Kubernetes et les plates-formes cloud.

Il isole les composants qui interagissent avec la plate-forme cloud des composants qui interagissent uniquement avec votre cluster.

Il n'est utilisé que lors de l'utilisation des ressources basées sur le cloud avec Kubernetes.



# ETCD

- Base de données de type Clé/Valeur
- Stocke l'état d'un cluster Kubernetes
- Point sensible (stateful) d'un cluster Kubernetes
- Projet intégré à la CNCF

# KUBE-APISERVER

- Les configurations d'objets (Pods, Service, RC, etc.) se font via l'API server
- Un point d'accès à l'état du cluster aux autres composants via une API REST
- Tous les composants sont reliés à l'API server

# KUBE-SCHEDULER

- Planifie les ressources sur le cluster
- En fonction de règles implicites (CPU, RAM, stockage disponible, etc.)
- En fonction de règles explicites (règles d'affinité et anti-affinité, labels, etc.)

# KUBE-PROXY

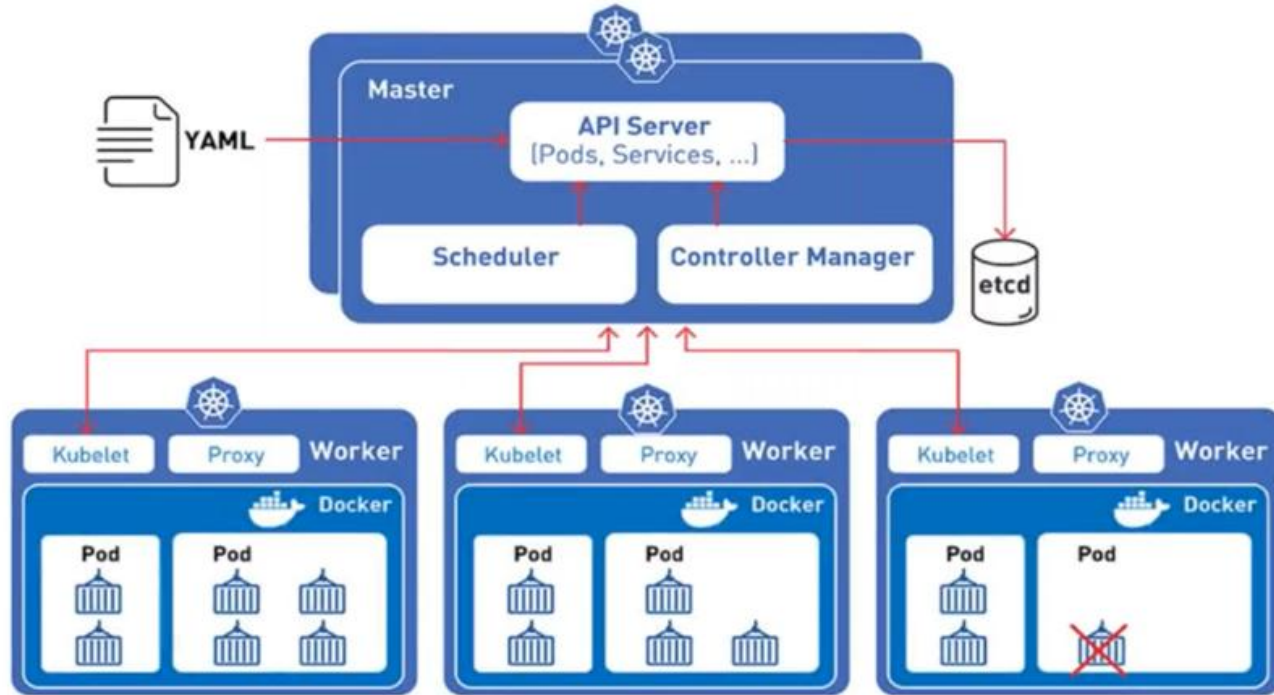
- L'agent réseau qui s'exécute sur chaque nœud (Control Plane et Workers), responsable des mises à jour dynamiques et de la maintenance de toutes les règles de mise en réseau sur le nœud. Il transmet les demandes de connexion aux conteneurs des pods.
- Il implémente les règles de transfert définies par les utilisateurs via les objets de l'API de service.
- Responsable de la publication des Services.
- Utilise iptables (créées dans chaque node)
- Route les paquets à destination des conteneurs et réalise le load balancing TCP/UDP/SCTP

# KUBE-CONTROLLER-MANAGER

- Boucle infinie qui contrôle l'état d'un cluster
- Effectue des opérations pour atteindre un état donné
- De base dans Kubernetes : replication controller, endpoints controller, namespace controller et serviceaccounts controller

# K8s : The big picture

## Kubernetes Architecture



# Gestion des objets et ressources

Kubernetes prend en charge deux approches principales pour gérer les ressources et les configurations :

- l'approche impérative
- l'approche déclarative

# Approche Impérative

L'approche impérative consiste à donner des instructions spécifiques à Kubernetes sur les actions à effectuer. Cela signifie que vous décrivez étape par étape les opérations à réaliser pour atteindre l'état souhaité du cluster. Par exemple, vous pouvez utiliser des commandes `kubectl` pour créer, mettre à jour ou supprimer des ressources directement.



# Approche Impérative

- Création d'un déploiement :

```
kubectl create deployment my-app --image=my-container-image
```

- Mise à l'échelle du déploiement :

```
kubectl scale deployment my-app --replicas=3
```

- Exposition du déploiement en tant que service :

```
kubectl expose deployment my-app --port=8080 --target-port=80
```

# Approche Déclarative

L'approche déclarative consiste à décrire l'état désiré du cluster et à laisser Kubernetes prendre les mesures nécessaires pour atteindre cet état. Vous décrivez les spécifications de vos ressources dans des fichiers de configuration (par exemple, des fichiers YAML) et utilisez kubectl pour appliquer ces fichiers sur le cluster. Kubernetes se charge ensuite d'apporter les modifications nécessaires pour aligner l'état actuel du cluster sur l'état déclaré dans les fichiers.

# Approche Déclarative

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app-container
          image: my-container-image
          ports:
            - containerPort: 80
```

```
kubectl apply -f my-app.yaml
```

# API RESOURCES

- Namespaces
- Labels
- Pods
- Deployments
- DaemonSets
- StatefulSets
- Jobs
- Cronjobs
- Service
- ...

# NAMESPACES

- Fournissent une séparation logique des ressources :
  - Par utilisateurs
  - Par projet / applications
  - Autres...
- Les objets existent uniquement au sein d'un namespace donné
- Évitent la collision de nom d'objets

# LABELS

- Système de clé/valeur
- Organisent les différents objets de Kubernetes (Pods, RC, Services, etc.) d'une manière cohérente qui reflète la structure de l'application
- Corrèlent des éléments de Kubernetes : par exemple un service vers des Pods

# LABELS

Exemple de label :

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

# POD

- Ensemble logique composé de un ou plusieurs conteneurs
- Les conteneurs d'un pod fonctionnent ensemble (instanciation et destruction) et sont orchestrés sur un même hôte
- Les conteneurs partagent certaines spécifications du
- Pod :
  - La stack IP (network namespace)
  - Inter-process communication (PID namespace)
  - Volumes
- C'est la plus petite et la plus simple unité dans Kubernetes



# DEPLOYMENT

- Permet d'assurer le fonctionnement d'un ensemble de Pods.
- Version, Update et Rollback.
- Anciennement appelés Replication Controllers

# DAEMONSET

- Assure que tous les noeuds exécutent une copie du pod
- Ne connaît pas la notion de replicas.
- Utilisé pour des besoins particuliers comme :
  - l'exécution d'agents de collection de logs comme *fluentd* ou *logstash*
  - l'exécution de pilotes pour du matériel comme nvidia-plugin
  - l'exécution d'agents de supervision comme *NewRelic* agent ou *Prometheus* node exporter

# DaemonSet : Définition

daemon-set-definition.yaml

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: monitoring-daemon
spec:
  selector:
    matchLabels:
      app: monitoring-agent
  template:
    metadata:
      labels:
        app: monitoring-agent
    spec:
      containers:
        - name: monitoring-agent
          image: mon
```



```
kubectl create -f daemon-set-definition.yaml
```

# STATEFULSET

- Un StatefulSet est une ressource k8s qui représente une charge de travail.
- Similaire à un déploiement : il gère un ensemble de pods en fonction d'une spécification de pod.
- La particularité d'un StatefulSet est qu'il gère les applications avec état .
- Chaque replica de pod est créé par ordre d'index (par exemple mysql-0, mysql-1, mysql-2), et cette identité est maintenue même après la reprogrammation des pods.
- Nécessite un Persistent Volume et un Storage Class.
- Supprimer un StatefulSet ne supprime pas le PV associé.

# JOB

- JOB est un objet k8s représentant une charge de travail temporaire qui s'exécutera pour réaliser une tâche, puis se terminera.
- Par exemple, vous pouvez avoir une tâche qui démarrera un pod pour effectuer un travail de nettoyage. Une fois le pod terminé, il se fermera et ne redémarrera jamais.
- Crée des pods et s'assurent qu'un certain nombre d'entre eux se terminent avec succès.
- Peut exécuter plusieurs pods en parallèle
- Si un noeud du cluster est en panne, les pods sont relancés dans un autre noeud.

# JOB

```
apiVersion: batch/v1
kind: Job
metadata:
  name: my-job
spec:
  template:
    spec:
      containers:
      - name: my-job-container
        image: my-container-image
        command: ["sh", "-c", "echo Hello from the job"]
      restartPolicy: Never
```

# CRON JOB

- Un CronJob permet de lancer des Jobs de manière planifiée.
- La programmation des Jobs se définit au format Cron.
- Le champ *jobTemplate* contient la définition de l'application à lancer comme Job.

