



# THALES - MDS Deployable

Workshop - OpenShift RHCOS et Namespace

---

Luigi Colagiovio

Architect OpenShift

# Document Information

CONFIDENTIAL Designator

## **Originator**

Red Hat Consulting

## **Owner**

Red Hat Consulting - Confidential; Restricted Distribution

## **Distribution**

Do not forward or copy without written permission.

## **Confidentiality**

This is a confidential document between Red Hat, Inc. and Customer (“Client”).

All information supplied to the Client for the purpose of this project is to be considered Red Hat confidential.

## **Disclaimer**

This document is not a quote, and does not represent an official Statement of Work. If acceptable, a formal quote can be issued, which will include a contract with the scope of work, cost, and any Client requirements if necessary.

# Contacts Red Hat

CONFIDENTIAL Designator

Name	Role	Mail	Phone
Luigi Colagiorgio	Architect OpenShift	lcolagio@redhat.com	07 70 21 98 73

# Contacts THALES MDS

CONFIDENTIAL Designator

## Contacts présents au workshop

Name	Role	Mail	Mobile
Sylvie GALLOUX	Scrum Master	sylvie.galloux@thalesgroup.com	
Boris HOUÉE	Product Owner	boris.houee@thalesgroup.com	
Pierre Novat	Architecte MDS	pierre.novat@thalesgroup.com	
Stephane Solier	Architecte Sécurité	stephane.solier@thalesgroup.com	

# Rappel des Demandes / Questions

CONFIDENTIAL Designator

## Questions ?

- Quel est l'implémentation de CRI-O sur RHCOS
- Comment fonctionne les machines config / Upgrade RHCOS

# Goals and Agenda

CONFIDENTIAL Designator

## Goals

- Comprendre les solutions techniques mise en oeuvre pour le cloisonnement des containers
  - dans Red Hat CoreOS (RHCOS)
  - dans les projets Openshift

## Agenda

- RH CoreOS
  - CoreOS Sécurité
  - Container Sécurité (CRIO)
  - Machines config
- Projets OpenShift
  - Projets et Namespaces Kubernetes
  - Cloisonnement ressource compute (Quota / cluster Quota / LimitRanges)
  - Cloisonnement réseau par network policy
  - Cloisennement ressource par node
- Annexes
  - OpenShift Concepts

# Sources documentaires

CONFIDENTIAL Designator

- [https://access.redhat.com/system/files/redhat\\_openshift\\_security\\_guide.pdf](https://access.redhat.com/system/files/redhat_openshift_security_guide.pdf)





# Red Hat Enterprise Linux CoreOS

# CoreOS Sécurité

The OpenShift operating  
system

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- **Basé sur RHEL:**
  - Système d'exploitation sous-jacent se compose principalement de composants RHEL.
  - Mêmes niveau de qualité, de sécurité et de contrôle que RHEL
- **Immutable:**
  - RHCOS est conçu pour être géré de manière plus spécifique qu'une installation RHEL par défaut.
  - La gestion est effectuée à distance depuis le cluster OpenShift Clusters
  - OpenShift stocke le dernier état des RHCOS dans le cluster, de sorte qu'il est toujours en mesure de créer et effectuer des mises à jour sur la base des dernières Configurations RHCOS.
- **CRI-O container runtime:**
  - RHCOS contient des fonctionnalités permettant d'utiliser les formats OCI et libcontainer des Containers
  - Il intègre le moteur de Container CRI-O au lieu du moteur de Container Docker.
  - CRI-O offre une empreinte et surface d'attaque plus petite.
  - CRI-O n'est disponible que sous forme de moteur de Container dans les clusters OpenShift.

# CoreOS - Points clefs / Philosophie

- **Ensemble d'outils de Container en ligne de commande :**

- RHCOS remplace Docker par un ensemble compatible d'outils compatible pour les Containers.
- La commande **podman** pour l'exécution, le démarrage, l'arrêt, la liste et la suppression des Containers et des images de Containers.
- La commande **skopeo** pour copier, authentifier et signer les images.
- La commande **crictl** pour interagir avec les Containers et les pods exécutés avec le moteur de Containers du CRI-O. Bien que l'utilisation directe de ces dans RHCOS est découragé, ils peuvent être utilisés pour le débogage des fins.

- **mises à jour de rpm-ostree:**

- RHCOS propose des mises à jour transactionnelles utilisant le système rpm-ostree.
- Les mises à jour sont livrées au moyen d'images de Containers et font partie du Processus de mise à jour du cluster OpenShift.
- L'image du Container déployé est extraite et écrite sur le disque, puis le bootloader est modifié pour démarrer la nouvelle version.
- Une fois la machine redémarrée, la mise à jour est réalisée de manière continue pour s'assurer que la capacité du clusters est impactée de façon minimale.

- **Mise à jour via Machine Config Operator:**

- Dans OCP, le Machine Config Operator s'occupe les mises à jour du système d'exploitation.  
La commande rpm-ostree permet la mise à niveau du système d'exploitation en tant qu'unité atomique au lieu de mettre à jour des paquets individuels, comme c'est le cas avec Yum,
- L'upgrade du système d'exploitation est réalisé lors des mises à niveau OCP et prend effet au prochain redémarrage.  
En cas de problème lors de la mise à niveau, un retour en arrière avec un redémarrage remet le système à l'état antérieur.

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- **Tout est container**

- Pour les systèmes **RHCOS**, la structure du système de fichiers **rpm-ostree** présente les caractéristiques suivantes :
  - **/usr** est l'endroit où sont stockés les binaires et les bibliothèques du système d'exploitation et est en lecture seule. Red Hat n'est pas favorable à la modification dans ce fs
  - **/etc, /boot, /var** sont modifiable sur le système mais ne sont destinés qu'à être modifiés par le **Machine Config Operator**
- Les fichiers écrits sur ces chemins sont persistants.
  - **/usr/local** est un lien symbolique vers **/var/usr/local**. Les logiciels peuvent être placé dans **/usr/local/{bin,sbin}**. Selon les paramètres des chemins, il se peut que les RHCOS n'utilisent pas ces chemins. Il n'est donc pas recommandé aux administrateurs de se fier à ces chemins.
  - La configuration du bootloader est gérée par **rpm-ostree**.
  - Le **Machine Config Operator** peut définir les paramètres du noyau pour les **machines pool config**.
  - Les différents chemins sont des liens symboliques vers **/var** :
    - **/root** -> **/var/roothome**
    - **/opt** -> **/var/opt**
    - **/srv** -> **/var/srv**
  - **/var/lib/containers** est le chemin pour stockager les images de container



# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- **RHCOS**

- Il n'est pas destiné à être géré en dehors d'un cluster OpenShift.
  - Conceptuellement, **RHCOS** est la couche de base de la plate-forme de Containers OpenShift 4.
  - En tant qu'élément du cluster, **RHCOS** est géré et mis à jour par le cluster.
  - Pour faire fonctionner des logiciels, les administrateurs de **RHCOS** doivent utiliser des Containers.
  - En raison de la configuration unique du système de fichiers, des mises à jour gérées et atomiques, RHCOS est incompatible au niveau de l'hôte avec la plupart des applications non Containerisées.

- **Immutable:**

- Un système d'exploitation **immutable** est un système dans lequel l'état ne persiste pas entre deux redémarrages. RHCOS n'est pas un système d'exploitation **immutable** car l'état peut persister.
- La commande **rpm-ostree** peut être utilisée pour remplacer et modifier l'image **ostree** sous-jacente.
- RHCOS possède des éléments **d'immutabilité** qui sont contrôlés par le **cluster OpenShift**. Cette immutabilité contrôlée fait référence au contenu du système d'exploitation de la machine qui est fourni par Red Hat et qui comprend le noyau, les initramfs, les paquets et la configuration par défaut. Le **Machine Config Operator** veille à ce que chaque nœud maintienne la configuration requise du point de vue des clusters.

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- **Ignition:**

- Le premier agent d'amorçage de **RHCOS** est **Ignition**.

*Extrait de la page Github d'ignition :*

- **Ignition** est l'utilitaire utilisé par **CoreOS Container Linux**, **Fedora CoreOS**, et **RHEL CoreOS** pour manipuler les disques lors des **initramfs**.
- Cela comprend le partitionnement des disques, le formatage des partitions, l'écriture des fichiers (c'est-à-dire les fichiers ordinaires, **systemd**), et la configuration des utilisateurs.
- Au premier démarrage, **Ignition** lit sa configuration à partir d'une source de vérité (**URL distante**, **métadonnées réseau service**, bridge d'hyperviseur, etc.) et applique la configuration.
- Lors de l'installation, **l'installateur** crée dynamiquement des profils **d'ignition** pour configurer et installer **RHCOS** et **OpenShift**.
- lors que les fichiers de configuration **d'ignition** peuvent être utilisé pour modifier le comportement du **bootstrap**, il ne doit être fait que lorsque cela est nécessaire, comme par exemple pour activer du matériel ou définir des paramètres ciblés.

# CoreOS - Points clefs / Philosophie

- **Machine Config Operator:**
  - OpenShift 4 est une plate-forme pilotée par les **opérateurs**.  
Cela permet:
    - à l'opérateurs **Machine Config Operator (MCO)** d'adopter une approche déclarative de la gestion du cycle de vie des composants des clusters. On appelle cela des **machines config**
    - de gérer automatiquement les mises à jour des clusters qui vont du noyau aux services situés plus haut dans les couches (par exemple ntp).
  - Le **MCO** est le composant qui relie **RHCOS** et le **cluster OpenShift**.
    - Il fournit les ressources **Machine Config** qui permettent d'acheminer la configuration du système d'exploitation à travers le cluster:  
Par exemple:
      - pour des options de configuration spécifiques comme le **kernelType** pour activer le noyau en temps réel ou **kernelArguments**.
      - pour des options de configuration du **runtime du Container** et du **kubelet** avec leurs objets respectifs, **ContainerRuntimeConfigs** et **KubeletConfigs**, qui sont traduits en **MachineConfiguration** par des contrôleurs dédiés.
      - pour définir des configurations utilisateurs à base de fichiers, des systemd units qui suivent le format de configuration CoreOS Ignition.
    - Les **Machine Config** peuvent être ajoutés ou modifiés sur les nœuds existants grâce aux objets MachineConfigs.
    - L'objectif du **MCO** et des **machines configs** sont de rendre la configuration de **RHCOS** cohérente sur l'ensemble des nœuds du cluster.

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- **Un mot sur les agents**

- De nombreux logiciels ont des agents qui exécutent une pléthore d'actions. Des fonctions telles que gestion de la configuration, gestion de l'état, audit, authentification, le contrôle d'accès logique, et la surveillance sont normalement présumés être disponibles pour un système Linux.
- L'utilisation d'agents dans **RHCOS** est considéré comme un **anti-modèle** dans l'architecture de sécurité **RHCOS**. **RHCOS** répond à une stratégie de type "**appliance**" en terme de gestion d'OS.
- De nombreux agents s'attendent à une représentation traditionnelle du système de fichiers et ne peuvent pas fonctionner sur RHCOS ; c'est vrai même des agents qui fonctionnent sur RHEL 8. Par exemple, l'emplacement du module noyau sous RHCOS est en lecture seule.

**Lorsqu'une installation d'agent est nécessaire, elle doit être Containerisé.**

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

## Modification des paramètres de sécurité RHCOS

- Comme Red Hat OpenShift 4 a été conçu pour être simple à installer et sécuriser par défaut (out of the box), un des principaux objectifs était de verrouiller les **workers** et **contrôle plane** (masters). Les nœuds qui ont dévié des paramètres par défaut risquent de créer des vulnérabilités de sécurité et de devenir difficiles à mettre à niveau.

En conséquence, OpenShift 4:

- **Décourage la modification directe des RHCOS :**

La pratique classique des administrateurs Linux qui consiste à se connecter à un nœud et à ajouter des logiciels ou à modifier des fichiers de configuration n'est plus adaptée.

D'autant que les modifications peuvent être écrasées par les mises à jour ou les opérateurs.

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- **Encourage la modification des nœuds via des pools :**
  - Tous les nœuds **workers** et **control plane** doivent être configurés de la même manière.
  - OpenShift offre des moyens d'appliquer les changements à tous les nœuds **similaires** en même temps en utilisant des opérateurs (MCO).
  - L'ajout de modifications via des pools seront également appliqués à de nouveaux nœuds.
- Il arrive que les exigences de sécurité exigent que des modifications soient apportées aux systèmes RHCOS dans les clusters OpenShift 4.
  - En règle générale, toute modification de sécurité doit être réduite au minimum et être mise en œuvre au niveau des clusters, dans la mesure du possible.
  - En gardant cette règle à l'esprit, il existe plusieurs points d'entrée où des modifications des nœuds **workers** et **control plane** d'OpenShift 4 peuvent être envisagées.
- Modifications de sécurité au moment de l'installation.
  - Pendant l'installation, RHCOS offre la possibilité d'ajouter des paramètres de sécurité qui doivent être en place avant le démarrage du cluster ou immédiatement après le premier démarrage.
  - Certains de ces paramètres ne sont actuellement pris en charge que lors de l'installation d'OpenShift 4 en mode bare metal, tandis que d'autres peuvent être effectués lors d'installations où un fournisseur de cloud ou un autre environnement fournit l'infrastructure.

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- **Installation bare metal :**
  - L'installation bare metal est considérée comme plus laborieuse que les installations basées sur le cloud.
  - Dans le cas des installations dans le cloud, l'installateur OpenShift peut diriger l'installation à l'aide d'API pour fournir l'infrastructure et l'environnement d'installation pour RHCOS est cohérent pour chaque installation ;
  - Ces fonctionnalités sont souvent absentes des installations en cloud. Installation de l'infrastructure fournie par l'installateur Pour les installations fournies par l'installateur dans OpenShift 4, il est possible d'interrompre la procédure d'installation d'OpenShift pour ajouter un fichier manifeste.
  - Bien que cela doive être fait de manière réfléchie, tout fichier de configuration peut être modifié et appliqué à tous les noeuds **worker** ou **control plane** en utilisant une fonction appelée **Ignition**. Les configurations d'Ignition sont mises en place pour le premier démarrage de chaque système. Les mêmes configs d'Ignition peuvent également être utilisées après que le cluster soit venu modifier les nœuds par le biais de MachineConfigs, en utilisant le Machine Config Operator (MCO).
- Parmi les exemples de modifications liées à la sécurité qui peuvent être activées lors de l'installation d'infrastructures fournies par l'installateur, on peut citer:
  - le mode FIPS, le cryptage des disques en nuage (EBS crypté) et les services de cryptage en nuage fournis par les fournisseurs de services en nuage (par exemple AWS, GCP, Azure).
  - Il existe des composants qui peuvent être modifiés pour l'installation finale, comme les fichiers de configuration RHCOS via les configs d'Ignition, le risque de mise en œuvre de ces composants peut varier.
  - Il est recommandé d'évaluer les risques en fonction des meilleures pratiques ou des exigences de chaque organisation.