# CRONJOB

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: my-cronjob
spec:
  schedule: "*/5 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: my-cronjob-container
              image: my-container-image
              command: ["sh", "-c", "echo Hello from the cron job"]
          restartPolicy: OnFailure
```



# Persistent Volumes

- Stockage défini par l'administrateur dans le cluster
- Détails de mise en œuvre pour votre stockage
- Cycle de vie indépendant du Pod
- Géré par le Kubelet
  - Mappe le stockage dans le nœud
  - Expose PV comme support à l'intérieur du conteneur

<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>

# PERSISTENTVOLUMECLAIMS

- Une *PersistentVolumeClaim* est une demande de stockage par un utilisateur/une ressource.
- Le stockage donné est extrait d'un *PersistentVolume*, qui est un objet k8s abstrait qui représente un stockage.
- Il existe différentes classes de stockage, qui représentent les différentes implémentations de stockage pouvant être utilisées par k8s.
- Le *PersistentVolumeClaim* spécifie également le mode d'accès au stockage (*ReadWriteOnce*, *ReadOnlyMany* ou *ReadWriteMany*)

Demo

Kubernetes : creation des objets

Demo

Kubernetes : kubectl

Demo

Gestion du stockage de conteneurs avec des volumes Kubernetes

# Module 02:

## Déploiement et gestion d'applications dans un cluster OpenShift



Rappels architecture OpenShift



Gestion de projets OpenShift



Configuration de l'accès à l'application via les Routes

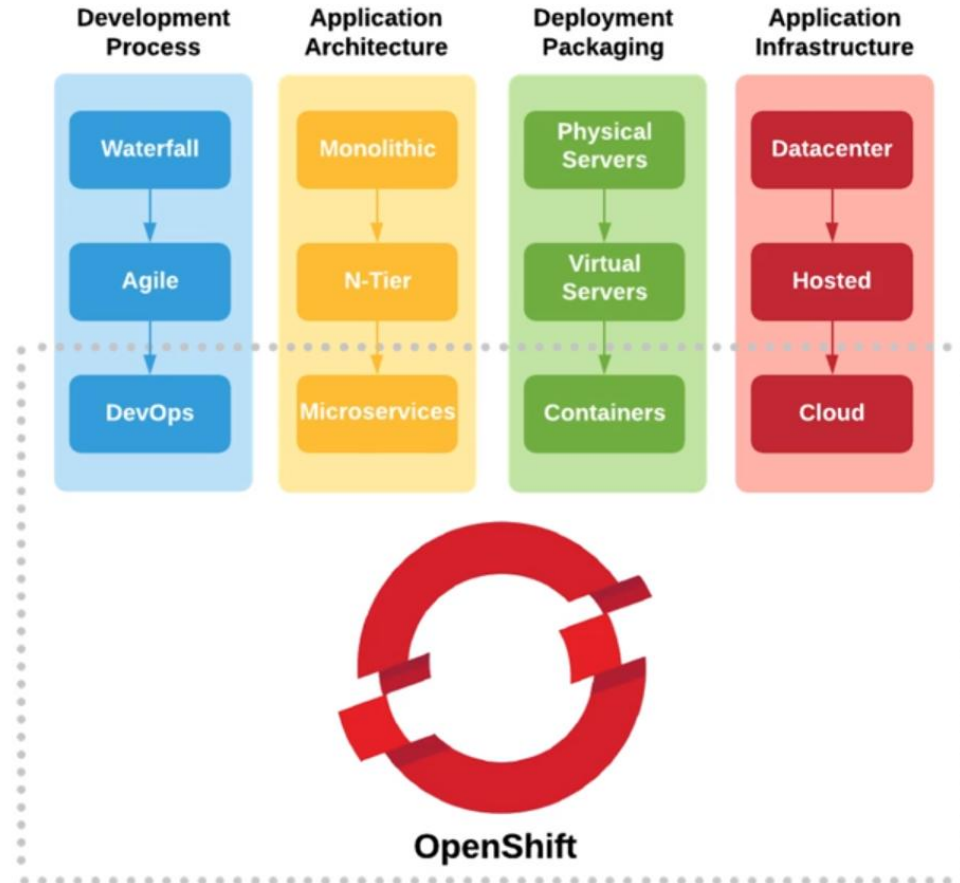


Gestion d'une application sur OpenShift

# Qu'est ce qu'OpenShift ?

- OpenShift constitue une **plateforme d'applications basée sur Kubernetes**.
- Ce logiciel est utilisé pour construire des environnements d'application et de développement décentralisés et capables de se
- mettre à l'échelle.
- Sa fonction d'orchestration des conteneurs se base sur [Kubernetes](#) et vient s'ajouter aux [nombreuses autres fonctions](#) issues de l'écosystème Open Source de la Cloud Native Computing Foundation (CNCF), toutes testées, mises en paquets et prises en charge par Red Hat, pour former une plateforme d'applications complète.
- En se basant sur OpenShift, des solutions propriétaires de PaaS, SaaS et [Containers as a Service](#) (CaaS) peuvent être implémentées.
- Ce logiciel permet une gestion complète du cycle de vie de l'application, y compris le développement, le déploiement, le pilotage et la maintenance.

# OpenShift – Plattform Moderne

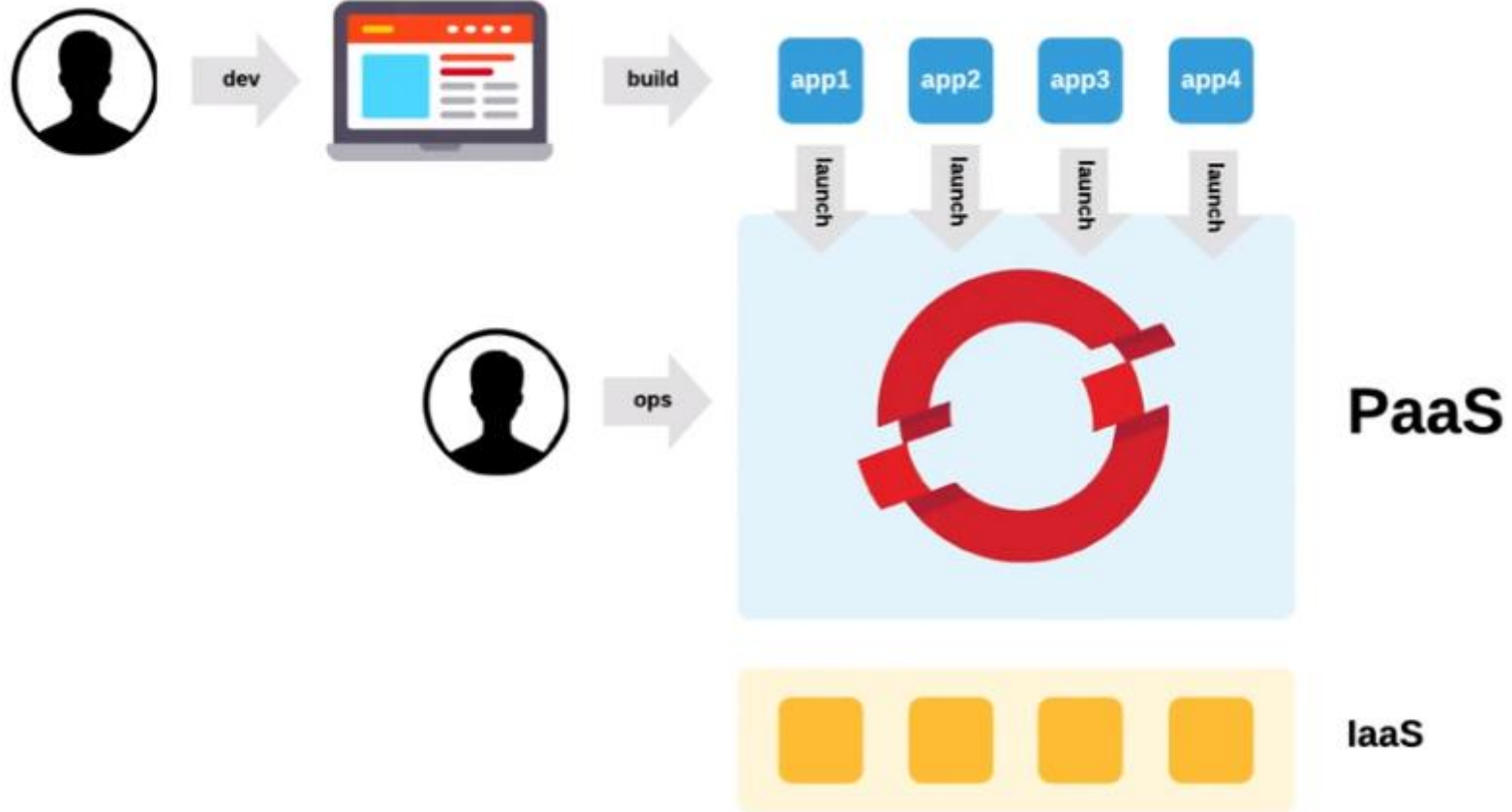




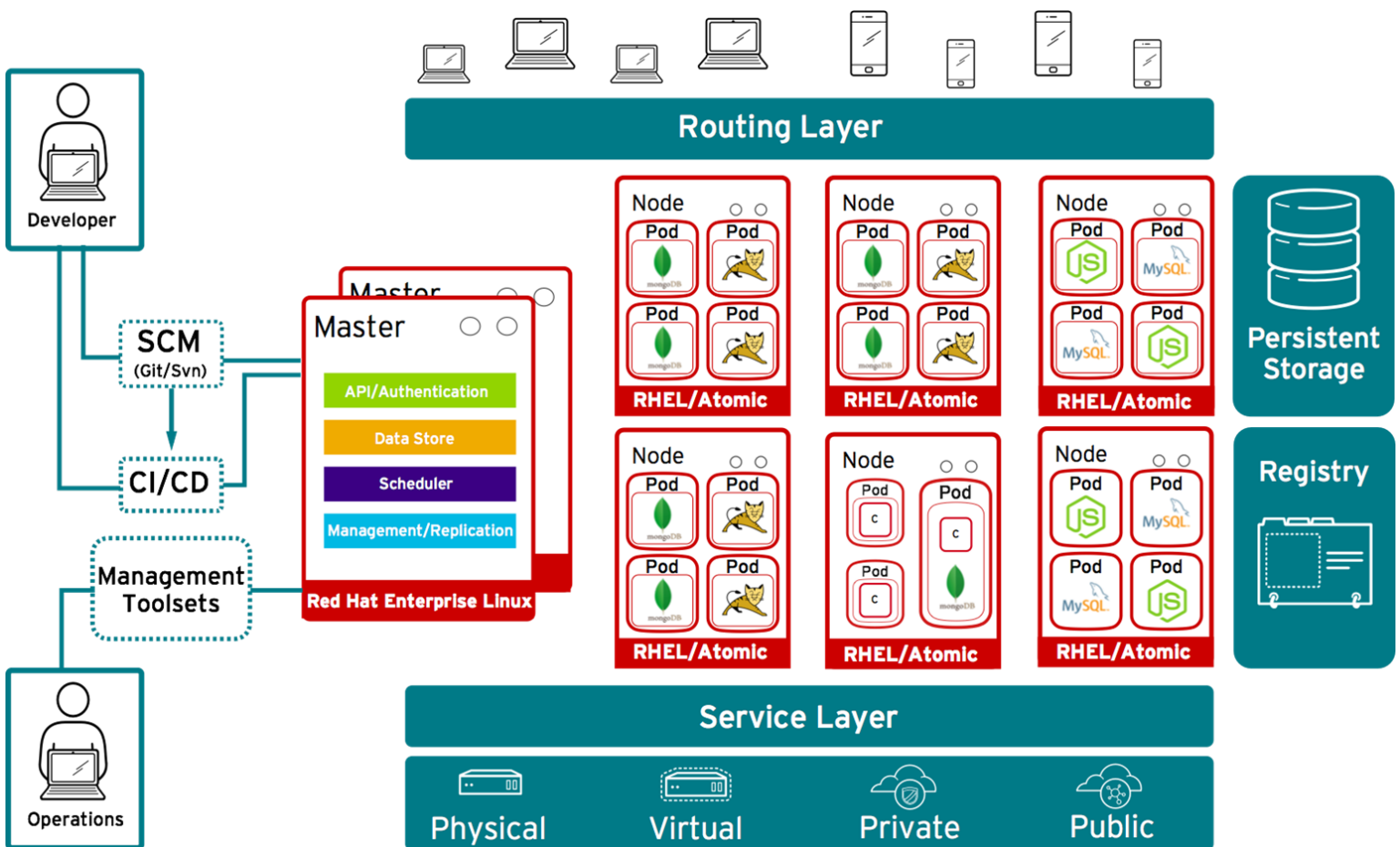
# OpenShift – Quelques fonctionnalités



# OpenShift – PaaS



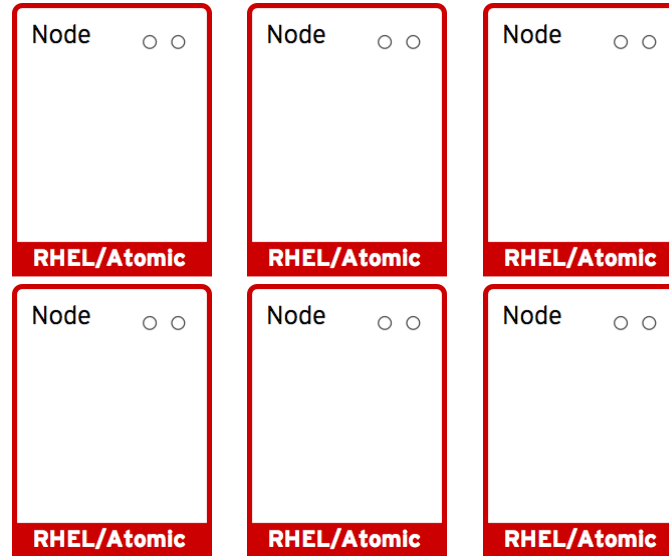
# OpenShift: Big Picture



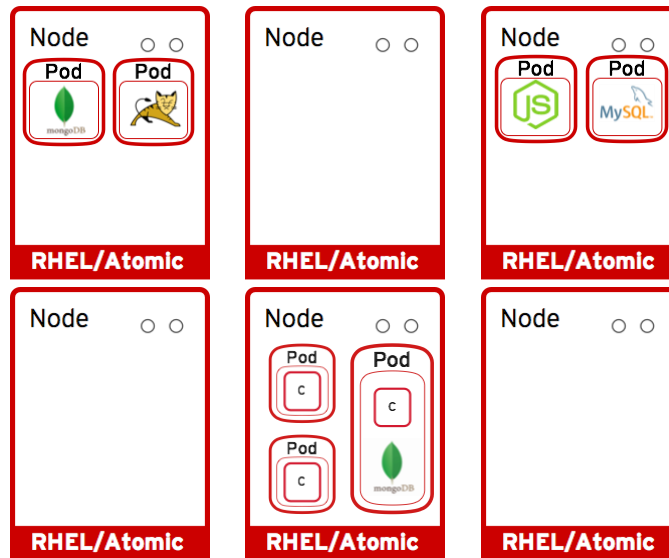
# OpenShift s'exécute sur l'infrastructure de votre choix



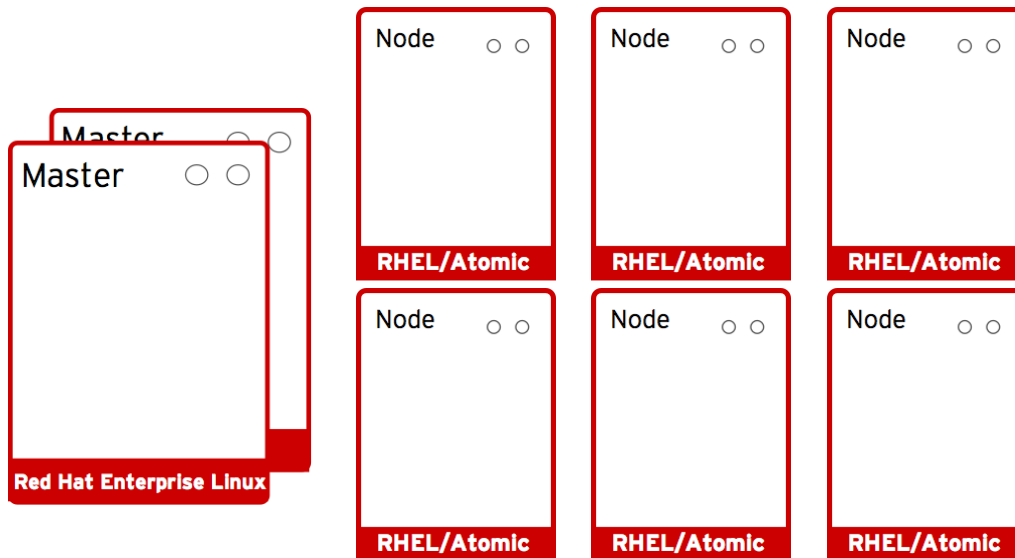
# Les nœuds sont des instances de RHEL où les applications seront exécutées



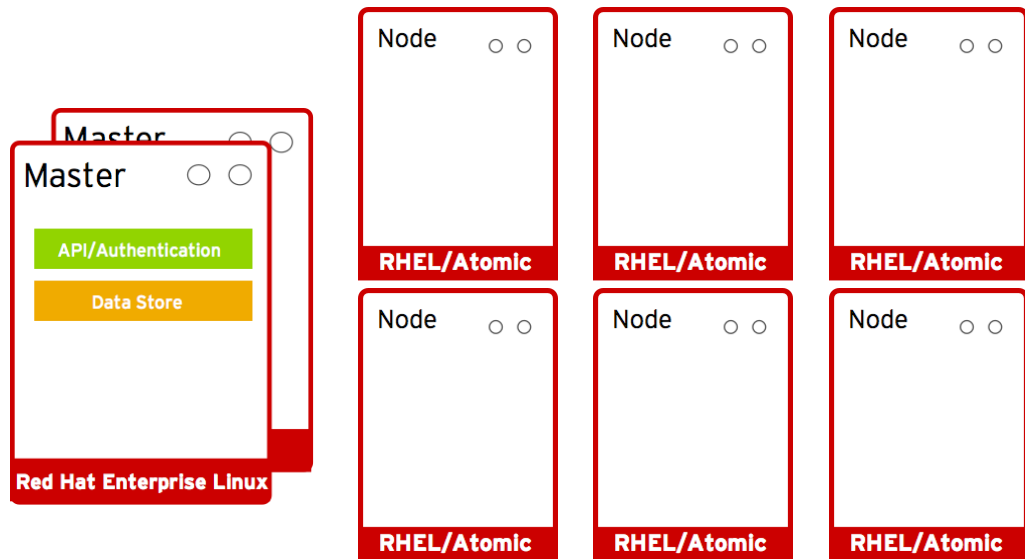
# Les applications s'exécutent dans des conteneurs Docker sur chaque nœud



# Les maîtres tirent parti de kubernetes pour orchestrer les nœuds/applications

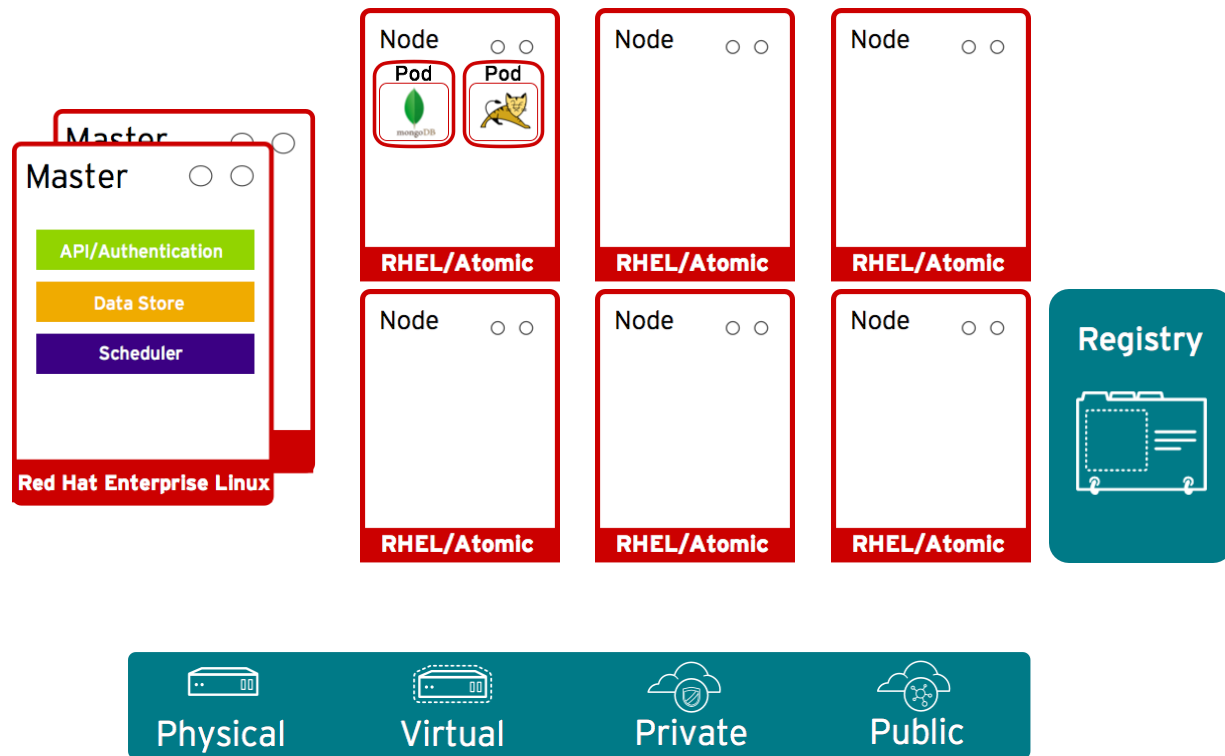


# Le master utilise le magasin de données clé-valeur etcd pour la persistance

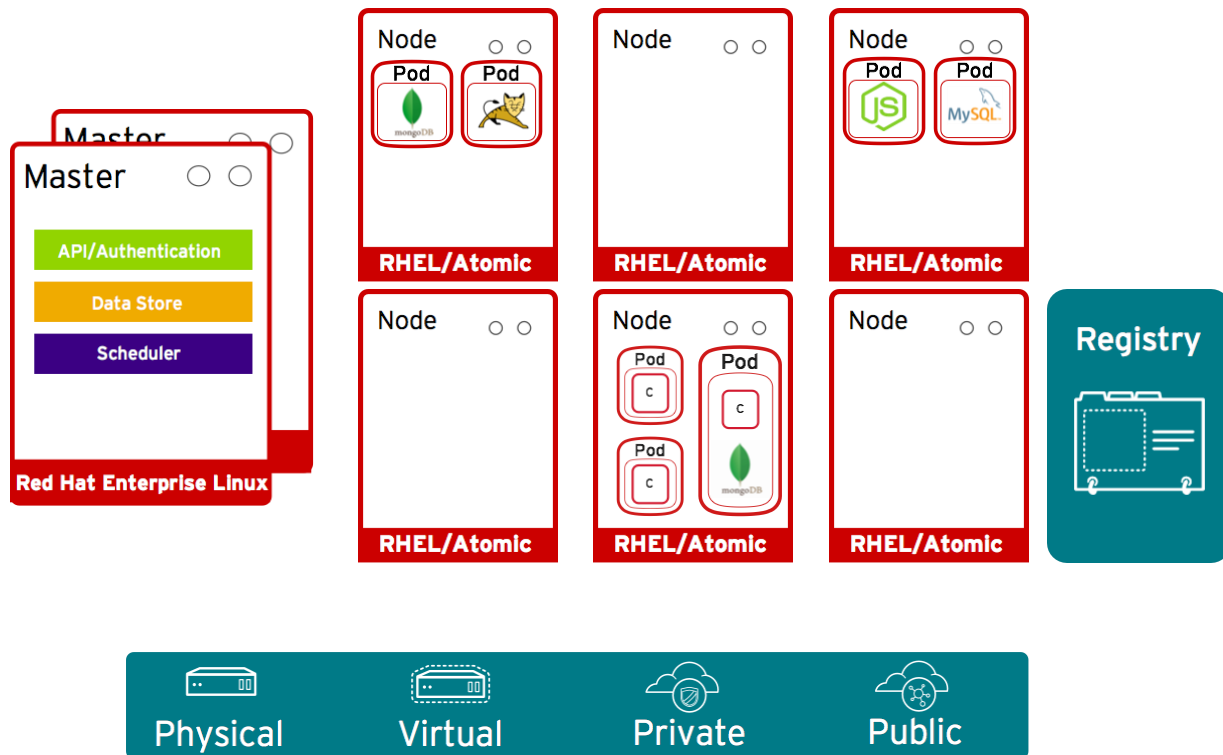




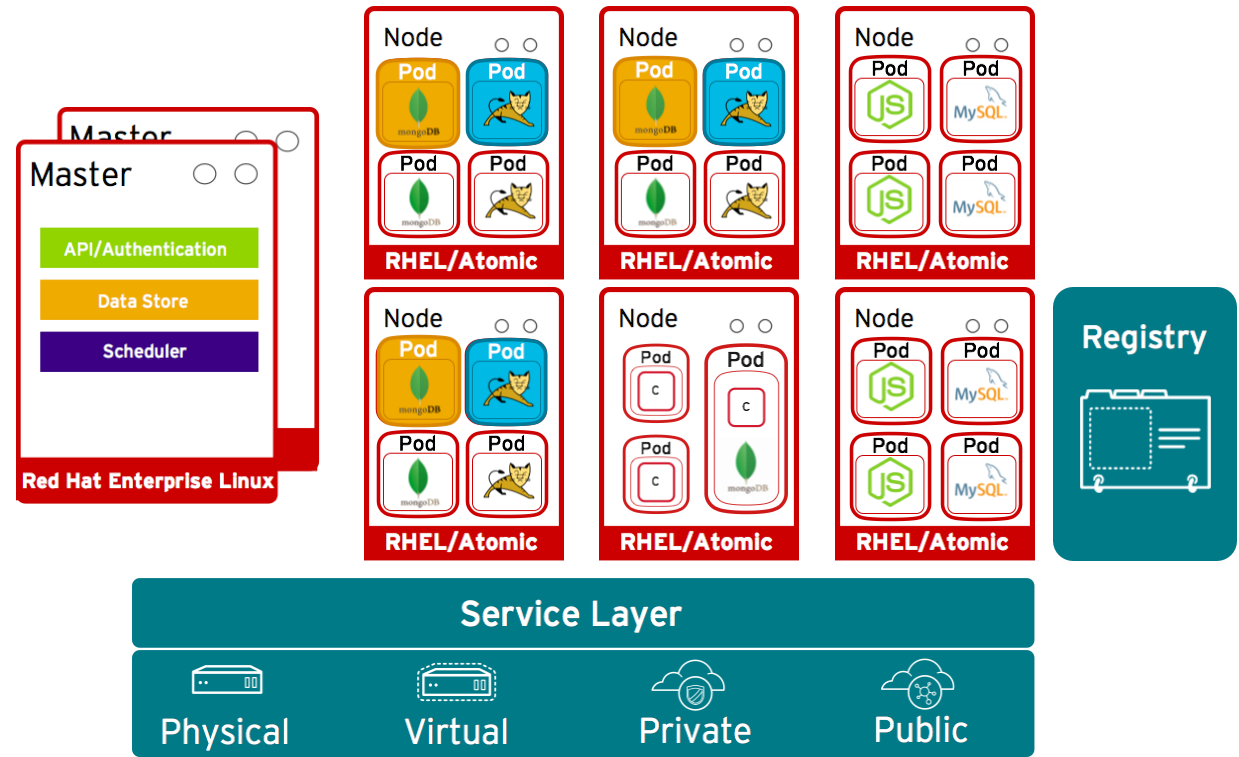
# Le maître fournit un planificateur pour le placement des pods sur les nœuds



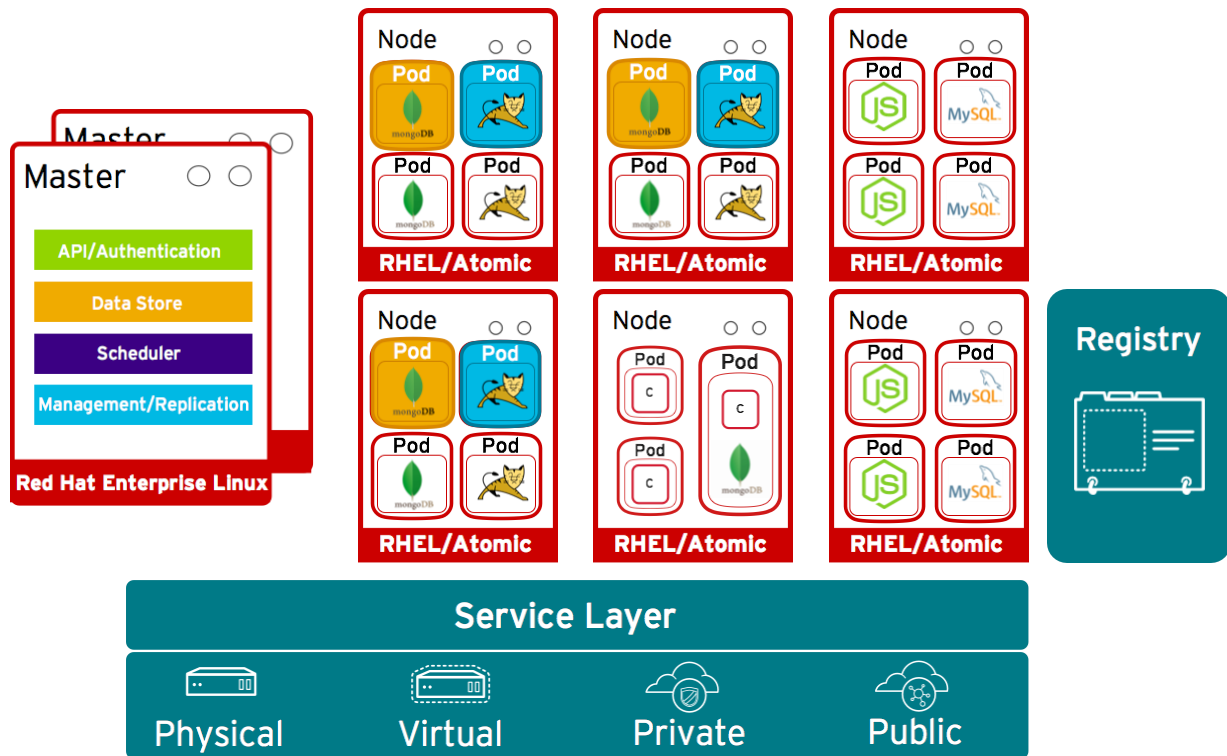
# Le placement des pods est déterminé en fonction de la politique définie



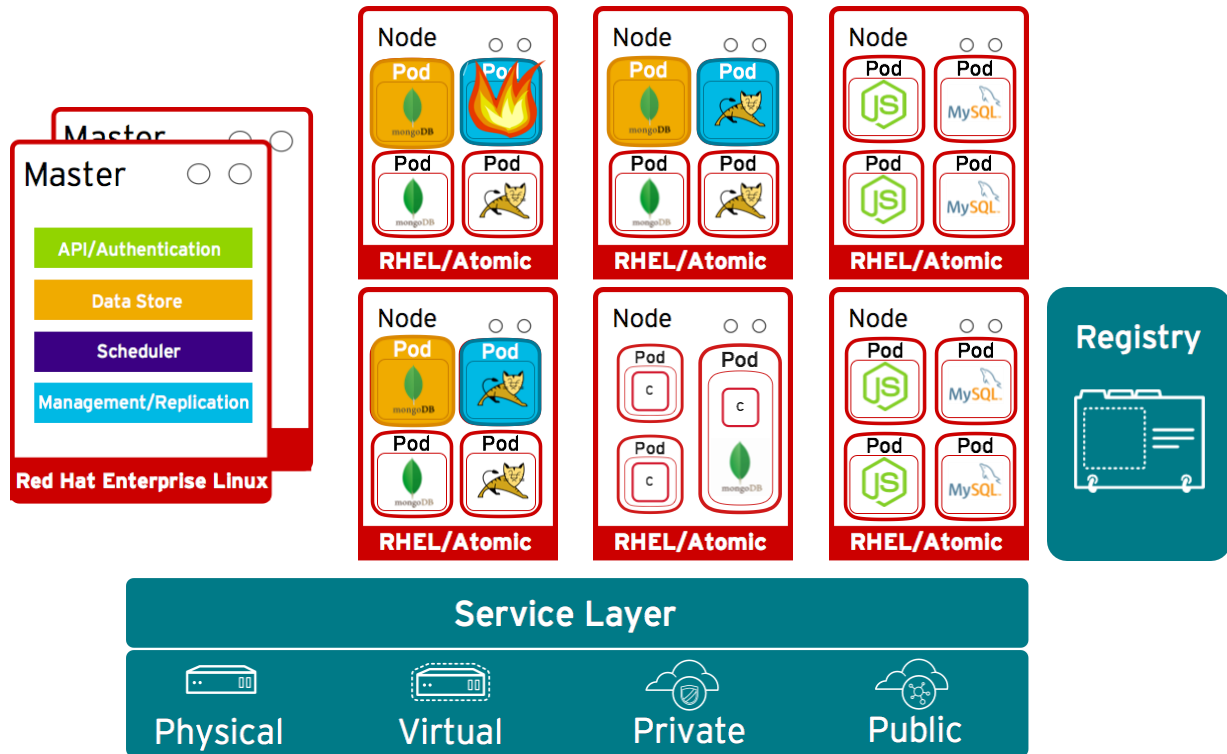
Les services permettent aux modules associés de se connecter les uns aux autres



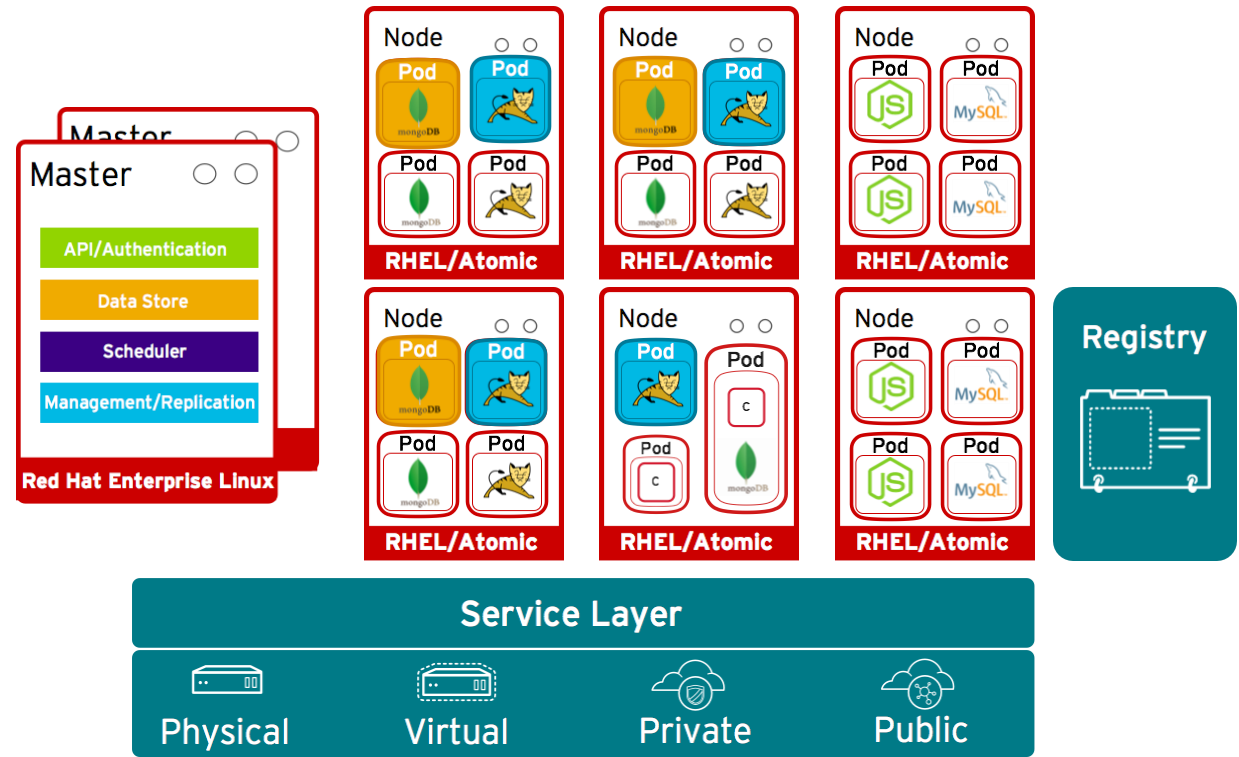
# Le contrôleur de gestion/réplication gère le cycle de vie du pod



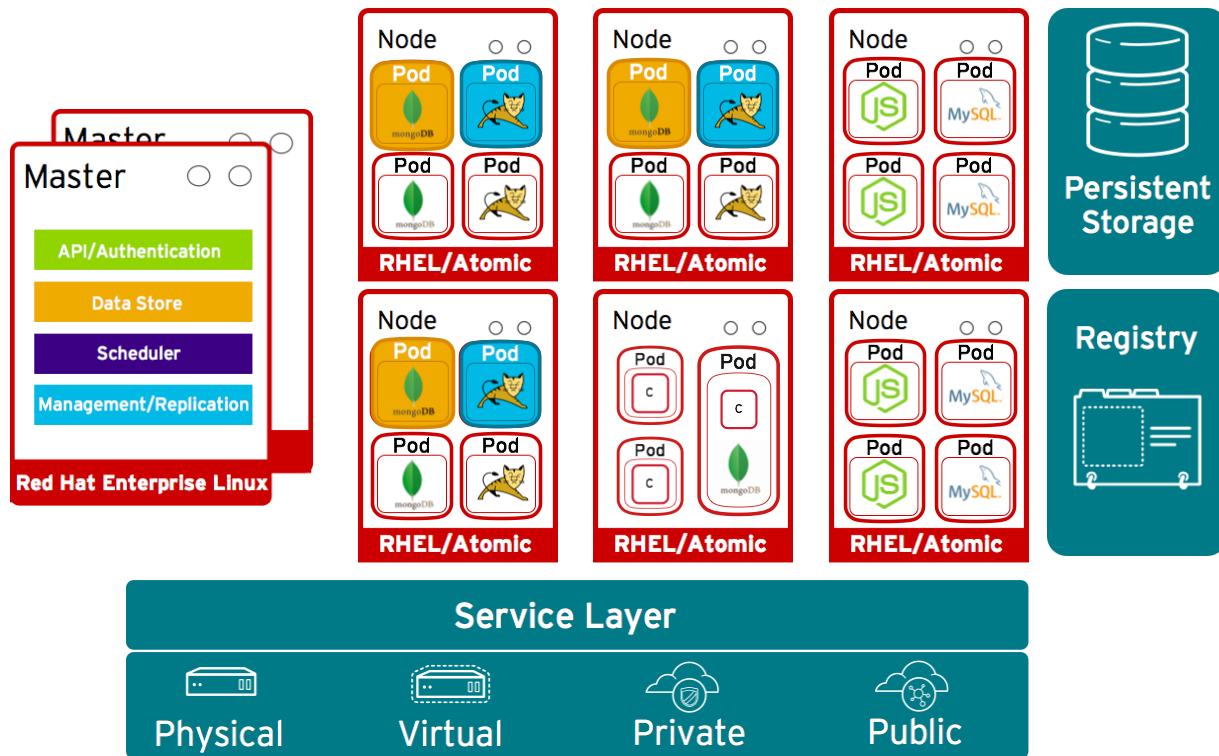
# Et si un pod tombe en panne ?



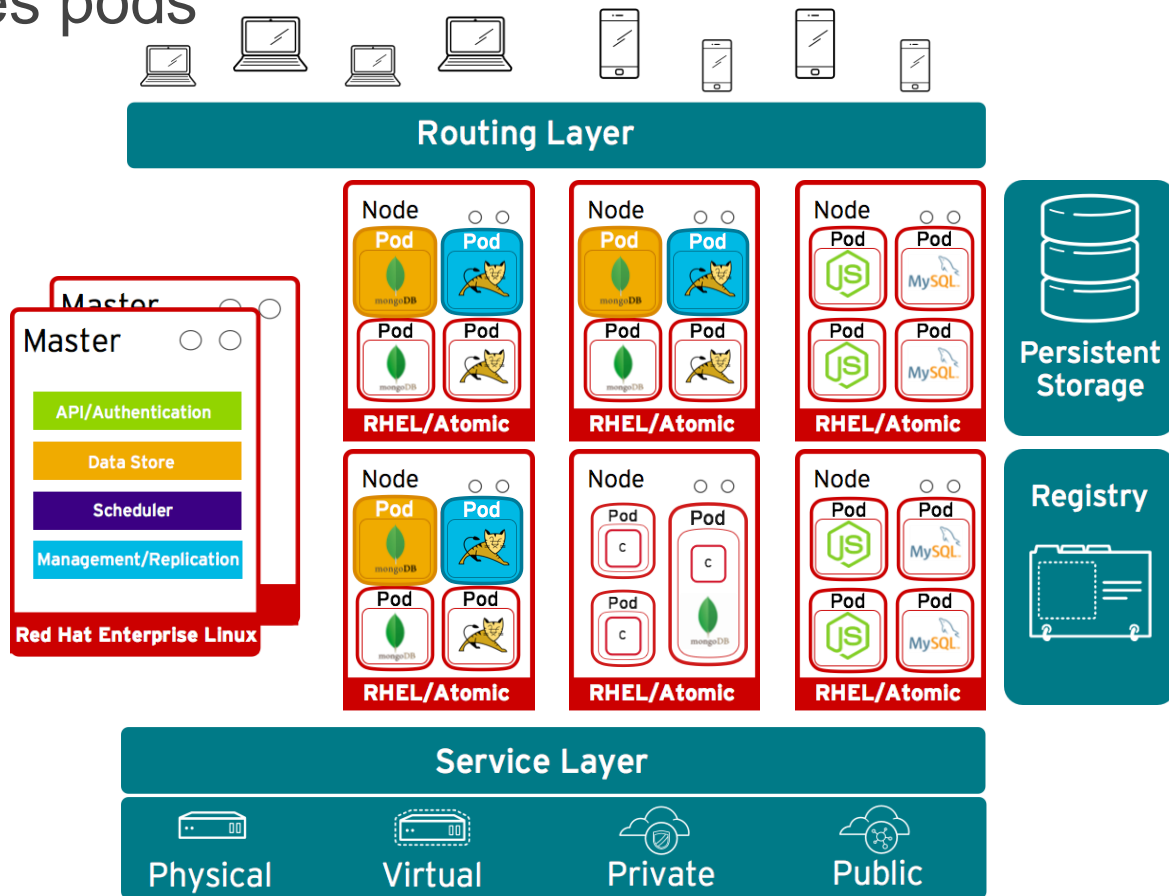
# OpenShift récupère et déploie automatiquement un nouveau Pod



# Les pods peuvent se connecter au stockage partagé pour les services avec état

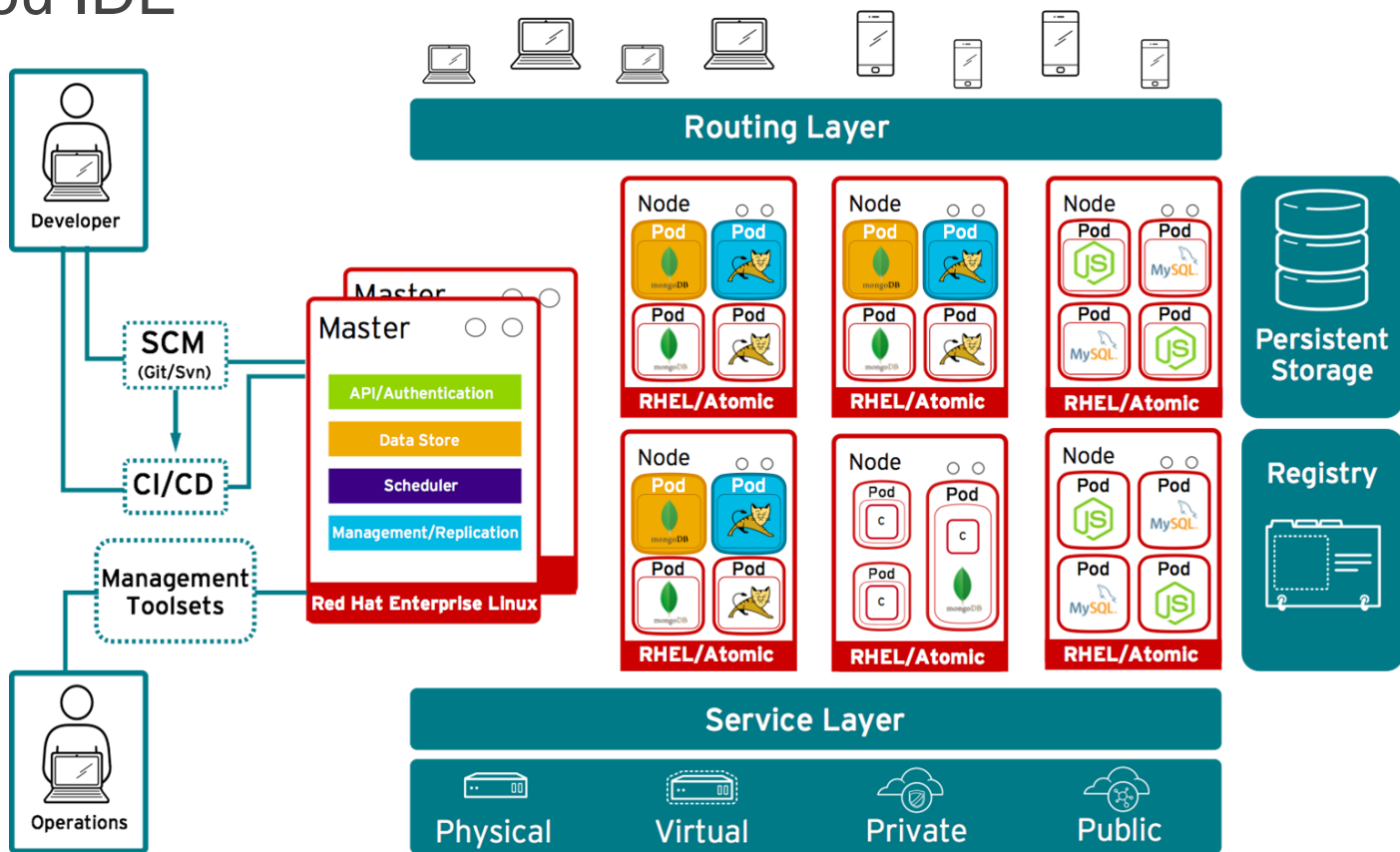


# La couche de routage achemine les demandes d'applications externes vers les pods





# Les développeurs accèdent à openShift via le Web, CLI ou IDE



# Les versions OpenShift ?

OpenShift ne se résume pas à un produit unique:

- « Red Hat OpenShift Container Platform » (OCP). Cette dernière est **installée sous la forme d'un cluster Kubernetes sur l'infrastructure Cloud hybride d'une entreprise.**
- **la version communautaire « OKD »**, qui portait autrefois le nom de « **OpenShift Origin** ».
- il existe également un large éventail de solutions « gérées »

# Qu'est-ce qu'un registre dans OpenShift ?

- Un registre de conteneur **contient des images de conteneurs qui sont créées en continu au fil du développement d'un logiciel.**
- Il est intéressant de noter qu'au sein d'OpenShift, le registre lui-même est implémenté en tant qu'opérateur.
- « Quay » constitue un **registre développé par Red Hat avec une attention particulière portée à la sécurité.**

« Quay constitue un registre de conteneur pour stocker des conteneurs, des charts Helm, et d'autres contenus relatifs aux conteneurs. » - Source :

<https://www.redhat.com/sysadmin/introduction-quay>

# Comment OpenShift est-il structuré ?

OpenShift est développé tout en haut de Kubernetes **en tant que cluster de conteneurs**. Au niveau du cluster, OpenShift comprend deux sous-niveaux :

- **Control Plane (régulateur)**: est composé de ce qu'on appelle des « control plane machines ». Ces dernières sont également connues sous le nom de « Master Machines » et gèrent le cluster de l'OpenShift Container Platform.
- **Worker Machines**: mènent à bien le travail effectif du cluster OpenShift. Les Master machines assignent des tâches aux worker machines et supervisent leur exécution.

Quels services  
sont exécutés sur  
les working  
machines ?

Une working machine exécute les services suivants et **fait donc partie du cluster OpenShift** :

- CRI-O, en tant qu'environnement d'exécution de conteneur,
- Kubelet, en tant que service qui accepte et procède les requêtes de lancer et d'interrompre des charges de travail,
- Un serveur proxy, qui prend en charge la communication entre les work machines.

# Les ressources dans OpenShift ?



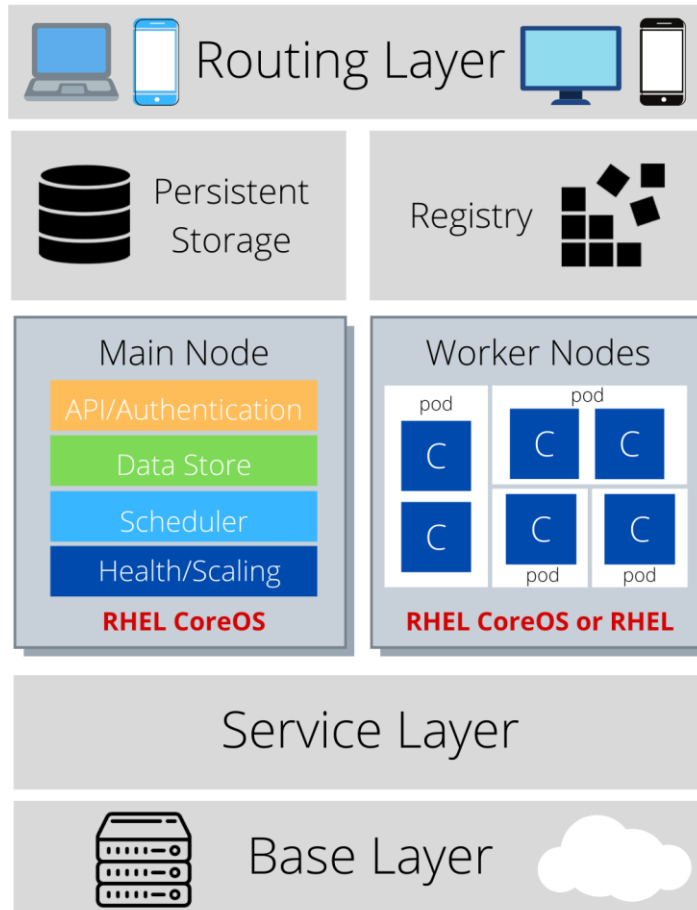