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- **Modifications de sécurité après l'installation :**
  - Une fois que le cluster OpenShift 4 est opérationnel, il est facile d'apporter des modifications directement aux systèmes RHCOS du cluster à l'aide du **Machine Config Operator (MCO)**.
  - Contrairement à la configuration effectuée sur des installations en **baremetal** les fonctionnalités ajoutées par le MCO sont sauvegardées par le cluster.
  - Les fonctionnalités du **MCO** sont appliquées immédiatement **aux nœuds existants et à tous les nœuds ajoutés dans le futur**. De même, il existe d'autres opérateurs qui peuvent être utilisés pour gérer des fonctions de sécurité spécifiques avec des attentes d'application similaires.
- **Machine Config Operator :**
  - L'utilisation du **Machine Config Operator** permet d'ajouter **des clés SSH**, des fichiers de configuration, des fichiers d'unité **systemd**, des arguments du noyau, des possibilités d'audit et d'autres fonctionnalités. Ces modifications peuvent être choisies pour s'appliquer aux pools **workers** et **control plane**.
  - Le format des MachineConfigs est le même que celui utilisé lors de l'installation (**ignition files**)

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- **Ajout de services de sécurité :**
  - Si l'hébergeur ou le fournisseur de cloud exige que des agents ou d'autres services soient exécutés sur chaque nœud, les éléments suivants peuvent être créés :
  - Kubernetes DaemonSet :
    - Les DaemonSets sont conçus pour exécuter un pod sur tous les nœuds du cluster, bien qu'ils puissent être limités à certains nœuds en fonction d'un nodeSelector spécifique.
    - Les DaemonSets sont particulièrement utiles pour gérer des événements tels que la journalisation, les analyses de virus, la vérification de l'intégrité des fichiers et le stockage en clusters sur chaque nœud.
  - Opérateur OpenShift :
    - Les opérateurs permettent **à un concepteur de formuler une solution de sécurité** comme un service déployable et maintenable qui tire parti de la disponibilité du système OpenShift. Les contrôles de version et les mises à jour des fonctions de sécurité sont gérés de la même manière que les autres services OpenShift.

# CoreOS - Points clefs / Philosophie

CONFIDENTIAL Designator

- En général, les deux approches **DaemonSet** et **Operateur** peuvent être utilisées s'il est acceptable de démarrer le service après le démarrage du moteur du Container CRI-O.  
Un modèle courant dans la pratique de Kubernetes est d'exécuter des démons de sécurité, ou des opérateurs OpenShift personnalisés, comme des pods privilégiés (c'est-à-dire root).
- **Il est recommandé qu'après la Containerisation des services, que :**
  - le pod soit paramétré - pour fonctionner là où c'est nécessaire
    - Tenir compte des endpoints, des ports et du niveau d'accès
    - Veiller à ce que les points des montages soient réduits au minimum
    - Utiliser des comptes de service pour les lier à des autorisations (RoleBinding). Par exemple, si le service a besoin d'accéder à un device spécifique sur le nœud, le service doit être exécuté avec cet ensemble de permissions.
    - Pensez à la scalabilité. Le nœud RHCOS est un membre du cluster. Se concentrer sur des nœuds individuels peut conduire à des solutions non évolutives qui rendent la maintenance, la mise à niveau et l'audit plus difficiles.

# CoreOS - Tâche de Sécurité

CONFIDENTIAL Designator

- **Tâches pour la réalisation des objectifs de sécurité**

- L'objectif de RHCOS est de faire partie de la plate-forme de Containers OpenShift.
- En revanche, la plupart des autres systèmes d'exploitation constituent les fondations sur lesquelles un univers de services pourrait être construit. Par conséquent, RHCOS fait des choix très avisés en ce qui concerne les interactions entre l'homme et l'interface : l'intervention humaine ne devrait pas être nécessaire pour le fonctionnement quotidien de RHCOS.
- Seules des circonstances exceptionnelles, telles que la reprise après sinistre, devraient exiger une intervention humaine pour le fonctionnement du RHCOS lui-même. La conception des RHCOS supporte la création de services qui permettent aux praticiens de minimiser cette interaction et de mieux caractériser la stratégie de gestion des services organisationnels. Ainsi, les services fonctionnent sous l'égide d'OpenShift et de son immédiateté de contrôle et de vigilance. Ainsi, le nombre de points d'entrée dans le cluster lors des événements d'intervention peut être réduit au minimum.

Certaines améliorations de la sécurité d'OpenShift doivent être effectuées avant le premier démarrage du cluster OpenShift, par exemple lors d'une installation OpenShift, pendant la construction du cluster ou lors d'une installation de système d'exploitation à nu. D'autres tâches peuvent être effectuées après le démarrage du cluster. Les sections suivantes décrivent les deux différents types de tâches.

# CoreOS - Tâche de Sécurité

CONFIDENTIAL Designator

## Sécurité du RHCOS pendant l'installation des clusters

- Les décisions suivantes sont à prendre en compte comme étant un préalables au premier démarrage du cluster.
  - le **mode FIPS** et le **Full Disk Encryption (FDE)** -
- **Mode FIPS**
  - Afin de garantir cette conformité, le mode **FIPS** doit être activé avant le premier démarrage des nœuds de clusters et ne doit pas être désactivé une fois activé.
  - De plus, une fois qu'un cluster a été installé, le mode FIPS doit rester activé pendant toute la durée de vie du cluster et pour tout nouveau nœud introduit, de peur qu'un point faible nesoit introduit.
  - FIPS ne peut pas être activé après l'installation ou désactivé et réactivé car il causera des problèmes entre les nœuds et le cluster. Cela peut se présenter sous la forme d'erreurs bizarres entre les systèmes d'un cluster effectuant des transactions cryptographiques entre eux.

# CoreOS - Tâche de Sécurité

CONFIDENTIAL Designator

## Sécurité du RHCOS pendant l'installation des clusters

- **Chiffrement intégral du disque**
  - Depuis OpenShift 4.3, RHCOS prend en charge le cryptage intégral du disque système. Tel que mis en œuvre, le cryptage du disque RHCOS est conforme à **FIPS** si le mode **FIPS** est activé.
  - Le chiffrement du disque RHCOS utilisera le chiffrement par bloc AES256-CBC. Ce chiffrement est nommé à différents endroits du système et dans les fichiers de configuration comme cbc(aes), aescbc, aes cbc et similaires. Par défaut, RHCOS démarre sans cryptage activé. Actuellement, les deux seules façons dont le cryptage du disque est pris en charge sont soit avec TPM2 et un serveur Tang.
- **Sécurité de la configuration du réseau RHCOS**
  - Le RHCOS lui-même n'a pas besoin d'être mis en réseau. Toutefois, lorsqu'il est déployé dans le contexte de la plate-forme de Containers OpenShift, la mise en réseau est nécessaire.
  - Par défaut, RHCOS est configuré pour utiliser le **DHCP** afin d'obtenir à la fois des informations de mise en réseau (**IP/Masque/Gateway/DNS**) et l'identité du réseau (**Hostname**).
  - Il faudra modifier **RHCOS** pour tenir compte d'une exigence d'adressage statique.

# CoreOS - Tâche de Sécurité

CONFIDENTIAL Designator

## Sécurité du RHCOS après la mise en place du cluster

- Dans la mesure du possible, les changements post-déploiement des nœuds RHCOS doivent être effectués au niveau des clusters.
  - Les modifications directes des nœuds RHCOS ne doivent pas être effectuées en se connectant à un nœud et en ajoutant des progiciels ou en modifiant des fichiers de configuration.
  - Une fois qu'un cluster OpenShift est opérationnel, la plupart des modifications directes aux nœuds RHCOS doivent être effectuées en appliquant les objets **MachineConfig** aux nœuds workers ou master.
- Les activités de sécurité post-déploiement pour RHCOS comprennent l'ajout de modules du noyau qui sont appliqués au démarrage des nœuds RHCOS et la modification des fichiers de configuration liés à la sécurité.
  - Pour illustrer la modification d'un fichier de configuration, la procédure ci-dessous modifie l'emplacement du serveur de temps utilisé pour synchroniser le service chronyd sur chaque nœud.
  - Les arguments du noyau et les modifications du fichier de configuration RHCOS peuvent être appliqués aux nœuds par le biais de MachineConfigs.

# CoreOS - Tâche de Sécurité

CONFIDENTIAL Designator

## Sécurité du RHCOS après la mise en place du cluster

- **Modifier un fichier de configuration sur les nœuds RHCOS :**

La procédure d'application de tout fichier de configuration à un nœud RHCOS est fondamentalement la même, quel que soit le fichier appliqué. La procédure suivante décrit comment remplacer le fichier chrony.conf afin de pouvoir modifier les paramètres ou attribuer un serveur de temps différent dans le service chronyd. Appliquez le changement en utilisant un MachineConfig, et il sera déployé à tous les noeuds workers ou master.

# CoreOS - Tâche de Sécurité

1 - Pour créer le contenu du fichier chrony.conf et l'encoder en base64, procédez comme suit :

```
$ cat << EOF | base64  
server clock.example.com iburst  
driftfile /var/lib/chrony/drift  
makestep 1.0 3  
rtcsync  
logdir /var/log/chrony  
EOF  
c2Vyd...  
pYi9jaHJvbnkv  
ZHJpZnQKbWFrZXN0ZXAgMS4wIDMKcnRjc3luYwpsb2dkaXIgL3Zhci9sb2cvY2h  
yb255Cg==
```

3 - Faire une copie de sauvegarde du fichier de configuration

Ce fichier peut être utilisé soit pendant l'installation d'OpenShift, soit via le MCO après l'exécution du cluster. Si le cluster n'est pas encore en place, générez des fichiers manifestes, ajoutez ce fichier au répertoire openshift et continuez à créer le cluster. Si le cluster est déjà en cours d'exécution, appliquez le fichier comme suit :

```
$ oc apply -f ./50_workers-chrony-configuration.yaml
```

2 - Remplacement de la chaîne base64 par celle créée précédemment, comme le montre l'exemple suivant. Dans cet exemple, le fichier est ajouté à tous les nœuds de workers :

```
$ cat << EOF > ./50_workers-chrony-configuration.yaml  
apiVersion: machineconfiguration.openshift.io/v1  
kind: MachineConfig  
metadata:  
  labels:  
    machineconfiguration.openshift.io/role: worker  
  name: workers-chrony-configuration  
spec:  
  config:  
    ignition:  
      config: {}  
      security:  
        tls: {}  
      timeouts: {}  
      version: 2.2.0  
    networkd: {}  
    passwd: {}  
    storage:  
      files:  
        - contents:  
            source: data:text/plain;charset=utf-  
AvdmFyL2xpYi9jaHJvbnkvZHJpZnQKbWFrZXN0ZXAgMS4wIDMKcnRjc3luYwpsb  
2dkaXIgL3Zhci9sb2cvY2hyb255Cg==  
        verification: {}  
      filesystem: root  
      mode: 420  
      path: /etc/chrony.conf  
osImageURL: ""  
EOF
```

# CoreOS - Tâche de Sécurité

CONFIDENTIAL Designator

## Sécurité du RHCOS après la mise en place du cluster

- **Ajoutez un argument du noyau aux nœuds RHCOS :**

- Certaines fonctions de sécurité doivent être réalisées en passant des arguments au noyau lorsqu'un nœud RHCOS démarre. Il existe des arguments du noyau qui permettent de faire des compromis entre la sécurité et la disponibilité du système.
- Un exemple d'argument lié au noyau qui peut avoir un impact sur la sécurité RHCOS est **pti**. L'activation de l'argument noyau d'isolation des tables de pages du noyau (pti=on) durcit le noyau pour empêcher le contournement de la randomisation de l'espace d'adressage du noyau (KASLR), tout en atténuant la vulnérabilité de sécurité Meltdown.
- Les arguments du noyau peuvent également être utilisés pour désactiver des ports physiques, tels que l'Universal Serial Bus (USB) and Firewire (IEEE 1394). Les périphériques d'entrée/sortie (E/S) comprennent, par exemple, les lecteurs de disques compacts (CD) et (DVD). La désactivation ou la suppression physique de ces ports de connexion et dispositifs d'entrée/sortie permet d'éviter l'exfiltration d'informations des systèmes d'information et l'introduction de codes malveillants dans les systèmes à partir de ces ports/dispositifs.