# Qu'est ce qu'Infra Node ?

Les nœuds d'infrastructure d'OpenShift sont des nœuds spécifiques dans un cluster OpenShift qui fournissent des fonctionnalités supplémentaires pour la plate-forme.

Voici quelques-unes des caractéristiques et des rôles principaux des nœuds d'infrastructure :

- **Routeurs**
- **Registres d'images**
- **Stockage Partagé**
- **Services réseau**
- **LoadBalancer**

# C'est quoi Infra Node ?





# Avantages d'OpenShift

OpenShift offre de nombreux avantages pour les entreprises qui souhaitent déployer et gérer leurs applications dans des environnements cloud :

- Simplification de la gestion des applications : OpenShift fournit une plateforme unifiée pour la gestion des applications, des conteneurs et des clusters Kubernetes, ce qui facilite grandement la tâche des développeurs et des équipes opérationnelles.
- Réduction des coûts : OpenShift utilise des technologies de conteneurisation pour optimiser l'utilisation des ressources informatiques, ce qui permet de réduire les coûts d'infrastructure et de matériel.
- Accélération du déploiement : OpenShift offre des outils de développement, d'intégration continue et de livraison continue (CI/CD) qui permettent de déployer rapidement des applications dans des environnements cloud.

# Avantages d'OpenShift

- **Flexibilité** : OpenShift peut être déployé sur site ou dans le cloud, ce qui offre une grande flexibilité aux entreprises pour choisir l'environnement qui convient le mieux à leurs besoins.
- **Sécurité renforcée** : OpenShift offre des fonctionnalités de sécurité avancées, telles que l'isolation des conteneurs, le chiffrement des données et la gestion des identités, pour assurer la sécurité des applications et des données.
- **Extensibilité** : OpenShift est une plateforme modulaire qui peut être étendue avec des plugins et des fonctionnalités supplémentaires pour répondre aux besoins spécifiques de chaque entreprise.

## Différents types d'installation

Openshift peut être installé de différentes manières :

- **OpenShift Container Platform (OCP)** : C'est la version complète d'OpenShift destinée aux environnements de production. Elle peut être installée sur des infrastructures sur site ou dans le cloud à l'aide de Red Hat OpenShift Installer ou des solutions de déploiement spécifiques à chaque fournisseur de cloud.
- **CodeReady Containers (CRC)** : Il s'agit d'une version légère d'OpenShift destinée au développement et aux tests locaux. CRC est installé sur une machine locale et offre une expérience OpenShift similaire.
- **OpenShift Origin** : Il s'agit de la version communautaire d'OpenShift. Elle est utilisée principalement pour tester ou contribuer au développement d'OpenShift.

# Exigences matériels

CodeReady Containers nécessite les ressources système suivantes :

- **4 CPU virtuels (vCPU)**
- **9 Go de mémoire libre**
- **35 Go d'espace de stockage**

# OpenShift : Console d'administration

The screenshot displays the OpenShift Container Platform console interface. The top navigation bar includes the Red Hat logo, the text "OpenShift Container Platform", and a user profile dropdown for "kube:admin". A blue banner at the top of the main content area states: "You are logged in as a temporary administrative user. Update the [cluster OAuth configuration](#) to allow others to log in."

The left sidebar contains a navigation menu with the following items: Home, Projects (selected), Status, Search, Events, Catalog, Workloads, Networking, Storage, Builds, Monitoring, Compute, and Administration.

The main content area is titled "Projects" and features a "Create Project" button and a search input field labeled "Filter Projects by name...". Below this is a table listing the projects in the cluster.

NAME ↑	STATUS	REQUESTER	LABELS
cloud-native-starter	Active	kube:admin	kiol.io/member-of=istio-system maistra.io/member-of=istio-system
default	Active	No requester	No labels
istio-system	Active	kube:admin	kiol.io/member-of=istio-system maistra.io/ignore-namespa...=ign... maistra.io/member-of=istio-system
kube-public	Active	No requester	No labels
kube-system	Active	No requester	No labels
openshift	Active	No requester	No labels
openshift-apiserver	Active	No requester	openshift.io/run-level=1
openshift-apiserver-operator	Active	No requester	openshift.io/cluster-monitori...=tr... openshift.io/run-level=0

Demo

Installation CRC

# Module 03:

## Conception d'applications conteneurisées

- ✓ Les principes KISS, DRY, YAGNI et SoC
- ✓ Les applications basées sur les microservices.
- ✓ Conception avancée de conteneurs

# KISS, DRY, YAGNI, SoC

Les principes KISS, DRY, YAGNI et SoC (sont des principes de conception et de développement logiciel qui visent à améliorer la qualité du code et à faciliter la maintenance.

- **KISS** (Keep It Simple, Stupid) encourage à garder les choses simples et éviter la complexité inutile. Il s'agit de privilégier des solutions simples et directes plutôt que des solutions compliquées.
- **DRY** (Don't Repeat Yourself) préconise d'éviter la duplication de code. L'idée est de ne pas répéter la même logique ou le même code à différents endroits, mais plutôt d'encapsuler cette logique dans des fonctions, des classes ou des modules réutilisables.



# KISS, DRY, YAGNI, SoC

- **YAGNI** (You Ain't Gonna Need It) conseille de ne pas ajouter des fonctionnalités ou du code dont on n'a pas réellement besoin. Il s'agit d'éviter de développer des fonctionnalités anticipées qui pourraient ne jamais être utilisées et d'attendre réellement le besoin avant de les implémenter.
- **SoC** (Separation of Concerns) préconise de séparer les différentes préoccupations ou responsabilités d'un système en modules distincts. Chaque module doit se concentrer sur une seule responsabilité et être indépendant des autres modules. Cela permet de faciliter la compréhension, la maintenance et la réutilisation du code.

# OpenShift ConfigMaps

OpenShift ConfigMap est un objet de configuration utilisé dans OpenShift pour stocker des données de configuration sous forme de paires clé-valeur.

Un ConfigMap est une ressource Kubernetes qui peut être utilisée pour fournir des configurations aux conteneurs s'exécutant dans les pods d'une application.

# Définition du ConfigMap

1 - Contient les données de configuration.

2 - Pointe vers un fichier contenant des données non UTF8.

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: default
data: ❶
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
binaryData:
  bar: L3Jvb3QvMTAw ❷
```