# CoreOS - Tâche de Sécurité

CONFIDENTIAL Designator

1 - Ajoutez l'argument du noyau à une entrée MachineConfig dans un fichier et nommez ce fichier de manière à ce qu'il soit inséré à un endroit approprié dans la liste des MachineConfigs. Par exemple, 20-worker-ption.yaml :

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 20-worker-ption.yaml
spec:
  config:
    ignition:
      version: 2.2.0
    kernelArguments:
      - pti=on
```

2 - Utilisez oc create pour ajouter l'argument du noyau à tous les noeuds workers du cluster :

```
oc create -f 20-worker-ption.yaml
oc get MachineConfig
oc get nodes
```

# CoreOS - Dépannage de la sécurité RHCOS

CONFIDENTIAL Designator

- Utilisation de la console Web pour le dépannage
- Dépannage des nœuds à partir de la ligne de commande
  - oc adm must-gather
- Dépannage de l'accès direct aux nœuds par Shell
  - ssh core@<yournode>
- Accès privilégié au shell avec oc debug
  - oc get nodes
  - oc debug node/<yournode>  
chroot /host bash
- Mises à jour du RHCOS

# CRI-O Container Sécurité

The OpenShift operating  
system

# CRI0 - Container Sécurité

CONFIDENTIAL Designator

**Les capacités de sécurité des Containers et des hôtes s'appuient sur quatre grands domaines :**

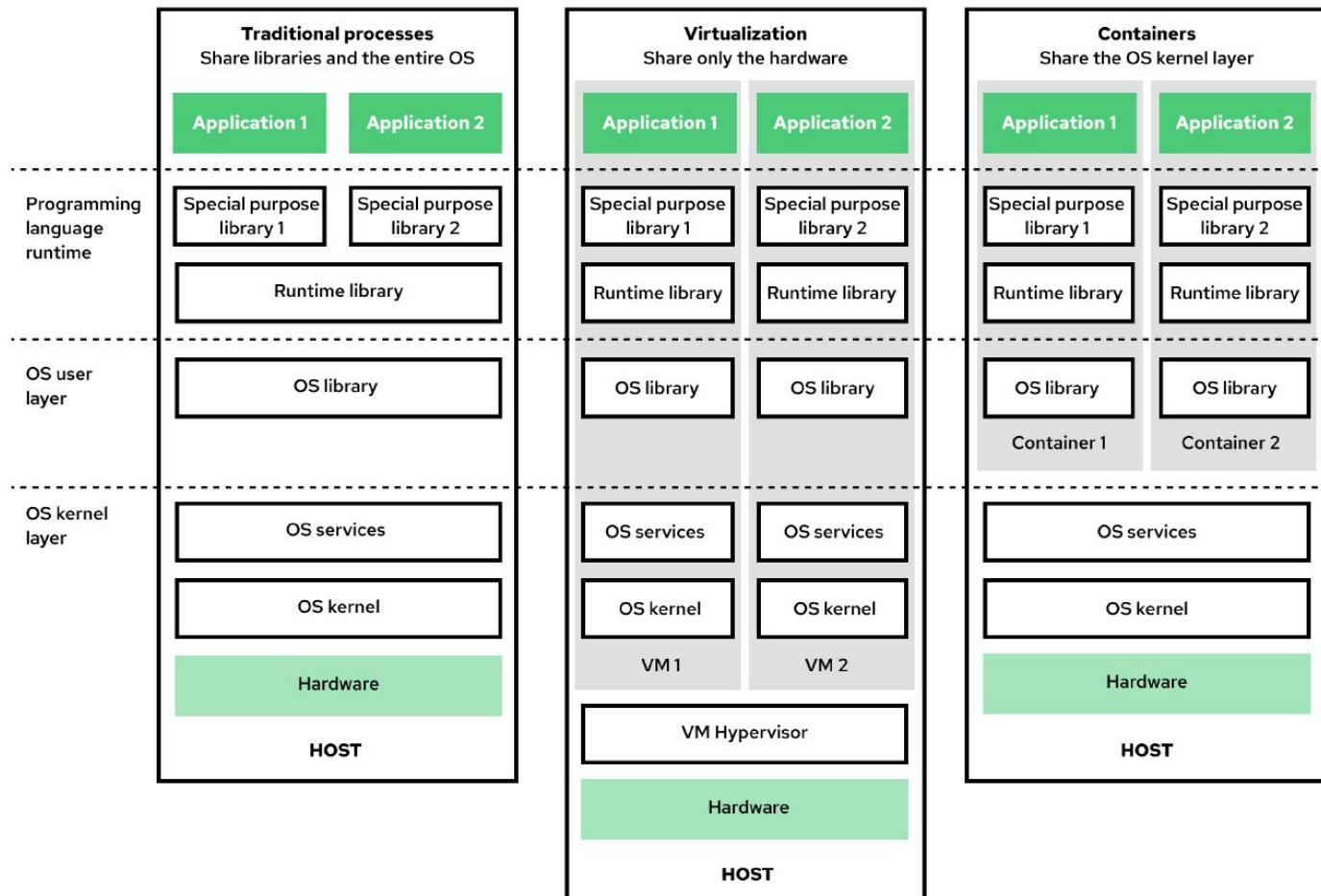
- Les **namespaces Linux** créent un partitionnement des ressources système, formant les limites logiques autour de la structure du Container
- La fonction secure computing (**seccomp**) permet de filtrer la disponibilité des appels système dans un Container
- Les **Linux capabilities** permettent la réduction des privilèges qui seraient normalement autorisés dans le contexte de l'utilisateur "root"
- Le système **SELinux** (Security-Enhanced Linux) impose un contrôle d'accès obligatoire pour les processus système, les services, les fichiers et les ressources réseau. Travaillant en amont avec la National Security Agency (NSA), Red Hat a introduit SELinux dans son produit Red Hat Enterprise Linux 4 depuis plus de 15 ans il y a quelques années.

# CARIO - Container Sécurité

CONFIDENTIAL Designator

- **L'évolution du Workload et de l'infrastructure vers les Containers**

- Les Containers sont souvent comparés aux machines virtuelles (VM) parce que les deux technologies offrent une isolation et un packing universel.
- La figure suivante compare des applications traditionnelles, qui s'exécutent comme des processus normaux sans sandbox, avec des applications s'exécutant sur des Containers et d'autres sur des VM.



# CRI0 - Container Sécurité

CONFIDENTIAL Designator

- Les Containers divisent le packaging traditionnel d'un système d'exploitation en une couche noyau et une couche utilisateur.
    - La couche noyau du système d'exploitation s'exécute sur l'hôte et comprend les services du noyau et du système d'exploitation tels que le système de fichiers en réseau et les clients de synchronisation du temps.
    - La couche utilisateur du système d'exploitation comprend les bibliothèques du système d'exploitation telles que la glibc et des outils tels que le shell Bash.
      - Un Container exécute une couche utilisateur minimale adaptée à cette application spécifique.
      - Tous les Containers partagent la couche noyau du système d'exploitation à partir de l'hôte.
    - Les applications Containerisées invoquent toujours des appels système vers la couche hôte et les API de service du système d'exploitation en utilisant les mêmes mécanismes que les applications traditionnelles.
- Avec les Containers, il n'y a pas de couche intermédiaire qui ajoute de la surcharge.

# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Comparaison des Containers et des machines virtuelles pour l'isolation des applications :

- Les Containers et les machines virtuelles (VM) isolent les applications de sorte que les vulnérabilités d'une application n'affectent pas les autres applications partageant le même hôte.
- La plupart des entreprises ne peuvent pas mettre à jour la couche noyau du système d'exploitation pour les correctifs de sécurité assez rapidement pour répondre aux brèches, car elles doivent d'abord s'assurer que les applications continuent de fonctionner sous les nouvelles versions.
- Le système d'exploitation hôte peut être mis à jour sans casser une application Containerisée car la plupart des applications ne dépendent pas d'un noyau de système d'exploitation spécifique et d'une version du système d'exploitation.
  - Elles dépendent des bibliothèques du système d'exploitation qui donnent accès aux appels système du noyau et aux API des services du système d'exploitation.
  - Contrairement à une VM, un Container n'inclut pas son propre noyau, ce qui limite la surface d'attaque du Container.
  - Les images de Containers permettent d'accélérer les pipelines d'intégration continue (CI) et de livraison continue (CD) car la construction des images de Containers est plus rapide que celle des images de VM, et les Containers démarrent plus vite que les VM. Cela permet aux organisations de déployer plus fréquemment la production et de réduire le temps nécessaire aux corrections de sécurité.



# CRI-O - Container Sécurité

CONFIDENTIAL Designator

- **Créer des Containers Linux avec un moteur de Container**
  - Un **moteur de Container** fournit un ensemble d'outils pour des tâches telles que la création d'images de Containers et le démarrage de Containers.
  - **CRI-O** est le moteur de Containers d'OpenShift. En tant que implémentation de la **CRI** (Container Runtime Interface) de Kubernetes, **CRI-O** permet l'utilisation de runtimes compatibles avec l'**OCI** (Open Container Initiative).
  - CRI-O est le moteur de Container par défaut d'OpenShift 4, en remplacement de Docker, qui était le moteur par défaut d'OpenShift 3.11.
- **Du point de vue de la sécurité, CRI-O a été développé pour :**
  - **s'aligner sur les versions de Kubernetes**

En s'alignant sur les versions de Kubernetes, les mises à jour du CRI-O sont toujours effectuées dans le seul but de mieux travailler avec la version actuelle de Kubernetes.
  - **Avoir une surface d'attaque réduite**

Le champ d'application du CRI-O est lié à l'interface d'exécution des Containers (CRI). En n'incluant pas de fonctionnalités supplémentaires pour une utilisation en ligne de commande directe ou d'autres facilités d'orchestration, l'empreinte du CRI-O est plus petite et les vulnérabilités sont réduites.
  - **Améliorer les performances**

Un ensemble de fonctionnalités plus petit encourage une meilleure performance parmi les fonctionnalités prises en charge.
  - **Permettre l'inclusion de différents runtimes de Container Actuellement, OpenShift supporte le runc container runtime.**

Cependant, à mesure que d'autres runtimes deviennent disponibles pour un support complet, comme le runtime Kata Containers, ils peuvent être supportés par CRI-O.



# CRI-O - Container Sécurité

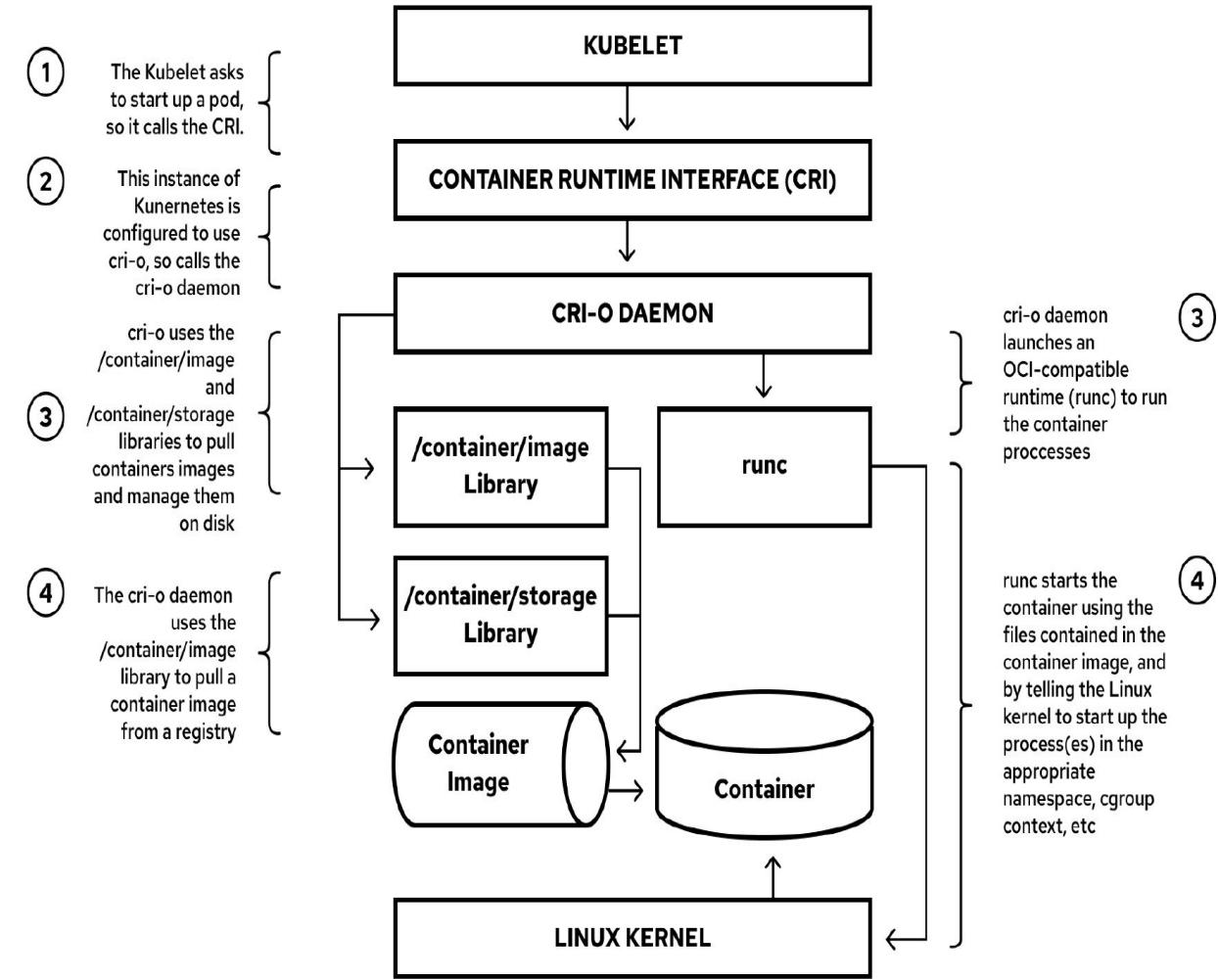
CONFIDENTIAL Designator

## Architecture CRI-O

- CRI-O n'est pas pris en charge en tant que moteur de Container autonome et doit être utilisé comme moteur de Container pour OpenShift. Pour faire fonctionner des Containers sans OpenShift, utilisez **Podman** sur un hôte **RHEL**.
- Le schéma qui suit montre les différents composants d'OpenShift et leur rôle dans la création d'un pod.
  - Les répertoires `/container/image` et `/container/storage` sont donnés à titre. **Par défaut, pour les déploiements OpenShift, le stockage du CRI-O se trouve sous `/var/lib/containers`.**
- **CRI-O** supporte le mode **FIPS**, en ce sens qu'il configure les Containers pour informer qu'ils fonctionnent en mode **FIPS**.

Remarque:

- La plupart des interactions avec **CRI-O** se font via le cluster OpenShift qui est le gestionnaire Containers. Toutefois, pour des questions de sécurité avancées, il existe des moyens de tuner **CRI-O** ou d'accéder directement à **CRI-O**.



# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Tunning de CRI-O

- Le tuning de **CRI-O** ne soit pas encouragé, l'adaptation de CRI-O pour des besoins de sécurité avancés peut se faire en modifiant le contenu du fichier **/etc/crio/crio.conf**.
- Dans **crio.conf**, on peut définir des éléments tels que les limites **PID** pour fixer le nombre maximum de processus autorisés dans un Container.
- On utilisera **MachineConfig** pour modifier un fichier de configuration RHCOS.

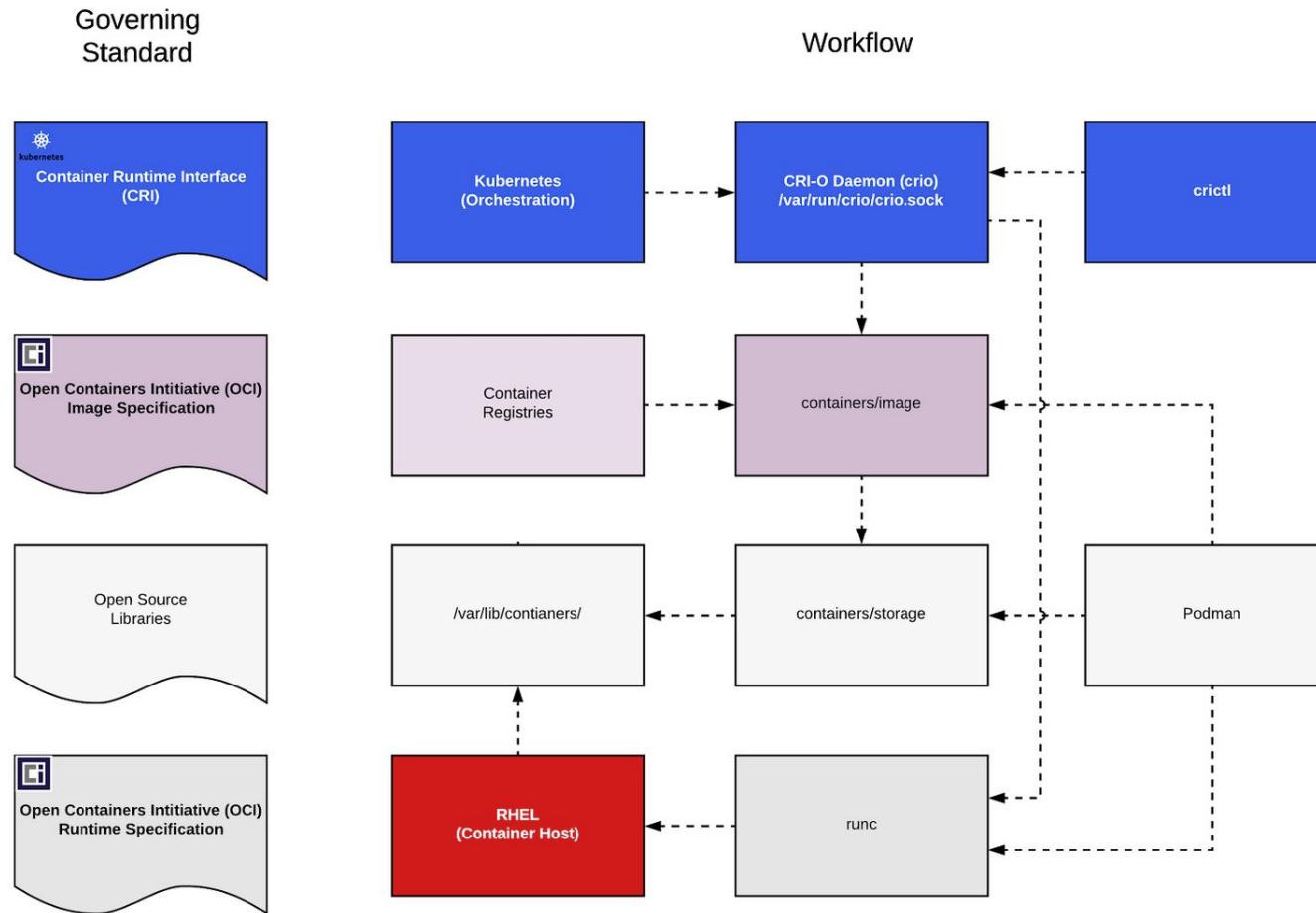
## Accéder directement au CRI-O

- Tout comme le tuning de **CRI-O** n'est pas encouragé, l'accès direct à **CRI-O** ne l'est pas non plus.
- A des fins de débogage, on peut accéder au **CRI-O** directement en utilisant la commande **crictl** depuis n'importe quel nœud OpenShift. Par exemple, sur un nœud OpenShift, on peut lister les **Containers** en cours d'exécution et voir les messages de logging pour un Container spécifique comme suit :
  - **crictl ps**
  - **crictl logs <pod ID>**

# CRI-O - Container Sécurité

CONFIDENTIAL Designator

- D'autres fonctionnalités de la commande **cubectl** peuvent être utilisées pour gérer les **Containers** et les **images** de Containers sur un nœud OpenShift. Pour des exemples d'autres utilisations de **cubectl**, voir la page **cubectl** sur Github. Un blog de Dan Walsh détaille la relation de complémentarité entre les outils cubectl et Podman : <https://www.openshift.com/blog/cubectl-vs-podman>



# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Sécurité des Containers dans le noyau Linux

- La partie d'un **moteur de Container** qui démarre et surveille efficacement les Containers est le **runtime** de Container. **runc** est le runtime de Container d'OpenShift.

Un **runtime Container** exploite un certain nombre de fonctionnalités du noyau Linux pour démarrer un Container :

- La caractéristique principale est le **namespace Linux**.
  - Les **namespaces** créent la **sandbox** qui isole un processus des autres processus s'exécutant en dehors de la sandbox.
  - Les **namespaces** fournissent, entre autres, une vue séparée de la table des processus, du système de fichiers hôte et du réseau.
- Les groupes de contrôle (**Cgroups**) constituent une autre caractéristique importante.
  - Les processus "**sandboxed**" sont des processus réguliers gérés par le noyau hôte partageant les ressources matérielles de l'hôte, telles que la mémoire et le CPU.
  - Les groupes de contrôle (**Cgroups**) imposent des limites à la consommation par **tranches de CPU**, **des pages de mémoire** et de la **capacité d'entrée/sortie** des processus, empêchant ainsi un seul processus d'utiliser toute la capacité de l'hôte pour lui-même.
- Bien qu'ils ne soient pas strictement nécessaires pour créer un processus "sandboxed", **SELinux**, **capabilities** et **seccomp** complètent les **namespaces** et les groupes de contrôle (**Cgroups**) pour empêcher un processus de sortir de son confinement et d'interférer indirectement avec d'autres processus du même hôte.

# CARIO - Container Sécurité

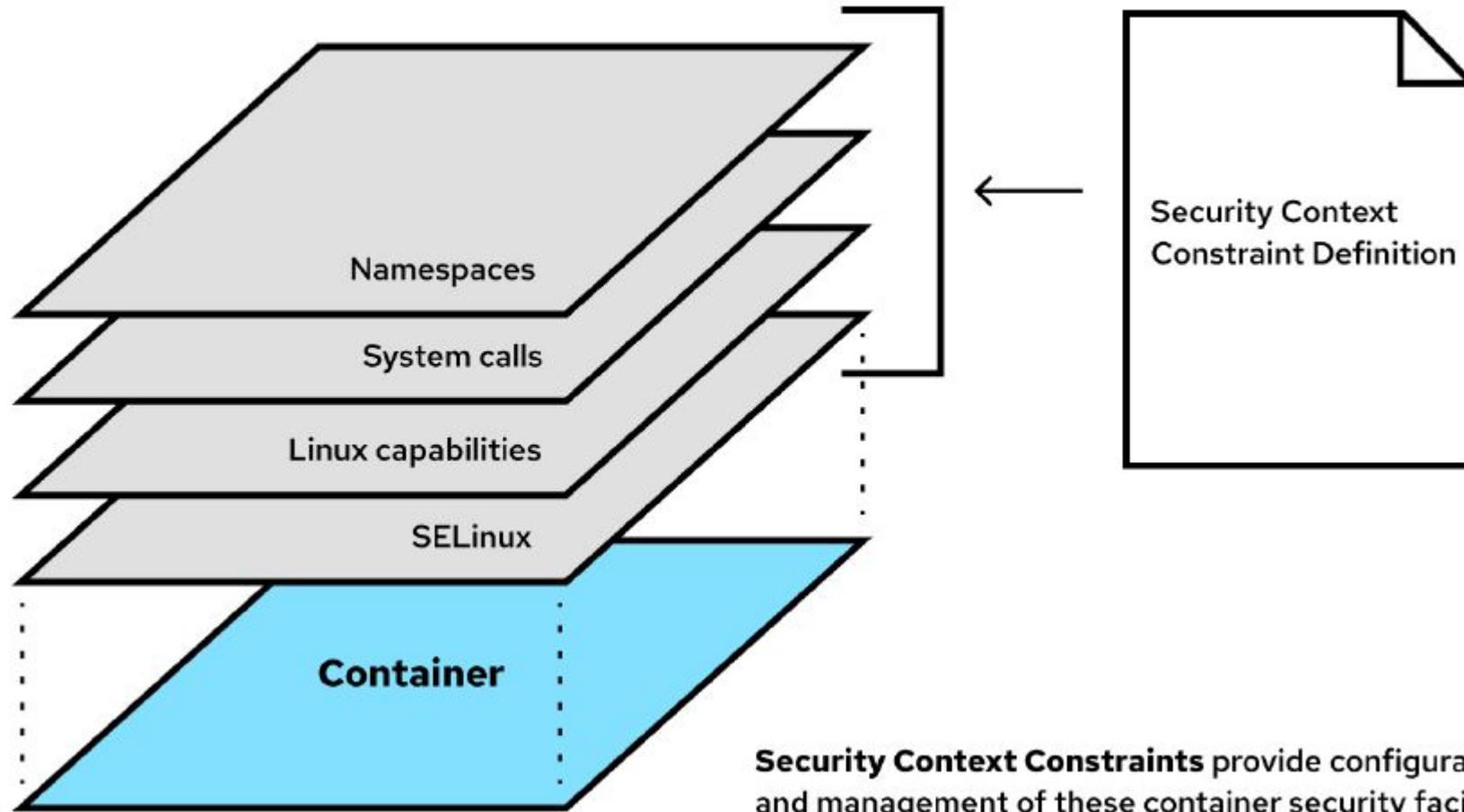
CONFIDENTIAL Designator

- D'autres logiciels, tels que les gestionnaires de services du système d'exploitation, **systemd** et les navigateurs web, dont **Firefox**, utilisent certaines des mêmes fonctionnalités du noyau pour exécuter des processus dans une sandbox.  
Les Containers Linux ne sont qu'un scénario spécifique, bien que plus général, de création de processus "sandboxed".
- Le rôle principal d'un **runtime de Container** est de faciliter l'exploitation de ces fonctionnalités en fournissant, par exemple, des outils de ligne de commande de haut niveau et des API qui s'interface avec un runtime de Container et des registres de Containers. Sans moteur de Container, un administrateur système devrait gérer chacune de ces fonctionnalités du noyau individuellement, en utilisant des commandes de bas niveau du système d'exploitation qui prendraient beaucoup de temps et seraient sujettes aux erreurs.
- Un **runtime de Container** est, par essence, une aide qui lance des processus à l'intérieur d'un ensemble de groupes de contrôle (**Cgroups**) et de **namespaces**, dans des contextes **SELinux restraints**, un ensemble secure computing (**seccomp**) et des paramètres **seccomp**.
- Un **runtime Container** garde la trace des processus qu'il a lancés et leur attribue un **ID Container**. Ces ID de Container n'ont de sens que pour le runtime de Container qui les a créés. Ils n'ont aucune signification pour le noyau.
- Un **moteur de Container** fournit également des fonctions de **debug** telles que le démarrage de nouveaux processus dans le même sandbox qu'un Container existant. Ces fonctions sont utiles pour exécuter des outils de diagnostic à l'intérieur d'un Container afin d'inspecter l'état des processus Containerisés et de s'interfacer avec les systèmes de fichiers et les réseaux de la même manière que le Container.

# CRIO - Container Sécurité

CONFIDENTIAL Designator

- Cette figure présente les quatre grands domaines de paramètres de sûreté qui peuvent être appliqués aux Containers.
- Les **Security Context Constraints (SCC)** dans OpenShift permettent de configurer et de gérer ce dispositif de sécurité.



# Namespace Linux

The OpenShift operating system

# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Description des namespaces Linux

- La fonctionnalité des **namespaces Linux** est une fonctionnalité du **noyau** qui permet l'abstraction de ressources pour les **processus**. C'est un moyen de changer la vue d'un processus sur le système.
- Un **processus** à l'intérieur d'un **namespace** ne voit que les ressources associées à ce namespace.
- Les **namespaces Linux** fournissent un niveau d'isolation pour plusieurs types de ressources :
  - Process Identifier (**PID**),
  - Inter-process Communication (**IPC**),
  - réseau, UNIX Time Sharing (**UTS**),
  - points de montage (**Mount**)
  - groupes de contrôle (**cgroups**),
  - utilisateur (**user**).
- Par exemple, les processus d'un **namespace réseau** ont leur propre pile réseau et ne peuvent pas voir les **ressources réseau** telles que les interfaces et les ports d'écoute des autres **namespaces** ou du **système hôte**. En fournissant cette couche d'isolation des ressources, les **namespaces Linux** sont une partie essentielle de la mise en œuvre des containers.

## Présentation des types de namespaces Linux

Le noyau Linux fournit les **namespaces linux** suivants :

- **PID** : Les **processus** dans un **namespace PID** ont leur propre espace de numérotation. Le premier **processus** dans le **namespace** reçoit le **PID 1**.  
De l'extérieur, la commande ps affiche toujours ces processus, mais le **noyau** mappe le **PID** dans le **namespace** à un **PID** sur le système hôte.
- **IPC** : Les **namespaces IPC** isolent les ressources IPC (communication interprocessus) du système V.  
Un processus dans un namespace IPC ne peut voir et utiliser que les objets IPC de cet namespace.
- **Réseau** : Les processus d'un **namespace réseau** disposent de leurs propres ressources réseau, telles que les périphériques, les piles IP, les règles de pare-feu et les tables de routage. Les **namespaces de réseau** permettent la construction de réseaux virtuels.
- **UTS** : Un namespace **UNIX Time Share** isole le **nom d'hôte et le nom de domaine**. De cette façon, chaque namespace peut avoir un nom d'hôte personnalisé. Le fait de changer le nom d'hôte à l'intérieur du **namespace** ne change pas le nom d'hôte sur le système hôte.

# CARIO - Container Sécurité

CONFIDENTIAL Designator

- **Mount** : Les processus dans un **namespace mount** ne peuvent voir que les **points de montage dans ce namespace**. Ils ne peuvent en aucun cas interagir avec les points de montage du système hôte ou d'un autre namespace de montage. Le namespaces de montage résument une vue complète du système de fichiers.
- **Cgroup** : Le système expose les **cgroups** comme une hiérarchie de répertoires sous /sys/fs/cgroup/.
  - Les **namespaces cgroup** restreignent la vue de cette hiérarchie de répertoires. De cette façon, les processus d'un namespace cgroup ne peuvent obtenir aucune information sur les limites de ressources fixées aux autres processus du système.
  - Remarquez que l'on **fixe des limites aux ressources au niveau du cgroup** ; les namespaces cgroup ne restreignent que la vue de la hiérarchie des répertoires.
- **User** : Les **namespaces user** isolent les identifiants des utilisateurs et des groupes.
  - Un processus à l'intérieur d'un namespace user apparaît avec **un ID utilisateur (UID)** et **un ID groupe (GID)** qui peuvent être différents sur le système hôte.
  - Cela permet de donner un accès root à un processus à l'intérieur d'un container. À l'extérieur du container, le système mappe le processus à un compte utilisateur non privilégié.
  - Un processus vulnérable qui doit s'exécuter en tant que root peut le faire à l'intérieur de son namespace utilisateur. S'il parvient à s'échapper de son namespace, il n'a que les droits du compte utilisateur non privilégié associé.
  - Les namespaces utilisateur sont déjà disponibles dans des outils hôtes tels que **Podman** mais ne sont pas encore implémentés dans Kubernetes.

**Un processus peut appartenir à différents types de namespaces en même temps.** Il peut appartenir à un **namespace PID** qui possède son propre espace de numéro d'identification de processus, ainsi qu'à un **namespace réseau** et un **namespace IPC**.



# C-group

The OpenShift operating system

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Gérer les ressources avec les Cgroups

- Les groupes de contrôle (ou **C-Groupes** en abrégé) sont un moyen de limiter l'accès aux ressources du système.

Parmi les différentes ressources qui peuvent être limitées, citons :

- **blkio** - Fixe des limites à la bande passante disponible vers et depuis les périphériques de blocage
- **cpu** - Fixe des limites au temps CPU disponible
- **cpuset** - Fixe les limites des unités centrales et des zones de mémoire disponibles dans les systèmes NUMA
- **memory** - Fixe les limites de la mémoire disponible
- **freezer** - Les tâches dans ce cgroupue sont suspendues

Ces sous-systèmes sont également connus sous le nom de contrôleurs de ressources ou de contrôleurs. OpenShift utilise des groupes de contrôle pour répartir équitablement les ressources disponibles entre les locataires. Les groupes protègent également les processus des containers contre les dépassemens par des processus malhonnêtes ou cupides.

# CARIO - Container Sécurité

CONFIDENTIAL Designator

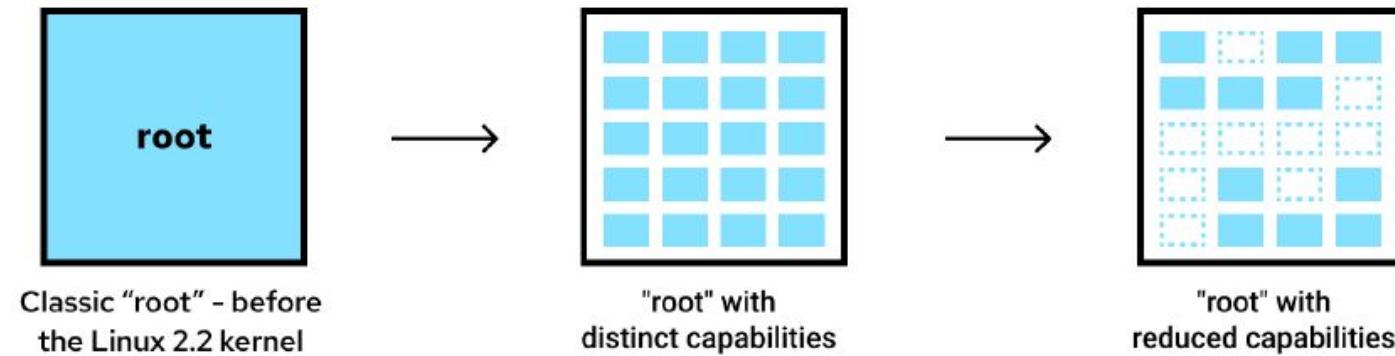
## Linux Capabilities

- Traditionnellement, sous Linux, il existe deux types d'identités d'utilisateurs : **privilégiées** et **non privilégiées**, l'utilisateur **root** étant **le compte privilégié** qui peut tout faire.
  - Cette absence de granularité a été problématique car les processus qui doivent effectuer des tâches administratives fonctionnent comme le compte racine privilégié et, en l'absence d'autres mécanismes de protection, peuvent tout faire.
  - Par conséquent, Linux fournit des capacités - des groupes de pouvoirs de super-utilisateur qui peuvent être sélectivement activés ou désactivés pour limiter le pouvoir dont dispose normalement l'utilisateur root. On peut en effet considérer que l'utilisateur root, au sens traditionnel du terme, possède toutes les capacités existantes.
- De nombreuses applications containerisées n'ont besoin que d'un sous-ensemble des priviléges de l'utilisateur root, par exemple un container faisant tourner un serveur httpd peut avoir besoin d'être bindé qu'au port 80, le reste de ses opérations peut être effectué en tant qu'utilisateur non root.
  - En termes de capacités, le serveur http a besoin de la **capacité CAP\_NET\_BIND** même lorsqu'il fonctionne en tant que root afin de se binder à un port privilégié.
  - D'autre part, il n'y a aucune raison pour que la plupart des containers aient besoin de régler le temps système, ce que root aurait normalement permis, et il n'est donc pas nécessaire qu'ils aient la **capacité CAP\_SYS\_TIME**.

# CRIO - Container Sécurité

CONFIDENTIAL Designator

- Le premier encadré, le comportement classique de l'utilisateur root a vu toutes les capacités activées - cela reflète les capacités de l'utilisateur root avant l'introduction du noyau Linux 2.2.
- Le deuxième encadré indique qu'avec le noyau Linux 2.2, la notion d'utilisateur **privilégié root** a été divisée en unités distinctes appelées **Capabilities**.



Un aperçu de toutes les capacités disponibles peut être trouvé dans la page de manuel "**Linux Capabilities**":  
<https://www.man7.org/linux/man-pages/man7/capabilities.7.html>

Les **capabilities** d'un container en cours d'exécution sur un hôte RHEL peuvent être visualisées en exécutant:  
**podman inspect \$container**.

# CARIO - Container Sécurité

CONFIDENTIAL Designator

- Les contraintes de contexte de sécurité (**SCC**) permettent de contrôler l'attribution des capacités de sécurité lors de la création des pods. C'est ce qui empêche un utilisateur d'ajouter arbitrairement des priviléges puissants tels que **CAP\_SYS\_ADMIN**.

Dans les instances **SCC**, il y a trois champs liés aux **Capabilities** :

- **requiredDropCapabilities** :  
Ces capacités seront toujours supprimées et ne peuvent pas être ajoutées
  - **allowedCapabilities** :  
Ces capacités peuvent être ajoutées. Il y a deux valeurs spéciales, null et \*, qui peuvent être ajoutées. Null signifie qu'aucune capacité supplémentaire ne peut être ajoutée et \* signifie que n'importe quelle capacité peut être ajoutée.
  - **defaultAddCapabilities** :  
Ces capacités seront ajoutées au pod à moins que le pod ne les supprime explicitement
- Les utilisateurs réguliers sont limités par le **SCC/restricted SCC** dont les attributs pertinents sont fixés aux valeurs suivantes :

allowCapabilities : null

defaultAddCapabilities : null

requiredDropCapabilities:

KILL

- MKNOD

- SETUID

- SETGID

# CRIO - Container Sécurité

CONFIDENTIAL Designator

- Avec un user **non cluster admin** programmons un pod qui supprime la capabilities **CAP\_SYS\_CHOWN**. Même si le pod fonctionne en tant que root, il n'est pas autorisée à changer la propriété du répertoire **/tmp**. En utilisant une définition de pods telle que celle-ci