# OpenShift Secret

OpenShift Secret est une fonctionnalité de la plateforme OpenShift, développée par Red Hat, qui permet de gérer et de sécuriser les informations sensibles nécessaires aux applications déployées sur la plateforme.

## Définition du Secret

```
$ echo -n 'mon_mot_de_passe' > mysecret.txt
```

```
$ oc create secret generic mysecret --from-file=mysecret.txt
```

# Définition du Secret

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  template:
    spec:
      containers:
        - name: myapp
          image: myapp:latest
          env:
            - name: MY_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysecret
                  key: mysecret.txt
```

# Qu'est ce qu'un tag?

- La commande "***oc tag myimage:latest myimage:v2 -n myproject***" crée une nouvelle étiquette (tag) pour une image Docker nommée "myimage" dans le projet OpenShift nommé "myproject".
- Plus précisément, cette commande crée une nouvelle étiquette "v2" pour l'image Docker "myimage:latest". Cela permet de référencer l'image Docker "myimage:latest" avec un nouveau nom "myimage:v2".

Demo

Utilisation des tags pour les images



# OpenShift : les pods

- Un "pod" est l'unité de base pour le déploiement et l'exécution d'applications.
- Un pod représente un groupe de un ou plusieurs conteneurs étroitement liés qui partagent des ressources, tels que l'espace de stockage, l'adresse IP et les ports de communication..
- Un pod est généralement considéré comme une unité éphémère et remplaçable.
- Les conteneurs à l'intérieur d'un pod partagent un contexte d'exécution commun, ce qui signifie qu'ils peuvent communiquer entre eux via la communication inter-processus (IPC) ou en utilisant des sockets partagés.

# Pod : commandes utiles

- Obtenez la documentation intégrée pour les pods

**oc explain pod**

- Obtenir des détails sur les spécifications du pod

**oc explain pod.spec**

- Obtenir des détails sur les spécifications du pod

**oc explain pod.spec.containers**

- Créer un Pod sur OpenShift basé sur un fichier

**oc create -f pods/pod.yaml**

# Pod : commandes utiles

- Afficher tous les pods en cours d'exécution

**oc get pods**

- Ouvrir un port local qui transfère le trafic vers un pod

**oc port-forward <pod name> <local port>:<pod port>**

- Exemple de 8080 à 8080 pour hello world

**oc port-forward hello-world-pod 8080:8080**

- Shell dans les pods

**oc rsh <nom du module>**

- Regardez les mises à jour en direct des pods

**oc get pods --watch**

- Supprimer toute ressource OpenShift

**oc delete <resource type> <resource name>**

Demo

Création de pods dans OpenShift

# Module 04:

## Publication d'images de conteneurs d'entreprise

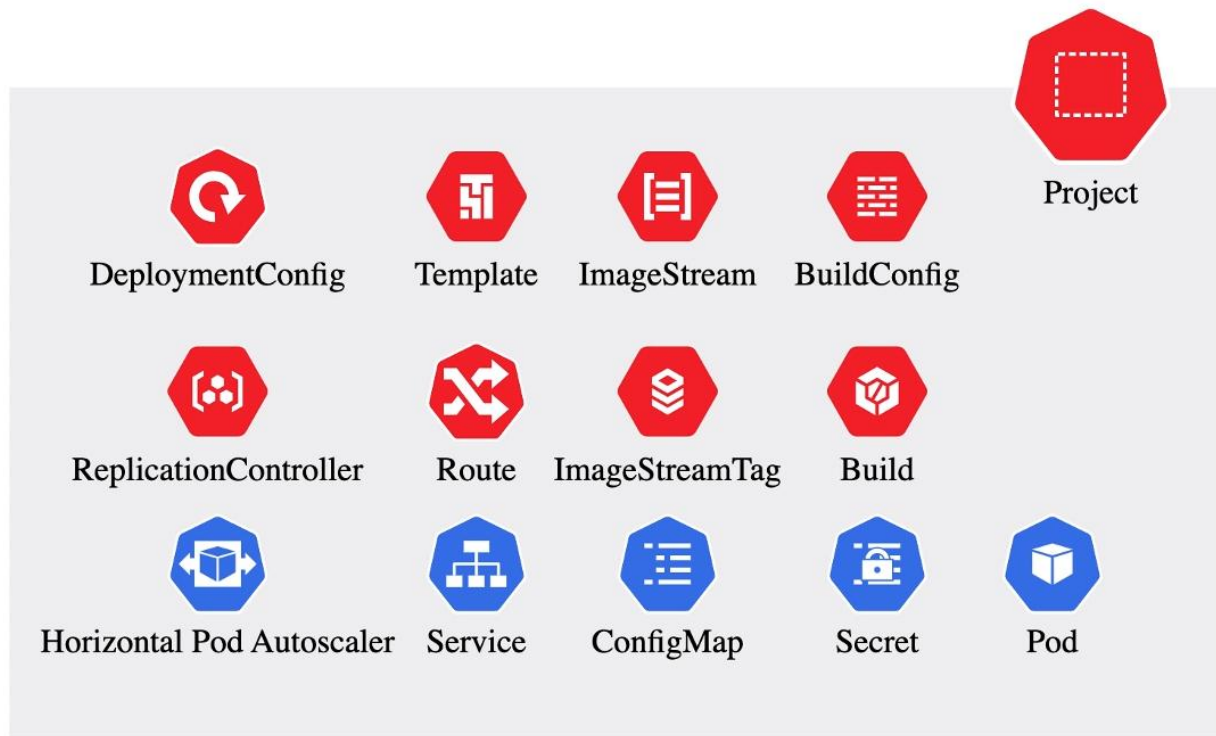


Notion de registre d'entreprise.



Les autorisations d'accès au registre OpenShift.

# Qu'est ce qu'un projet OpenShift?



# C'est quoi un projet OpenShift?

- Un projet est un espace de travail isolé où les applications peuvent être déployées et exécutées.
- Les projets sont utilisés pour organiser et isoler les applications déployées dans OpenShift, en offrant une isolation des ressources telles que les réseaux, les politiques de sécurité et les quotas.

# C'est quoi un projet OpenShift?

- Chaque projet possède ses propres ressources, telles que les images Docker, les applications, les services, les volumes, les secrets et les configurations.
- Les projets permettent également de gérer les utilisateurs et les autorisations, en offrant un contrôle d'accès précis aux différentes ressources du projet.
- `oc new-project <nom-du-projet>`



# Projet OpenShift

- # Voir le projet en cours

oc project

- # Créer un nouveau projet

oc new-project demo-project

- # Lister tous les projets

oc projects

- # Changer de projet

oc project <nom du projet>

Demo

Utilisation des projets dans OS

# Image container et imagestreams

- Une image contient un ensemble de logiciels prêts à être exécutés
- un conteneur est une instance en cours d'exécution d'une image de conteneur.
- Un flux d'images (imagestream) permet de stocker différentes versions de la même image de base. Ces différentes versions sont représentées par des balises (tags) différentes sur le même nom d'image.

# Qu'est ce qu'un objet ImageStream?

- **ImageStream** représente une liste de références d'images Docker. Cet objet peut être utilisé pour simplifier la gestion des images Docker dans le cluster OpenShift.
- **ImageStream** est utilisée pour stocker des métadonnées sur une ou plusieurs images Docker. Elle fournit un point de référence unique pour une image et peut être utilisée pour déployer une nouvelle version de l'image ou pour suivre les mises à jour de l'image.

# Qu'est ce qu'un objet ImageStream?

L'objet ImageStream inclut les informations suivantes :

- un nom et une description de l'image
- un tag pour identifier une version spécifique de l'image
- une référence à l'image Docker, qui peut être stockée dans un registre d'images Docker tel que Docker Hub, ou un registre d'images OpenShift interne

Qu'est ce qu'un objet ImageStream?



ImageStreamTag

New Images  
Trigger  
Deployments



DeploymentConfig

# Qu'est ce qu'un objet ImageStream?

Les images source peuvent être stockées dans l'un des éléments suivants :

- Registre intégré d'OpenShift Container Platform.
- Un registre externe, par exemple [register.redhat.io](https://register.redhat.io) ou [Quay.io](https://quay.io).
- Autres flux d'images dans le cluster OpenShift Container Platform.

Demo

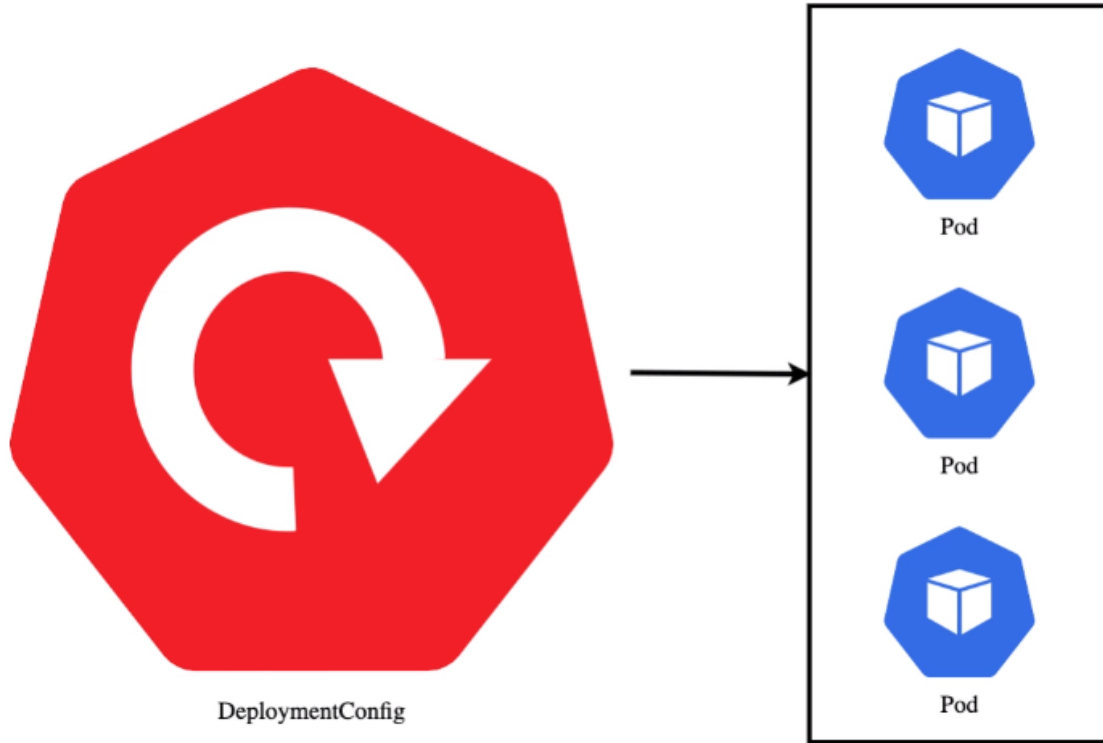
Créer une application Openshift à partir d'une image



# DeploymentConfig

- DeploymentConfig est un objet spécifique à la plateforme OpenShift qui permet de déployer et de gérer des applications dans un cluster OpenShift. Il s'agit d'une abstraction autour de l'objet Kubernetes Deployment.
- Un DeploymentConfig définit la configuration et les paramètres de déploiement d'une application, notamment le nombre de répliques (instances) de l'application à exécuter, les stratégies de déploiement (comme le déploiement progressif ou le déploiement instantané), les mises à jour automatiques, etc.

# DeploymentConfig



# Module 05:

## Build d'applications depuis OpenShift



Description du processus de construction OpenShift (Build, BuildConfig, DeploymentConfig)



Mise en œuvre de crochets de version après soumission.

# Objet BuildConfig

- BuildConfig est un objet Kubernetes qui représente un processus de construction d'une image Docker.
- BuildConfig peut être créé à partir d'une image Docker de base ou à partir d'un code source et peut inclure des étapes de construction personnalisées qui permettent de générer l'image Docker finale.
- L'objectif principal d'un objet BuildConfig est de permettre aux développeurs de construire des images Docker de manière automatisée et de les intégrer dans un processus de déploiement continu (CI/CD).
- BuildConfig représente un processus de construction automatisé d'une image Docker à partir d'une image de base ou d'un code source, et est utilisé pour intégrer la construction d'images Docker dans un processus de déploiement continu.

# BuildConfig

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: nodejs-sample
spec:
  source:
    git:
      uri: https://github.com/sclorg/nodejs-ex.git
      ref: master
  strategy:
    type: Source
    sourceStrategy:
      from:
        kind: ImageStreamTag
        namespace: openshift
        name: nodejs:14-ubi8
  output:
    to:
      kind: ImageStreamTag
      name: nodejs-sample:latest
  triggers:
    - type: ConfigChange
    - type: ImageChange
```

# Objet BuildConfig

`apiVersion: build.openshift.io/v1`

On utilise l'API OpenShift pour la gestion des builds.

`kind: BuildConfig`

C'est un objet de type **BuildConfig**.

`metadata.name: nodejs-sample`

Le nom donné à cette BuildConfig est `nodejs-sample` .

`source.git.uri`

OpenShift va récupérer le code depuis ce dépôt Git : `https://github.com/sclorg/nodejs-ex.git` .

`source.git.ref`

La branche utilisée est `master` .

# Objet BuildConfig

`type: Source`

Type de build : **S2I** (*Source-to-Image*)

`sourceStrategy.from.kind:`

On utilise une image existante comme base (via ImageStream).

`ImageStreamTag`

`sourceStrategy.from.namespace:`

L'image de base est dans le projet `openshift` .

`openshift`

`sourceStrategy.from.name: nodejs:14-`

On utilise Node.js version 14 basé sur UBI8 (Universal Base Image Red Hat).

`ubi8`

# Objet BuildConfig

`output.to.kind: ImageStreamTag`

Le résultat sera stocké sous forme d'une **ImageStreamTag**.

`output.to.name: nodejs-sample:latest`

Le nom de l'image générée sera `nodejs-sample:latest` .

`type: ConfigChange`

Rebuild automatique si la BuildConfig est modifiée.

`type: ImageChange`

Rebuild automatique si l'image de base ( `nodejs:14-ubi8` ) change.



# Objet BuildConfig

Ce BuildConfig :

- Clone un projet Node.js depuis GitHub
- Utilise une image Node.js officielle (nodejs:14-ubi8)
- Construit une nouvelle image conteneur avec ton application dedans
- Stocke l'image dans OpenShift sous nodejs-sample:latest
- Déclenche automatiquement des builds si le code ou l'image de base changent.

# Objet BuildConfig

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: maven-webapp-build
spec:
  runPolicy: <... list the run policy ...>
  triggers:
    <... list of triggers ...>
  source:
    <... input parameters or source ...>
  strategy:
    <... build strategy to use ...>
  output:
    <... repository details ...>
  postCommit:
    <... optional build hooks ...>
```

**triggers** et **postcommit** sont optionnels.

## Objet BuildConfig

- **triggers** - peut être utilisé pour définir la manière dont cette configuration peut être déclenchée. Des webhooks comme les webhooks GitHub/GitLab peuvent être utilisés pour déclencher ce build.
- **source** - sous cette section, les paramètres d'entrée et le référentiel de code source sont définis.
- **strategy** - définir la manière dont Openshift doit construire l'image
- **output** - Cette directive est utilisée pour définir un référentiel de sortie d'image. Il peut s'agir d'une version Openshift dans un référentiel privé, d'un référentiel d'images privé hébergé sur site ou du hub Docker.

# DeploymentConfig

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: mon-deployment
spec:
  replicas: 3
  selector:
    app: mon-application
  template:
    metadata:
      labels:
        app: mon-application
    spec:
      containers:
        - name: mon-container
          image: mon-image:latest
          ports:
            - containerPort: 8080
```

Demo

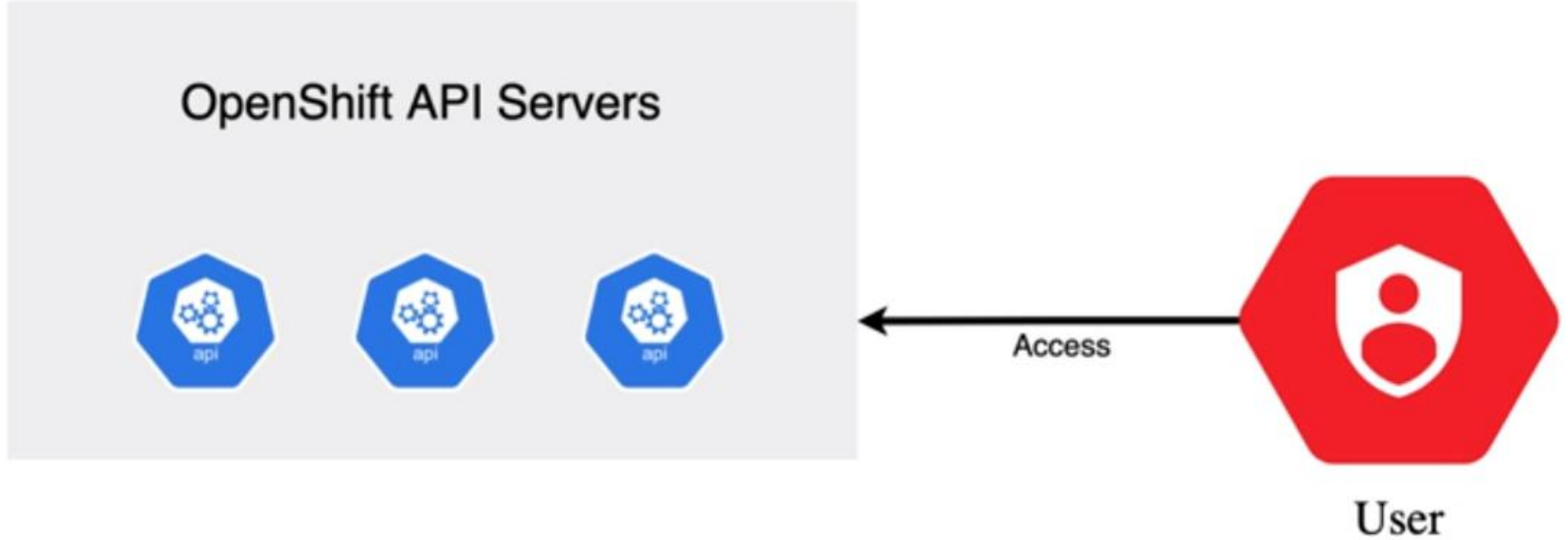
Créer un objet build

# Module 06:

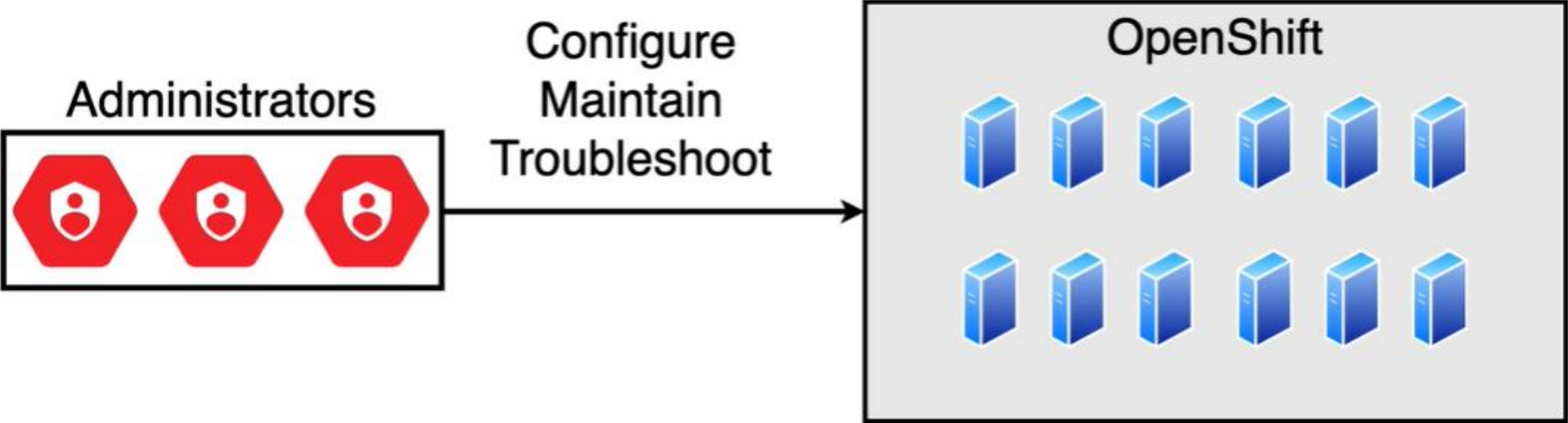
## Personnalisation de versions source-to-image

- ✓ Pourquoi personnaliser une version S2I (source-to-image).
- ✓ Personnalisation d'une image S2I

# Les utilisateurs dans Openshift

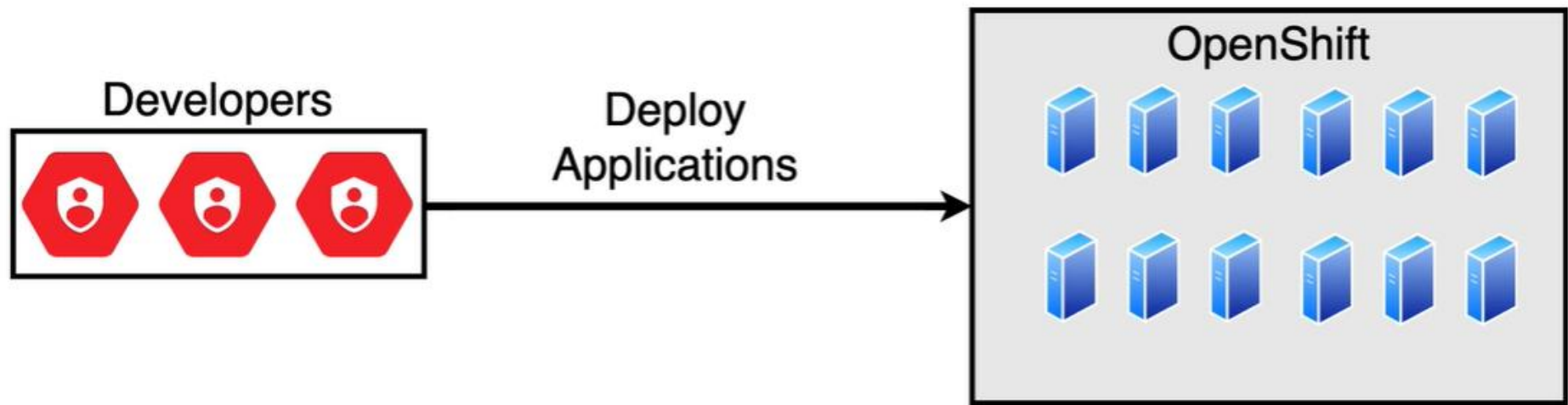


# Les utilisateurs dans Openshift : Les administrateurs

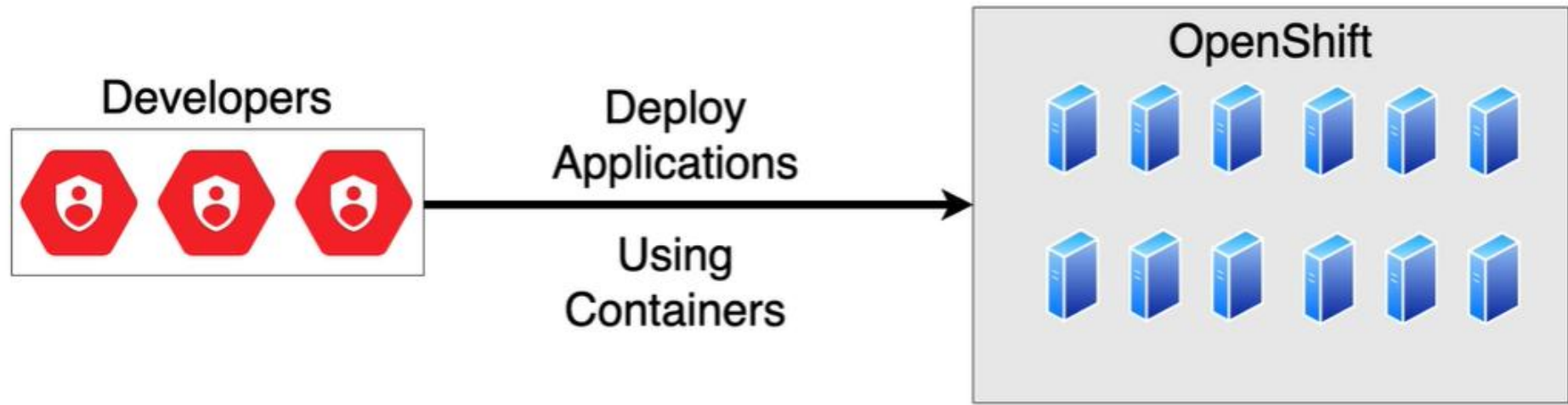




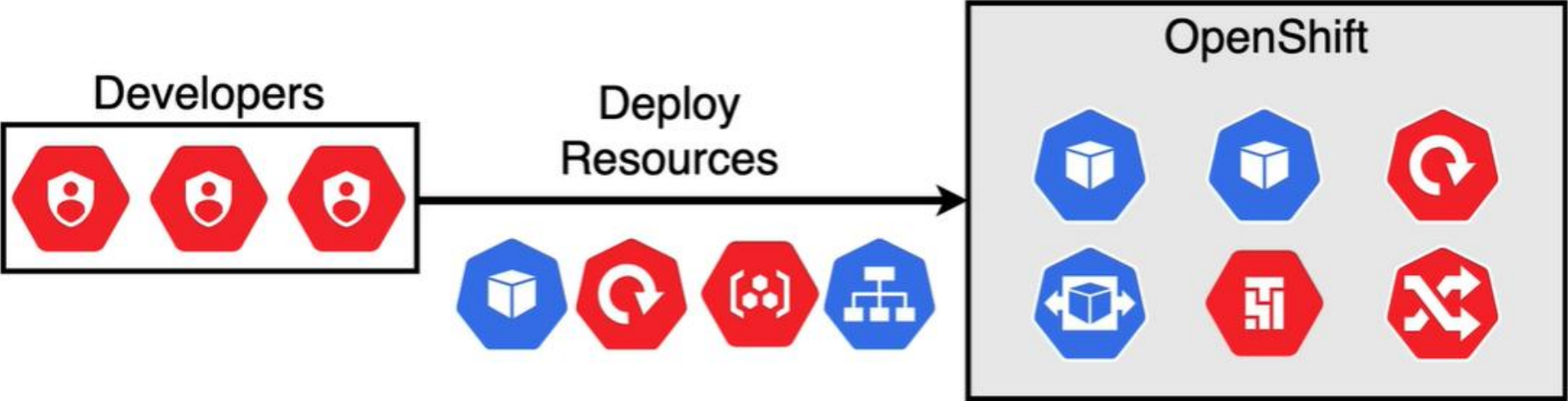
# Les utilisateurs dans Openshift : les developpeurs



# Les utilisateurs dans Openshift : les developpeurs



# Les utilisateurs dans Openshift : les developpeurs



# Les utilisateurs dans Openshift : les developpeurs



User



DeploymentConfig



Build



ImageStream



Route



Project



ReplicationController



BuildConfig



ImageStreamTag



Template



Pod



ConfigMap



Secret

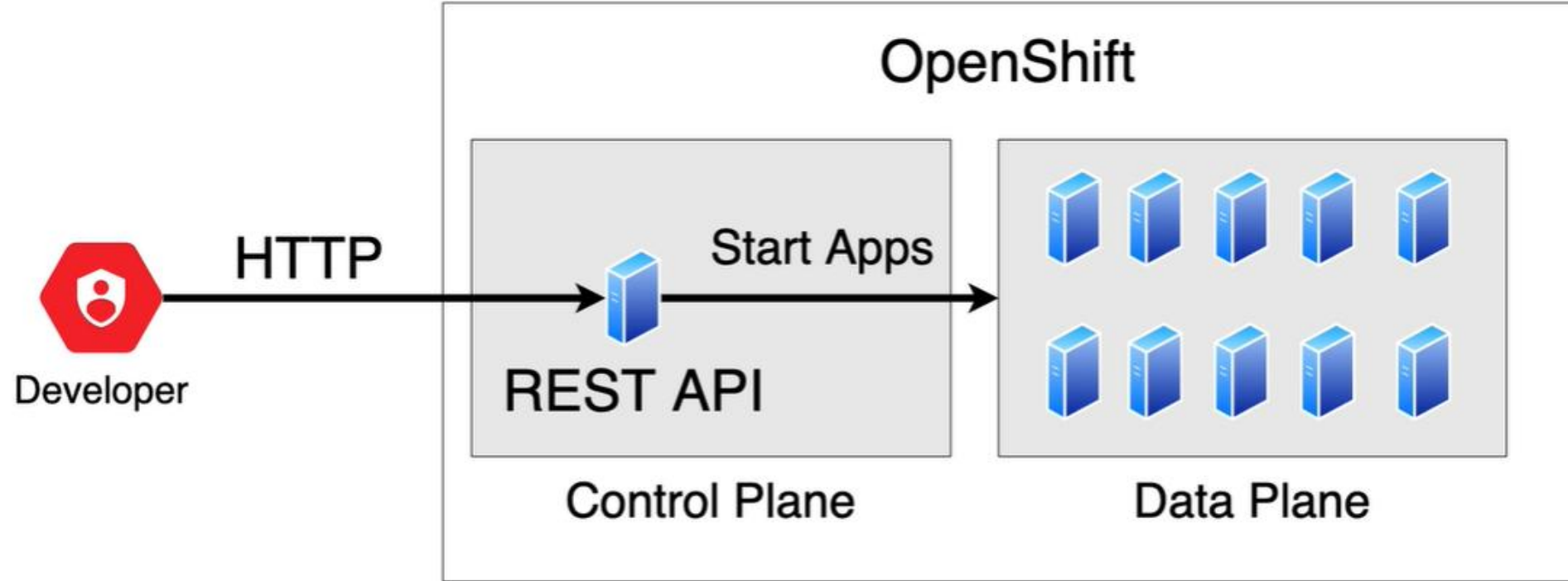


Service



Autoscaler (HPA)