```
$ oc create -f pod-drop-caps.yaml
```

```
pod/caps-test created
```

```
$ oc logs caps-test
```

```
uid=0(root) gid=0(root) groups=0(root)
```

```
chown: changing ownership of '/tmp': Operation not permitted
```

```
drwxrwxrwt. 1 root root 6 Mar 31 14:54 /tmp
```

```
apiVersion: "v1"
kind: Pod
metadata: [REDACTED]
  name: caps-test1
spec:
  containers:
    - name: caps-container
      image: registry.access.redhat.com/ubi8/ubi
      command: ["/bin/sh"]
      args: ["-c", "chown bin.bin /tmp; ls -ld /tmp"]
      securityContext:
        capabilities:
          drop:
            - CHOWN
  restartPolicy: Never
```

# CARIO - Container Sécurité

CONFIDENTIAL Designator

- Cela signifie que l'utilisateur contraint par le **SCC restricted** ne peut pas obtenir de nouvelles capacités. Essayons cela en lançant un pod qui gagnerait la capability **CAP\_SYS\_ADMIN**, devenant essentiellement root.

```
$ oc create -f pod-add-caps.yaml
```

```
Error from server (Forbidden): error when creating "pod-addcaps.yaml": pods "caps-test" is forbidden: unable to validate against any security context constraint: [capabilities.add: Invalid value: "SYS_ADMIN": capability may not be added]
```

```
kind: Pod
apiVersion: v1
metadata:
  name: caps-test
spec:
  containers:
    - name: caps-container
      command:
        - /bin/sh
      securityContext:
        capabilities:
          add:
            - SYS ADMIN
      image: registry.access.redhat.com/ubi8/ubi
      args:
        - '-c'
        - cat /proc/1/status
  restartPolicy: Never
```

# Secure Computing (seccomp)

The OpenShift operating  
system

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Secure Computing (seccomp) Profiles

- En général, un **container** ou un **pods** OpenShift fait tourner une seule application qui exécute une ou plusieurs tâches bien définies. Par conséquent, l'application ne nécessite généralement qu'un petit sous-ensemble des API du noyau du système d'exploitation sous-jacent
  - par exemple,
    - un container exécutant un serveur httpd n'a pas à appeler l'appel système mount
    - En même temps, tous les containers sur l'hôte, qui peuvent exécuter de nombreuses applications différentes, partagent le même noyau qui est utilisé par l'hôte lui-même (en utilisant l'exemple précédent, l'hôte a sûrement besoin de l'appel système mount) et aurait accès à l'ensemble de l'API du noyau.
- Pour limiter le vecteur d'attaque d'un processus détourné s'exécutant dans un container, la fonction du noyau **Linux seccomp** (secure computing mode) peut être utilisée pour limiter le processus s'exécutant dans un container à n'appeler qu'un sous-ensemble des appels système disponibles.
  - Podman et OpenShift proposent tous deux des profils seccomp par défaut qui sont utilisés pour un container, sauf indication contraire.
  - Les profils par défaut réduisent considérablement le nombre d'appels système disponibles, de plus de 300 disponibles dans le noyau Linux 5.x, à environ la moitié. Ce nombre est toujours supérieur à celui qu'une application typique utiliserait.

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Secure Computing (seccomp) Profiles

- Par exemple:
  - Avec **Podman** sur un hôte RHEL, la politique seccomp par défaut est stockée dans **/usr/share/containers/seccomp.json**, éventuellement remplacée par **/etc/containers/seccomp.json**. Notre application d'exemple sera juste **ls /**, qui peut être lancée avec **podman** en exécutant **fedora:latest ls /**.
  - Dans un deuxième temps, nous allons modifier la politique par défaut de **seccomp** et supprimer l'appel système **stat()**, qui est ce que **ls** a utilisé pour connaître les détails du répertoire **/**.
  - Copiez le fichier **/usr/share/containers/seccomp.json** dans un autre emplacement, supprimez l'appel "**stat**", et exécutez à nouveau l'invocation de podman, cette fois-ci en pointant sur la politique modifiée :

```
$ cp /usr/share/containers/seccomp.json /tmp/seccomp.json  
$ vim /tmp/seccomp.json # remove the stat call  
$ podman run --security-opt seccomp=/tmp/seccomp.json -it fedora:latest ls /
```

- Cette commande échouerait maintenant parce que le nouveau profil seccomp ne permet plus au binaire **ls** d'exécuter l'appel système **stat**. Même si cet exemple est négatif, nous avons apporté une application qui fonctionne bien et l'avons brisée avec une politique trop restrictive, et nous espérons qu'il illustre le point :

```
$ podman run --security-opt seccomp=/tmp/seccomp.json -it fedora:latest ls /  
ls: cannot access '/': Operation not permitted
```



# CRIO - Container Sécurité

CONFIDENTIAL Designator

- Dans OpenShift, tout pod annoté avec **seccomp.security.alpha.kubernetes.io/pod** utiliserait le profil seccomp spécifié en valeur de l'annotation.
- Les annotations sont vérifiées par le contrôleur d'admission du SCC par rapport au SCC actuel lié au rôle de l'utilisateur actuel. Si le profil seccomp est autorisé pour cet utilisateur, le kubelet exécute le pod sur un nœud avec le profil seccomp spécifié. Notez que le fichier de définition du profil doit exister sur le nœud où le pod est programmé.

Pour mettre tout cela ensemble, afin de programmer un pod avec un profil seccomp personnalisé, il faut :

- S'assurer que la politique seccomp existe sur le nœud. Le répertoire par défaut pour les profils seccomp se trouve dans `/var/lib/kubelet/seccomp/`
- Assurez-vous que le profil est autorisé dans un SCC lié au rôle de l'utilisateur qui programme les pods
- Annotez les pods avec le profil seccomp souhaité

# CRI-O - Container Sécurité

CONFIDENTIAL Designator

- Essayons le même exemple que nous avons montré plus tôt avec **podman**, cette fois dans un environnement OpenShift. Copiez le profil seccomp CRI-O par défaut (/etc/crio/seccomp.json) de l'un des nœuds de clusters vers votre système local. Pour ce faire, ouvrez une session de débogage, puis copiez le fichier du pod de débogage sur votre système local :

```
$ oc get node  
$ oc debug node/ip-10-0-164-156.ec2.internal
```

La session de débogage étant toujours active, connectez-vous au cluster à partir d'un second shell (oc login), puis copiez le fichier dans votre système local :

```
$ oc project default  
$ oc cp default/ip-10-0-151-213us-east-2computeinternal-debug:/host/etc/crio/seccomp.json $HOME/seccomp.json  
$ cp seccomp.json seccomp-nostat.json  
$ vim seccomp-nostat.json # remove the "stat" system call  
$ diff -u seccomp.json seccomp-nostat.json  
--- seccomp.json 2020-04-09 21:11:56.410394097  
+0200  
+++ seccomp-nostat.json 2020-04-09 19:57:20.113819802 +0200  
@@ -318,7 +318,6 @@  
"socketcall",  
"socketpair",  
"splice",  
- "stat",  
"stat64",
```

# CARIO - Container Sécurité

CONFIDENTIAL Designator

- En utilisant MachineConfig, téléchargez le fichier dans /var/lib/kubelet/seccomp/seccomp-nostat.json. Suivez la documentation de Machine Config Operator, remplacez simplement le contenu et les chemins d'accès.

```
cat <<EOF >machineconfiguration.ymal
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 50-example-seccomp-nostat
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-8;base64,$(cat seccomp-nostat.json | base64 -w0)
            filesystem: root
            mode: 0644
            path: /var/lib/kubelet/seccomp/seccomp-nostat.json
EOF
oc apply -f ./machineconfiguration.ymal
```



# CRIO - Container Sécurité

CONFIDENTIAL Designator

- Une fois que le fichier est là, créez un pods en utilisant un profil tel que celui ci-dessous :

Si le pods est exécuté maintenant, la commande ls échouera, car il n'est pas autorisé à exécuter l'appel système stat(1) :

```
apiVersion: "v1"
kind: Pod
metadata:
  name: seccomp-test
  annotations:
    seccomp.security.alpha.kubernetes.io/pod
      "localhost/seccomp-nostat.json"
spec:
  containers:
    - name: seccomp-container
      image: registry.access.redhat.com/ubi8/ubi
      command: ["/bin/ls", "/"]

  restartPolicy: Never
```

1 line

```
ls: cannot access '/': Operation not permitted
```



# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Secure Computing (seccomp) Profiles

- La création des profils de seccomp pour une application peut être fastidieuse et nécessite souvent une connaissance approfondie de l'application en plus du langage ou du framework de programmation.
  - Par exemple, le développeur doit être conscient qu'un framework qui met en place un serveur réseau pour accepter les connexions se traduirait par l'appel des appels système socket(2), bind(2) et listen(2).
  - D'une part, écrire une politique seccomp trop lâche pourrait laisser certains appels système ouverts.
  - D'autre part, écrire une politique seccomp trop stricte pourrait faire échouer l'application avec des erreurs telles que "Permission refusée".
  - Comme la plupart des profils seccomp du monde réel seraient probablement un peu trop lâches, il est important de mettre en place des protections de sécurité par couches pour s'assurer que si un acteur malveillant contourne une couche, comme seccomp, il sera arrêté par une autre couche, comme les capacités ou SELinux.
- Malheureusement, il n'existe actuellement aucun outil ou opérateur fourni par OpenShift qui aiderait à développer les profils seccomp.
- Il existe des outils tiers, ou bien le développeur pourrait vouloir retracer les appels effectués par son application dans un pipeline de CI grâce à des outils tels que eBPF et augmenter continuellement sa politique seccomp. Un exposé de Dan Walsh sur le sujet présente quelques-unes des options disponibles.

# SE-LINUX

The OpenShift operating  
system

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

- **Security-Enhanced Linux** a commencé comme un projet au sein de l'Agence de sécurité nationale (NSA) pour apporter le contrôle d'accès obligatoire (MAC) à Linux. Depuis son introduction, le développement ultérieur de SELinux a été un effort conjoint entre la NSA, Red Hat et la communauté des développeurs SELinux. Étant donné la disponibilité de nombreuses autres installations qui peuvent améliorer la sécurité et l'isolation des containers, SELinux se présente comme un composant intégral qui garantit que les containers adoptent la posture de contrôle d'accès obligatoire. À cet égard, SELinux améliore l'expérience de la sécurité des containers en séparant véritablement les containers les uns des autres tout en protégeant l'hôte de manière intuitive.
- **Concepts de base**
  - Comme d'autres auteurs l'ont déjà mentionné, **SELinux** est un système de labellisation. Cela signifie que tout ce qui se trouve dans Linux a un label :
    - Processes
    - Files
    - Directories
    - System objects

Les règles de politique contrôlent l'accès entre les processus et les objets, par exemple un processus appelé container\_t peut accéder à des fichiers appelés container\_file\_t. Lorsque SELinux est activé, tout est refusé, sauf si une règle l'autorise explicitement. Cela constitue une politique de sécurité très solide. Les labels des objets sont appelées contextes SELinux, et les contextes eux-mêmes sont composés de l'utilisateur, du rôle, du type et du niveau de sécurité. Il faut garder à l'esprit, lors de l'écriture des politiques, qu'il est normal de spécifier qu'un certain type SELinux peut interagir avec un autre type SELinux. Il n'est pas courant de spécifier le contexte complet d'un objet. Ainsi, un container s'exécutant sur un nœud aurait un contexte SELinux comme suit : System\_u :system\_r:container\_t:s0:c667,023 Red Hat

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

- Les règles de politique contrôlent l'accès entre les processus et les objets, par exemple un processus appelé **container\_t** peut accéder à des fichiers appelés **container\_file\_t**.
- Lorsque SELinux est activé, tout est refusé, sauf si une règle l'autorise explicitement. Cela constitue une politique de sécurité très solide. Les labels des objets sont appelées contextes SELinux, et les contextes eux-mêmes sont composés de l'utilisateur, du rôle, du type et du niveau de sécurité. Il faut garder à l'esprit, lors de l'écriture des politiques, qu'il est normal de spécifier qu'un certain type SELinux peut interagir avec un autre type SELinux. Il n'est pas courant de spécifier le contexte complet d'un objet. Ainsi, un container s'exécutant sur un nœud aurait un contexte SELinux comme suit : **System\_u :system\_r:container\_t:s0:c667,c123**

ou:

- **system\_u** est le user
- **system\_r** est le role
- **container\_t** est le type de ; on parle aussi parfois de domaine
- **S0 :c667, c123** est le niveau de sécurité ou le label MCS

## Domaines non limités

- Normalement, dans les systèmes Red Hat, chaque paquet RPM fourni est accompagné d'une politique SELinux qui permet au processus de se dérouler en toute sécurité. Si vous installez un binaire qui n'est normalement pas fourni avec une politique SELinux, exécutez ce binaire avec un domaine non confiné (**unconfined\_t**). Ce type est autorisé à faire presque tout ce qu'il faut dans le système et ne doit normalement pas être utilisé pour les démons. Au lieu de cela, il est censé permettre aux utilisateurs d'interagir avec le système sans que SELinux ne les bloque.

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

### SELinux en containers

- Bien que SELinux n'ait pas été conçu spécifiquement pour assurer la sécurité des containers, des travaux ont déjà été réalisés sur la sécurisation des machines virtuelles. Des concepts similaires ont été mis en œuvre et adaptés pour sécuriser les containers. Une chose importante à considérer est que tous les containers sont démarrés, par défaut, avec l'étiquette, **container\_t**. Il s'agit d'une étiquette prédéfinie à laquelle est déjà attachée une politique. Il est déjà très contraint, ce qui en fait un excellent défaut !

Pour un usage pratique, les containers avec le label **container\_t** peuvent :

- **Read, write, et execute** des fichiers étiquetés **container\_file\_t**
  - **Read, write, et execute** des fichiers étiquetés **usr\_t**, qui sont des fichiers dans /usr
  - **Read** les fichiers étiquetés **etc\_t**, qui sont des fichiers sous /etc
  - **Bind** à tout port inutilisé sur l'hôte
- Il existe d'autres capacités et détails pour ce label, mais il est préférable de les laisser en tant que détails de mise en œuvre.

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

- Normalement, tout processus s'exécutant sous le nom de **container\_t** devrait pouvoir interagir avec tout fichier étiqueté **container\_file\_t**; ce comportement impliquerait alors que les containers peuvent interagir ou s'immiscer dans les fichiers les uns des autres. Il serait alors possible pour un container de manipuler les données sensibles d'un autre container, ce qui n'est pas souhaitable. Heureusement, il existe un autre mécanisme qui protège les containers contre les attaques mutuelles, et ce sont les étiquettes **MCS**.
- **MCS** signifie Multi-Category Security et vient de Multi-Level Security. Concrètement, chaque container se voit attribuer sa propre étiquette MCS, et donc eux seuls peuvent accéder aux objets ayant une étiquette MCS correspondante. Ils peuvent cependant accéder à des objets qui n'ont pas de jeu de catégories.

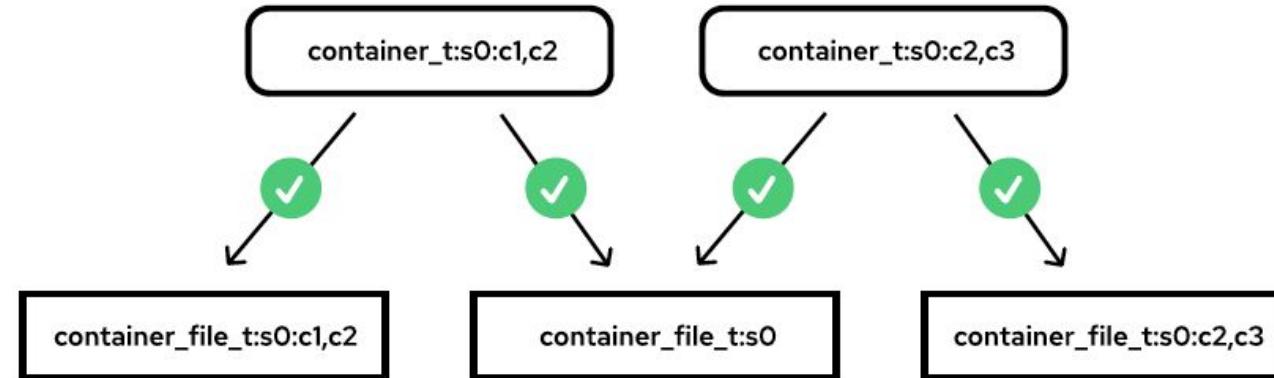
# CRIOL - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

Prenons un exemple rapide :

Multi-Category Security Labeling for Containers



Dans ce cas, nous avons deux containers :

- Les deux containers utilisent l'étiquette **container\_t** par défaut
- Ils essaient tous deux d'accéder à des fichiers portant l'étiquette **container\_file\_t**
- **container\_t:s0:c1,c2** a un label MCS correspondant comme **container\_file\_t:s0:c1,c2** lui donnant accès
- **container\_t:s0:c1,c2** ne peut pas accéder **container\_file\_t:s0:c2,c3** parce que le label MCS ne correspond pas
- Les deux **containers\_t:s0:c1,c2** et **container\_t:s0:c1,c2** peuvent accéder **container\_file\_t:s0** puisque ce fichier n'a pas d'ensemble de catégories. Dans un tel cas, le dossier peut être partagé entre les containers

Red Hat

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

### Podman

- Consolidons les concepts que nous avons exposés il y a peu de temps.

Pour savoir si une installation podman supporte le label SELinux, procédez comme suit :

```
$ sestatus
```

```
SELinux status:          enabled
```

```
...
```

```
$ podman run -ti registry.access.redhat.com/ubi8/ubi /bin/bash
```

```
$ podman ps
```

```
$ podman inspect 3fdf4f63ab82 | jq ".[0].MountLabel"
```

```
"system_u:object_r:container_file_t:s0:c116,c225"
```

```
$ podman inspect 3fdf4f63ab82 | jq ".[0].ProcessLabel"
```

```
"system_u:system_r:container_t:s0:c116,c225"
```

```
$ podman inspect 3fdf4f63ab82 | jq -r ".[0].State.Pid" |
```

```
..
```

```
system_u:system_r:container_t:s0:c116,c225 777931 pts/0 Ss+ 0:00 /bin/bash
```

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

- Créons maintenant un container qui lie - monte un répertoire qui est partagé entre les containers :

```
$ mkdir container-public
```

```
$ echo "This is accessible by all containers" > container-public/text.txt
```

```
$ podman run -ti -v=./container-public/:/public:z registry.access.redhat.com/ubi8/ubi /bin/bash
```

```
$ cat /public/text.txt
```

This is accessible by all containers

L'option -Z à la fin de la déclaration de volume demandait à podman de ré-étiqueter ce répertoire de manière à ce qu'il soit partagé entre les containers. Nous pouvons vérifier cela dans un autre terminal :

```
$ ls -Z container-public/
```

```
system_u:object_r:container_file_t:s0 text.txt
```

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

- Créons maintenant un container qui montera un répertoire qui est partagé entre les containers :

```
$ mkdir container-public
```

```
$ echo "This is accessible by all containers" > container-public/text.txt
```

```
$ podman run -ti -v=./container-public/:/public:z registry.access.redhat.com/ubi8/ubi /bin/bash
```

```
$ cat /public/text.txt
```

This is accessible by all containers

L'option -Z à la fin de la déclaration de volume demandait à podman de ré-étiqueter ce répertoire de manière à ce qu'il soit partagé entre les containers. Nous pouvons vérifier cela dans un autre terminal :

```
$ ls -Z container-public/
```

```
system_u:object_r:container_file_t:s0 text.txt
```

# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

- Créons un **container A** qui montera un répertoire avec un label MCS **non partagé** :

```
$ mkdir container-private
```

```
$ echo "This is private" > container-private/text.txt
```

```
$ podman run -ti -v=./container-private:/private:Z registry.access.redhat.com/ubi8/ubi /bin/bash
```

```
$ cat container-private/text.txt
```

This is private

L'option -Z à la fin de la déclaration de volume demandait à podman de ré-étiqueter ce répertoire de manière à ce qu'il soit partagé entre les containers. Nous pouvons vérifier cela dans un autre terminal :

```
$ podman inspect e42033efb780 | jq ".[0].MountLabel"
```

```
"system_u:object_r:container_file_t:s0:c15,c221"
```

```
$ ls -Z container-private/
```

```
system_u:object_r:container_file_t:s0:c15,c221 text.txt
```

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

- Créons un **container A** qui montera un répertoire avec un label MCS **non partagé** :

```
$ podman run -ti -v=./container-private/:/private:Z registry.access.redhat.com/ubi8/ubi /bin/bash
```

```
$ cat container-private/text.txt
```

This is private

L'option -Z à la fin de la déclaration de volume demandait à podman de ré-étiqueter ce répertoire de manière à ce qu'il soit partagé entre les containers. Nous pouvons vérifier cela dans un autre terminal :

```
$ podman inspect bf0aae021337 | jq ".[0].MountLabel"  
"system_u:object_r:container_file_t:s0:c275,c578"
```

```
$ ls -Z container-private/  
system_u:object_r:container_file_t:s0:c275,c578 text.txt
```

Comme on peut le voir, Podman a relancé le répertoire, et il est maintenant privé au nouveau répertoire. Depuis que le répertoire a été relabelisé, le premier container (e42033efb780) n'a plus accès à ce fichier. Le rendant effectivement privé pour le deuxième container (bf0aae021337).