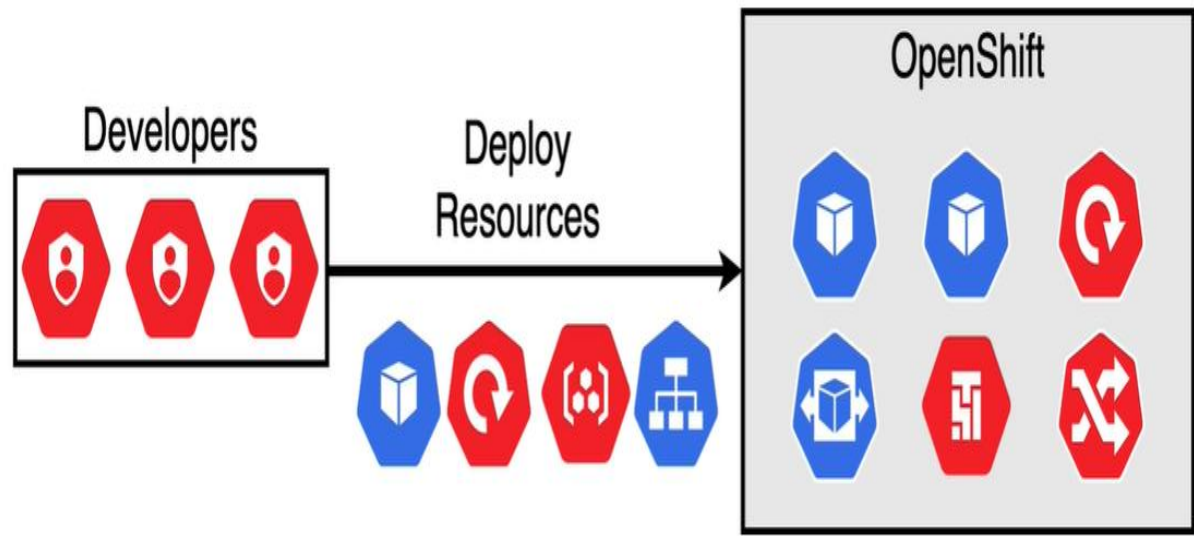
# Les utilisateurs dans Openshift : les developpeurs



# Les utilisateurs dans Openshift

- Un utilisateur est une entité qui a accès et interagit avec la plateforme OpenShift.
- Un utilisateur peut être une personne ou un système automatisé.
- Chaque utilisateur d'OpenShift dispose d'un compte avec des identifiants ou des clés d'accès, qui lui permettent de se connecter à OpenShift et d'effectuer certaines actions autorisées en fonction de ses permissions.
- Les utilisateurs dans OpenShift sont associés à des rôles et des autorisations.
- Les rôles déterminent les actions spécifiques qu'un utilisateur peut effectuer, tandis que les autorisations contrôlent l'accès de l'utilisateur à certaines ressources, telles que les projets, les applications et les services

# Source-to-Image S2I



# S2I (Source-to-Image) dans OpenShift

Fonctionnalités principales et cycle d'utilisation



# Qu'est-ce que S2I ?

- S2I (Source-to-Image) est un outil intégré à OpenShift qui permet de construire automatiquement une image exécutable à partir du code source d'une application, sans nécessiter de Dockerfile.

# Fonctionnalités principales de S2I

- Construction d'images à partir du code source
- Utilisation d'images builder préconfigurées (Java, Node.js, etc.)
- Automatisation du processus de build
- Intégration dans les pipelines Tekton

# Cycle typique d'utilisation de S2I

1. Le développeur pousse du code sur GitHub
2. OpenShift détecte le changement (webhook)
3. Le pipeline déclenche un build S2I
4. Une image exécutable est construite et déployée automatiquement

# Source-to-Image S2I

- S2I Builder est un outil de développement inclus dans OpenShift qui permet de créer des images Docker à partir du code source d'une application.
- S2I simplifie le processus de développement et de déploiement d'applications dans des conteneurs, car il n'est plus nécessaire de créer manuellement une image Docker pour chaque application.
- Avec S2I Builder, les développeurs peuvent créer des images Docker à partir de leur code source simplement en poussant le code source vers le référentiel Git de l'application.
- S2I Builder va ensuite compiler le code source, créer une image Docker et pousser cette image vers un référentiel d'images. Cela permet aux développeurs de se concentrer sur la création d'applications, plutôt que sur la gestion de l'infrastructure.

# Source-to-Image S2I

- S2I Builder est extensible et peut être utilisé avec plusieurs langages de programmation, tels que Java, Python, Ruby, Go et Node.js. De plus,
- S2I Builder prend en charge les environnements multi-stades, ce qui permet de déployer des applications dans des environnements de production à partir de différents environnements de développement et de test.

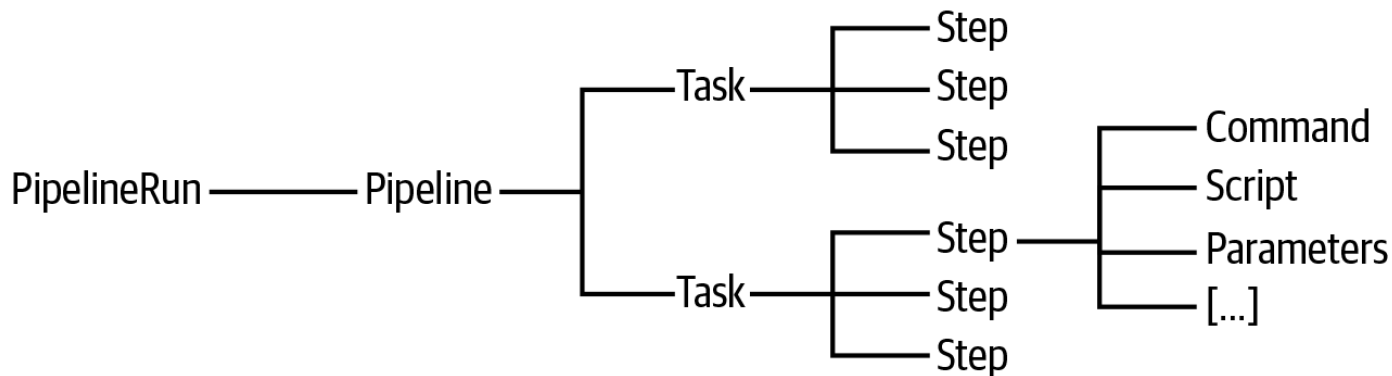
Demo

Déployer une application à partir du code source

# Introduction aux Pipelines dans OpenShift

Automatisation CI/CD avec Tekton

- Un pipeline OpenShift est un enchaînement d'étapes
- automatisées pour construire, tester et déployer
- des applications dans un cluster OpenShift.
- Basé sur Tekton (open source).





- Le pipeline décrit :
  - - Les étapes à exécuter (Tasks)
  - - L'ordre d'exécution (runAfter)
  - - Les paramètres et workspaces utilisés
- Exécuté via une instance : PipelineRun.

- Phases typiques :
- 1. Fetch (git-clone)
- 2. Build (s2i-java, buildah)
- 3. Test (facultatif)
- 4. Deploy (openshift-client)
- 5. Vérifications post-déploiement

- Avantages :
- - Automatisation complète
- - Intégration native avec OpenShift
- - Flexibilité totale
- - Séparation Dev et Ops
- - Observabilité améliorée

- Les pipelines OpenShift créent des chaînes de déploiement
- modernes, robustes et cloud-native,
- s'appuyant sur Kubernetes et Tekton.

# Module 07:

## Conception d'applications à partir de modèles dans OpenShift

- ✓ Notion de modèle OpenShift
- ✓ Modèle multi-conteneurs

# Les Modèles dans Openshift

- Un "template" désigne un modèle ou un schéma préétabli utilisé comme base pour créer des instances ou des copies d'une structure spécifique.
- En général un fichier de configuration décrivant la spécification d'une application ou d'un ensemble d'objets liés, tels que des pods, des services, des volumes, des routes, etc.
- Les templates sont écrits dans un format spécifique (généralement YAML ou JSON) et peuvent être utilisés pour déployer et répliquer facilement des applications complexes.
- Un template permet de définir les caractéristiques, les paramètres et les relations entre les différents composants de l'application. Il peut inclure des informations telles que les images de conteneur à utiliser, les ports d'écoute, les variables d'environnement, les ressources requises, les dépendances, etc.

# Qu'est-ce qu'un Template ?

- Un fichier YAML ou JSON regroupant plusieurs objets Kubernetes/OpenShift
- Permet le déploiement d'une application complète en une seule commande
- Utilise des paramètres pour personnaliser facilement

# Que contient un Template ?

- Objets : Deployment, Service, Route, BuildConfig...
- Paramètres personnalisables (APP\_NAME, PORT...)
- Valeurs par défaut pour simplifier le déploiement



# Exemple pratique : Node.js Template

- Deployment (application Node.js)
- Service (port 8080)
- Route (expose l'application sur Internet)
- Paramètre : APP\_NAME

# Comment utiliser un Template ?

1. Sauvegarder le template en .yaml

2. Commande :

```
oc process -f template.yaml | oc apply -f -
```

- Automatisation complète
- Paramétrage facile

# Avantages des Templates

- Déploiement rapide d'applications
- Réutilisation simple dans CI/CD et GitOps
- Personnalisation dynamique avec paramètres
- Simplification de la standardisation des déploiements

Demo

Création d'un template custom

# Module 08:

## Gestion de déploiement d'applications

- ✓ Monitoring des applications déployées
- ✓ Stratégie de déploiement adaptée au monitoring planifié.

# Déployer une application

Il est possible de créer et de déployer à partir de différents types de sources :

- images Docker
  - **oc new-app nginx:latest**
- Fichiers de configurations yaml
  - **oc new-app -f chemin/vers/fichier.yaml**
- À partir d'un référentiel git
  - **oc new-app https://github.com/votre-utilisateur/votre-repo.git**

# Stratégie personnalisée OpenShift - OpenShift Custom Strategy

Stratégie personnalisée OpenShift est une méthode utilisée dans la plateforme de conteneurisation OpenShift pour déployer et gérer des applications. OpenShift est basé sur Kubernetes et fournit des fonctionnalités supplémentaires pour faciliter le déploiement, la gestion et l'évolutivité des applications conteneurisées.

# Route

- OpenShift Route est objet de configuration qui permet de gérer l'accès externe aux services déployés sur un cluster OpenShift.
- En utilisant les Routes, vous pouvez exposer les applications et les services aux utilisateurs finaux via des URL accessibles depuis l'extérieur du cluster.
- Une Route OpenShift agit comme un équilibreur de charge et un routeur de trafic pour les services. Elle permet de faire correspondre une URL publique à un service interne du cluster, en redirigeant le trafic entrant vers les pods appropriés.
- Les Routes peuvent également fournir des fonctionnalités avancées telles que le chiffrement SSL/TLS, l'équilibrage de charge basé sur des règles, la gestion du trafic en fonction de différents chemins d'URL, etc.
- Les Routes dans OpenShift sont utilisées pour exposer les services et les applications de manière sécurisée et contrôlée aux utilisateurs externes.



# Route



User



DeploymentConfig



Build



ImageStream



Route



Project



ReplicationController



BuildConfig



ImageStreamTag



Template



Pod



ConfigMap



Secret

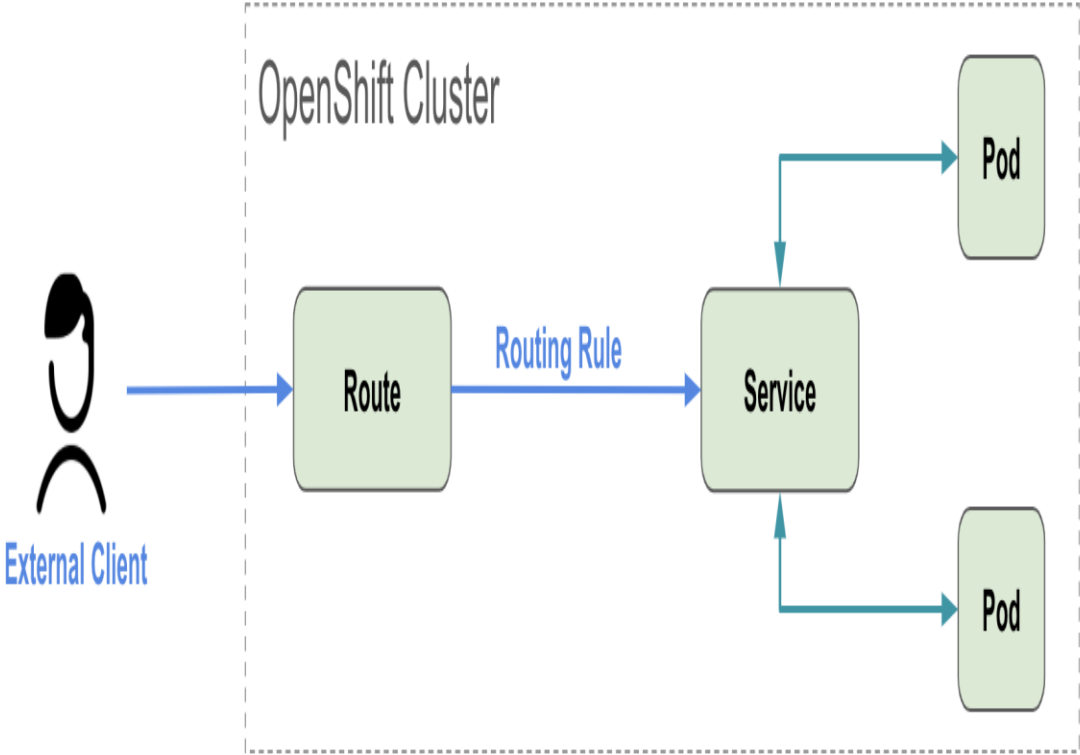


Service



Autoscaler (HPA)

# Route



# Routes:

La commande "**oc expose svc/dashboard**" permet d'exposer le service.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: exemple-route
spec:
  to:
    kind: Service
    name: exemple-service
  port:
    targetPort: 8080
  tls:
    termination: edge
    insecureEdgeTerminationPolicy: Redirect
```

La commande "**oc get routes**" Confirm that a new route has been registered for the **dashboard** service..

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: mon-route
spec:
  to:
    kind: Service
    name: mon-service
  port:
    targetPort: 8080
  tls:
    termination: edge
    insecureEdgeTerminationPolicy: Redirect
```

Demo

Exposer une application à l'extérieur