```
[root@e42033efb780 /]# cat private/text.txt  
cat: private/text.txt: Permission denied
```



# CRI-O - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

### Kubernetes

- Kubernetes a la possibilité d'attribuer des étiquettes **SELinux** aux containers. Cela fait partie de la spécification normale des pods et aucune extension spéciale n'est nécessaire pour cela.
- OpenShift, étant une distribution de Kubernetes, prendra en compte ces spécifications et elles seront exécutées par CRI-O qui est l'interface d'exécution des containers (CRI) qui est installée, par défaut, dans OpenShift.
- Les paramètres de **SELinux** se trouvent dans la section **SecurityContext** du fichier de spécification du pods. Les options SELinux pour le container peuvent être définies comme suit :

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: my-container
    image: registry.access.redhat.com/ubi8/ubi:latest
    command: ["/bin/bash"]
    args: ["-c", "while true; do sleep 2; done"]
    securityContext:
      seLinuxOptions:
        type: container_t
        level: s0:c275,c578
      volumeMounts:
      - name: priv-dir
        mountPath: /container-private
    restartPolicy: Never
    volumes:
    - name: priv-dir
      emptyDir: {}
```

# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

- Même si l'API le permet, il n'est pas nécessaire de définir les paramètres **user** et de **role** seLinuxOptions.

Dans ce cas, le type (domaine) **container\_t** spécifié est déjà la valeur par défaut utilisée par le CRI-O.  
Il est possible de spécifier tout autre type, mais il doit être installé sur l'hôte.

Nous avons spécifié un niveau (label **MCS**) de **s0:c275,c578** qui rend effectivement le volume privé pour ce container spécifique.

Pour utiliser un contexte spécifique, étant donné la nature déclarative de Kubernetes, dites explicitement à Kubernetes d'utiliser ce contexte.

- Creusons un peu ce qui se passe ici ! Et, pour cela, nous devons nous connecter à l'hôte et vérifier :

```
$ crictl ps --name my-container
$ crictl inspect <pid> | grep empty-dir
$ cd /var/lib/kubelet/pods/3a5c223e-ce0b-4fe9-adae-bde73683b16f/volumes/kubernetes.io~empty-dir/priv-dir
$ ls -alZ
drwxrwxrwx. 2 root root system_u:object_r:container_file_t:s0:c275,c578 22 Jun 12 10:42
...
...
```

Ce que nous avons fait ici, c'est rechercher notre container fonctionnant sur le système, vérifier le répertoire créé pour remplir ce montage de volume, et vérifier que le répertoire a le label MCS que nous avons spécifié dans le paramètre de niveau.



# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

- Il y a une mise en garde pour relabeliser les volumes SELinux dans Kubernetes. Étant donné que des opérations sur l'hôte pourraient potentiellement endommager le déploiement, le CRI ne tentera pas de relabeliser un hostPath.

Si nous voulons qu'un répertoire soit partagé dans des containers, nous pouvons le spécifier comme suit :

```
seLinuxOptions :  
  level: s0:
```

- OpenShift offre des niveaux supplémentaires de sécurité et d'automatisation en ce qui concerne l'utilisation des labels SELinux. Cela dépend du **Security Context Constraint** que le **compte de service** a utilisé pour créer le **pod** qu'il est capable d'utiliser.

# CRI0 - Container Sécurité

CONFIDENTIAL Designator

## Security-Enhanced Linux (SE Linux)

### Comment SELinux me protège-t-il des attaques ?

Bien que SELinux ne soit pas inclus dans certaines distributions de Kubernetes, son inclusion dans OpenShift a permis d'assurer une protection contre plusieurs CVE importants. Par exemple :

- CVE-2015-3627 - Ouverture non sécurisée du descripteur de fichier 1 entraînant une escalade des privilèges
- CVE-2015-3630 - Les chemins d'accès en lecture/écriture permettent de modifier l'hôte et la divulgation d'informations
- CVE-2015-3631 - Les montages en volume permettent les RMLL - escalade de profil
- CVE-2016-9962 - Vulnérabilité de l'exécution du RunC
- CVE-2019-5736 - L'exécution de containers malveillants permet la fuite de containers et l'accès au système de fichiers hôte

La raison pour laquelle il a protégé l'hôte dans de si nombreux cas est le système d'étiquetage lui-même. Les containers fonctionnent avec un domaine container\_t. Même si un container s'échappait (et il le peut), le container\_t n'a pas la permission d'accéder à d'autres fichiers sur l'hôte, donc toute action que l'attaquant pourrait essayer d'exécuter sur d'autres étiquettes sera bloquée.



# Security Contexts Constraints (SCC)

The OpenShift operating  
system

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security Contexts Constraints

- Dans Red Hat OpenShift Container Platform, **Security Context Constraints (SCC)** limitent les privilèges des **pods**.
- Cela permet de renforcer ou d'assouplir leurs **capabilités**, garantissant ainsi que les applications en cours d'exécution n'utilisent pas de privilèges dont elles n'ont pas besoin.
- Les **SCC** sont similaires aux politiques qui imposent certaines actions ou en empêchent d'autres de la part d'un service ou d'un utilisateur. L'utilisation des **SCC** permet de contrôler précisément le niveau de privilèges pour l'application et les applications de l'utilisateur et, si nécessaire, de leur accorder des privilèges plus permissifs ou plus restrictifs.
- Les **SCC** sont des ressources **OpenShift** ;
  - Ils définissent un ensemble de conditions (ou règles) auxquelles un pod doit satisfaire pour être créé (ou admis dans le cluster).
  - Une plateforme de containers OpenShift est livrée avec un ensemble de **SCC**, allant d'une politique restrictive qui restreint l'utilisateur root dans les pods et supprime certaines capacités Linux, à une politique plus permissive qui autorise l'utilisateur root dans les pods, l'utilisation de tout UID et GID, et l'accès au système de fichiers hôte du container.
- Par défaut, pour les utilisateurs authentifiés, les ressources déployées dans un projet héritent du contexte de sécurité **restreint**, qui empêche les applications de s'exécuter en tant que root et d'escalader les privilèges (**allowPrivilegeEscalation**).
  - Ce contexte de sécurité bloque également certaines capacités, telles que **mknod** ou **setuid**, ce qui augmente encore la sécurité de l'environnement.
  - Si un attaquant réalise un exploit et sort du container, il n'a toujours pas accès à l'hôte du container en tant que root.

# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Security Contexts Constraints

- Utilisez les **SCC** pour gérer les paramètres de sécurité suivants :

### Privilege mode

- Ce paramètre permet ou empêche le fonctionnement du container en mode privilégié, c'est-à-dire l'accès aux ressources hôtes sous-jacentes du container, telles que les périphériques matériels. Ce mode contourne de nombreuses restrictions, telles que les **Cgroups**, **Linux capabilities**, **seccomp profiles**, et l'usage spécifique de **users** ou **group IDs**.

### Privileges escalation

- Ce paramètre permet d'activer ou de désactiver l'escalade des privilèges à l'intérieur d'un container (le drapeau `allowPrivilegeEscalation`).

### Capacités Linux

- Ce paramètre permet d'ajouter ou de supprimer des capacités Linux dans les containers. Par exemple, en supprimant la capacité KILL qui empêche un utilisateur de tuer des processus qu'il ne possède pas.

# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Security Contexts Constraints

### Seccomp profiles

- Ce paramètre permet de spécifier quels profils Seccomp sont autorisés à être utilisés par le pods. Les profils Seccomp autorisent ou bloquent à leur tour certains appels système. Par exemple, bloquer l'appel système de montage pour empêcher le container de tenter de monter des répertoires s'il n'en a pas besoin.

### Volume types

- Ce paramètre permet d'autoriser ou d'empêcher l'accès à certains types de volumes par les applications ; cela inclut les volumes persistants, les cartes de configuration et les répertoires temporaires (emptyDir).

### System control (Sysctl) settings

- L'interface Sysctl permet de modifier les paramètres du noyau au moment de l'exécution. Elle permet de régler le noyau en temps réel. Utilisez ce paramètre SCC pour autoriser ou empêcher l'exécution de certains contrôles du système.

### Host resources

- Cet ensemble de paramètres permet d'autoriser ou d'empêcher l'accès d'un pods aux ressources hôtes suivantes : namespaces IPC, réseaux hôtes, ports hôtes et espaces de noms PID hôtes.

# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Security Contexts Constraints

### Read-only root system

- Ce paramètre permet de rendre le système de fichiers racine en lecture seule. Cela empêche toute modification et oblige les utilisateurs à monter un volume s'ils ont besoin de stocker des données.

### User and group IDs

- Ces paramètres permettent de restreindre l'attribution des ID d'utilisateurs et de groupes à une plage spécifique. Ce sont des paramètres utiles pour limiter les utilisateurs à un certain ensemble d'ID ou de GID.

### SELinux labels

- Ce paramètre permet de définir un label SELinux pour les pods. OpenShift supporte tous les champs SELinux disponibles : utilisateur, rôle, type et niveau.

### File system groups

- Ce paramètre permet de définir des groupes supplémentaires pour l'utilisateur, ce qui est généralement nécessaire pour accéder à un dispositif de blocage. Cela permet de fournir un accès adéquat au stockage en bloc, tel que Ceph RBD, et iSCSI.

# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security Contexts Constraints

Les **SCC** sont des ressources OpenShift qui peuvent être listées avec la commande

**\$ oc get scc**

NAME	AGE
anyuid	9d
hostaccess	9d
hostmount-anyuid	9d
hostnetwork	9d
node-exporter	9d
nonroot	9d
privileged	9d
restricted	9d

**\$ oc describe scc restricted**

```
Name: restricted
Priority: <none>
Access:
  Users: <none>
  Groups: system:authenticated
Settings:
  Allow Privileged: false
  Allow Privilege Escalation: true
  Default Add Capabilities: <none>
  Required Drop Capabilities: KILL,MKNOD,SETUID,SETGID
  Allowed Capabilities: <none>
  Allowed Seccomp Profiles: <none>
  Allowed Volume Types: configMap,downwardAPI,emptyDir,persistentVolumeClaim,projected,secret
  Allowed Flexvolumes: <all>
  Allowed Unsafe Sysctls: <none>
  Forbidden Sysctls: <none>
  Allow Host Network: false
  Allow Host Ports: false
  Allow Host PID: false
  Allow Host IPC: false
  Read Only Root Filesystem: false
  Run As User Strategy: MustRunAsRange
    UID: <none>
    UID Range Min: <none>
    UID Range Max: <none>
  SELinux Context Strategy: MustRunAs
    User: <none>
    Role: <none>
    Type: <none>
    Level: <none>
  FSGroup Strategy: MustRunAs
    Ranges: <none>
  Supplemental Groups Strategy: RunAsAny
    Ranges: <none>
```

# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Security Contexts Constraints

La Security Context Constraints (**SCC**) permet de définir ce qu'un **POD** a le droit ou n'a pas le droit de faire. Pour rappel, un POD est un ensemble de containers (1 à n) partageant le même réseau (communication en localhost) et les mêmes volumes, comme par exemple un pod contenant un container Apache et un container Tomcat.

### SCC restricted

- Le **SCC restricted** est associé par défaut à tous les POD. Cela implique qu'ils utilisent la stratégie **MustRunAsRange** pour l'utilisateur de démarrage des containers.
- Cette stratégie autorise uniquement une **plage de UID permettant de démarrer les containers**.
- Le SCC ne définit pas cette plage car UID Range Min et Max du "Run As User" est défini à none.
- En fait, la plage est définie par le projet

```
$ oc describe scc restricted
```

<b>Name:</b>	restricted
<b>Priority:</b>	none
<b>Access:</b>	none
<b>Users:</b>	system:authenticated
<b>Groups:</b>	
<b>Settings:</b>	
<b>Allow Privileged:</b>	false
<b>Default Add Capabilities:</b>	none
<b>Required Drop Capabilities:</b>	KILL,MKNOD,SETUID,SETGID
<b>Allowed Capabilities:</b>	none
<b>Allowed Seccomp Profiles:</b>	none
<b>Allowed Volume Types:</b>	configMap,downwardAPI,emptyDir,persistentVolumeClaim,projected,secret
<b>Allowed Flexvolumes:</b>	all
<b>Allow Host Network:</b>	false
<b>Allow Host Ports:</b>	false
<b>Allow Host PID:</b>	false
<b>Allow Host IPC:</b>	false
<b>Read Only Root Filesystem:</b>	false
<b>Run As User Strategy:</b>	<b>MustRunAsRange</b>
<b>UID:</b>	none
<b>UID Range Min:</b>	none
<b>UID Range Max:</b>	none
<b>SELinux Context Strategy:</b>	<b>MustRunAs</b>
<b>User:</b>	none
<b>Role:</b>	none
<b>Type:</b>	none
<b>Level:</b>	none
<b>FSGroup Strategy:</b>	<b>MustRunAs</b>
<b>Ranges:</b>	none
<b>Supplemental Groups Strategy:</b>	<b>RunAsAny</b>
<b>Ranges:</b>	none

```
$ oc describe project hello-world
```

<b>Name:</b>	hello-world
<b>Annotations:</b>	openshift.io/description=Hello World openshift.io/display-name=hello-world openshift.io/sa.scc.mcs=s0:c24,c19 openshift.io/sa.scc.supplemental-groups=1000290000/10000 openshift.io/sa.scc.uid-range=1000290000/10000
<b>Display Name:</b>	hello-world
...	



# CARIO - Container Sécurité

CONFIDENTIAL Designator

## Security Contexts Constraints

- L'annotation **openshift.io/sa.scc.uid-range** permet d'indiquer avec quel **UID** les containers vont et peuvent être démarrés. Par défaut, tous les containers vont être démarrés avec l'**UID 1000290000**.  
Il est possible de surcharger les **POD** du projet pour qu'ils soient démarrés avec un autre UID.  
Par contre, l'**UID** doit être compris entre **1000290000** et **1000290000+10000-1=1000299999**.
- Pour quelles raisons OpenShift utilise-t-il des UID différents par projet ?  
L'une d'entre elles est que l'écriture dans les volumes soit effectuée avec des UID différents afin qu'un projet ne puisse pas lire le volume d'un autre.

```
$ oc describe project hello-world

Name:          hello-world
Annotations:   openshift.io/description=Hello World
                openshift.io/display-name=hello-world
                openshift.io/sa.scc.mcs=s0:c24,c19
                openshift.io/sa.scc.supplemental-groups=1000290000/10000
                openshift.io/sa.scc.uid-range=1000290000/10000
Display Name:  hello-world
...
```

```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000290001
  containers:
```

# CRIO - Container Sécurité

CONFIDENTIAL Designator

## Security Contexts Constraints

- Les **POD** surchargés avec un **UID** en dehors de la plage tomberont en erreur. De même les images définies avec le **USER root (UID=1)** dans le **Dockerfile** ne fonctionneront pas.
- Concernant la stratégie des **Supplemental Groups** qui est définie à **RunAsAny**. Cela signifie qu'il n'y aucune restriction sur les GID de groupe. On peut par exemple ajouter le groupe `nfsnobody(65534)` pour permettre aux POD d'accéder aux volumes NFS.
- **SCC anyuid**
  - Pour pouvoir utiliser une image utilisant le **USER** défini dans le **Dockerfile**, il faudrait utiliser le **SCC anyuid**, ce qui est déconseillé, c'est pour cela qu'il est préférable d'aborder cette problématique sous un angle différent comme par exemple refaire le Dockerfile.
- **SCC privileged**
  - Ce SCC permet de démarrer les containers avec le mode **--privileged=true et --cap-add=ALL**.

# Machine Config

The OpenShift operating system

# Machine Config Operator

CONFIDENTIAL Designator

- L'opérateur **Machine Config Operator** permet de configurer le cluster OCP, permet des mises à niveau en mode rolling upgrade et empêche la dérive entre les nouveaux nœuds et les nœuds existants. Le MCO est le cœur de ce qui fait de RHCOS un système d'exploitation natif pour kubernetes.

La configuration du système d'exploitation est stockée et appliquée à l'ensemble du cluster via l'opérateur **Machine Config Operator**

- Sous-ensemble de modules ignition applicable après le provisionnement
  - SSH keys
  - Files
  - systemd units
  - kernel arguments
- Standard k8s YAML/JSON manifests
- L'état souhaité des nœuds est vérifié/réparé régulièrement
- Peut être mis en pause pour suspendre les opérations

```
# test.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: test-file
spec:
  config:
    storage:
      files:
        - contents:
            source: data:,hello%20world%0A
            verification: {}
      filesystem: root
      mode: 420
      path: /etc/test
```

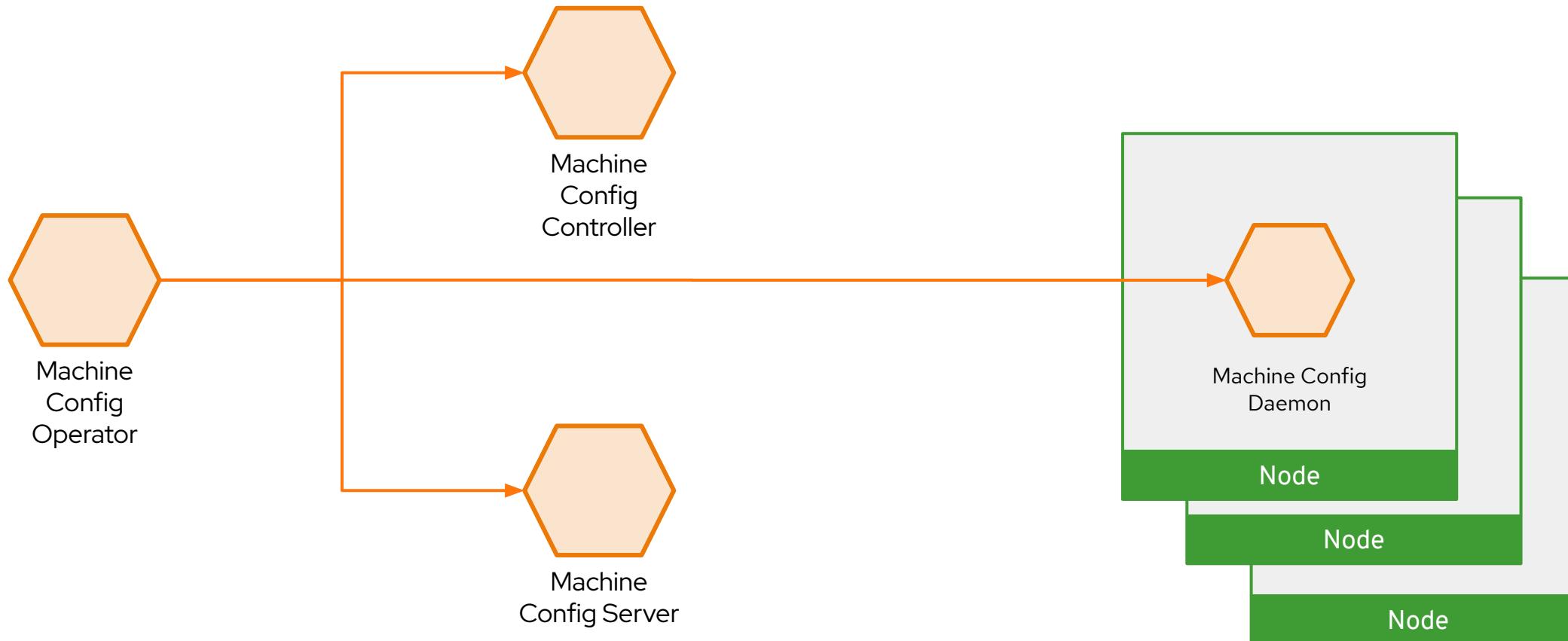
# Machine Config Operator

## Operator/Operand Relationships

CONFIDENTIAL Designator

L'opérateur Machine Config Operator est responsable du déploiement de trois principaux opérateur:

- le Machine Config Controller (MCC)
- le Machine Config Server (MCS)
- le Machine Config Daemon(MCD)



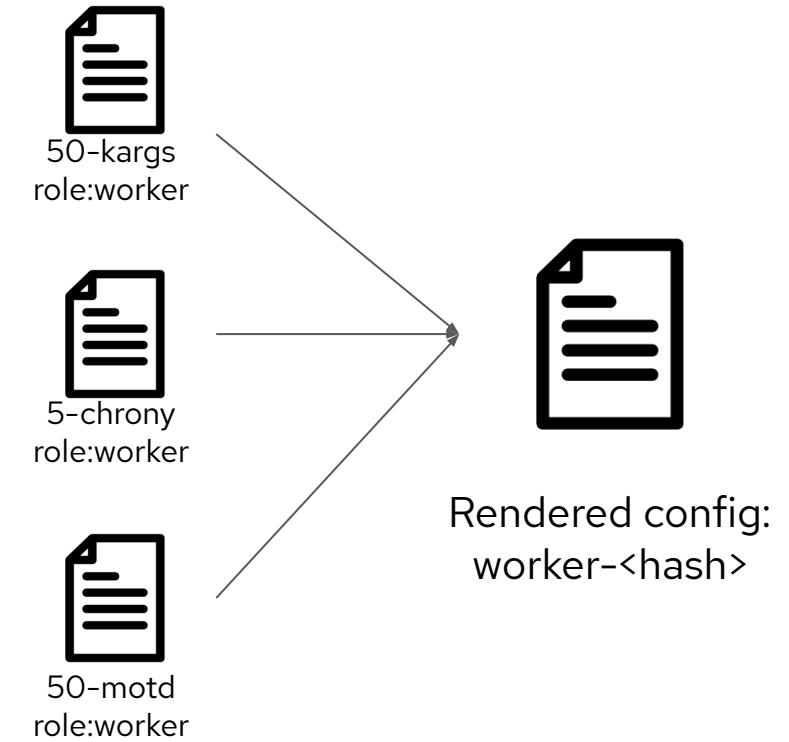
# Machine Config Operator

CONFIDENTIAL Designator

- **MachineConfig** sont convertis dans une configuration unique qui cible un **MachineConfigPool**, un groupe de système. C'est le travail du **Machine Config Controller**.
  - Les défaut **MachineConfigPool** sont **master & worker**
  - Des **MachineConfigPools** custom peuvent être créé au besoin.
- Un **MachineConfigPool** a deux choses qu'il cible : les nœuds pertinents et les objets MachineConfig pertinents.
  - Le MachineConfigPool possède un sélecteur de nœuds qui détermine à quels nœuds le pool s'applique.
  - Le MachineConfigPool a un MachineConfigSelector qui détermine les MachineConfigs qui appartiennent / sont pertinents.

## Machine Config and Machine Config Pool

Inheritance-based mapping of configuration to nodes



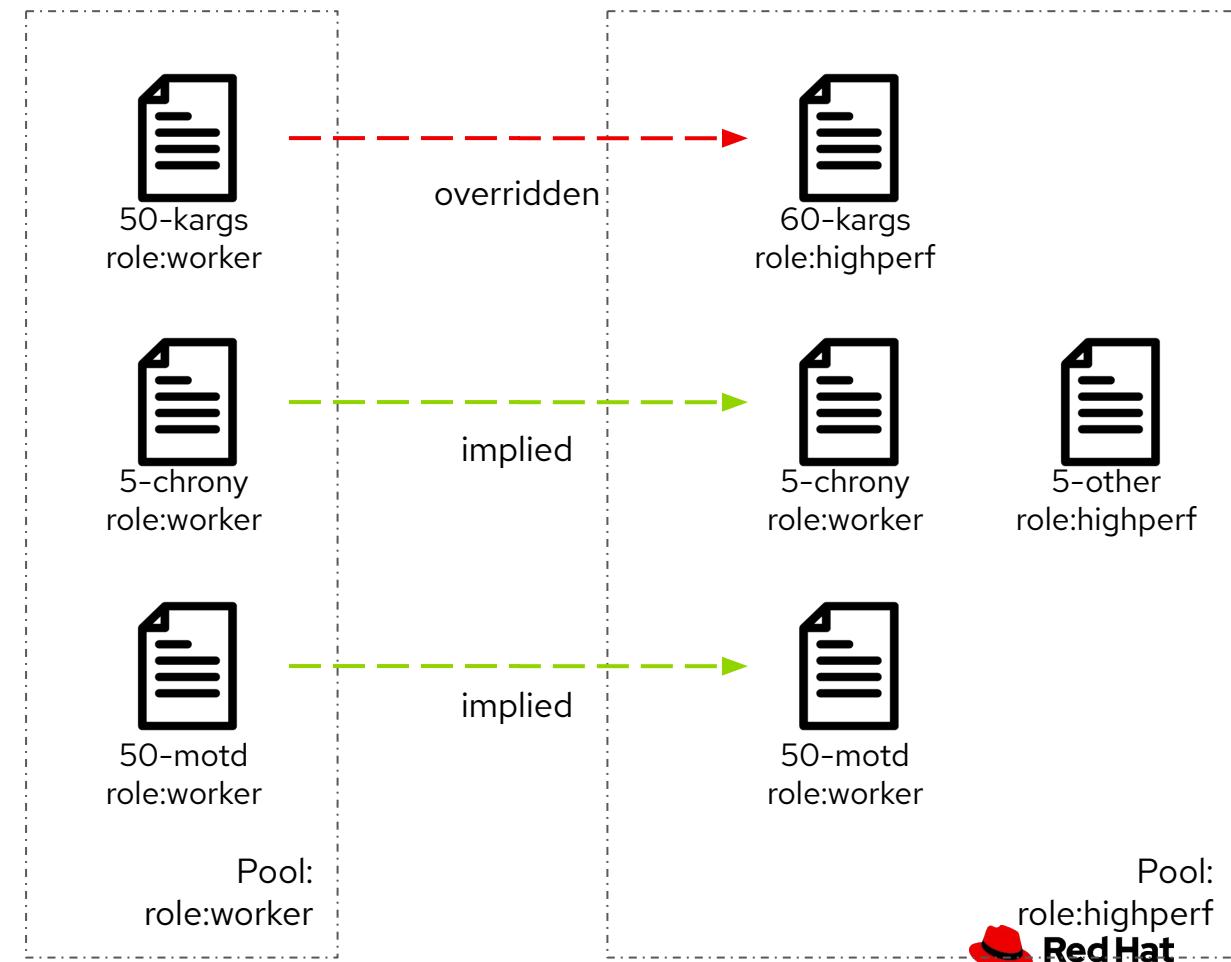
# Machine Config Operator

CONFIDENTIAL Designator

- Comme tous les nœuds doivent être des workers, les pools personnalisés héritent ou remplacent toute configuration de machine qui s'applique aux worker. Un pool personnalisé serait défini pour cibler des label de nœuds spécifiques et définirait les objets MachineConfig à prendre en compte.
- Dans cet exemple, tous les worker du cluster reçoivent un message du jour (MOTD), une configuration Chrony/NTP et quelques arguments du noyau. Dans le cas du Machine Config Pool personnalisé appelé "highperf", les arguments du noyau sont remplacés parce que le préfixe alphanumérique (60-) est plus grand/plus grand que le groupe des workers.  
Note : Les nœuds ne peuvent avoir qu'un seul MachineConfigPool personnalisé qui s'applique à eux.
- Le pool highperf a également un fichier unique, other, qui s'applique. Et, comme il n'y a pas de surcharge pour les configs chrony ou MOTD, les noeuds highperf recevront les versions par défaut de ces fichiers pour les workers.

## Custom Machine Config Pools

Hierarchical/layered configuration rendering



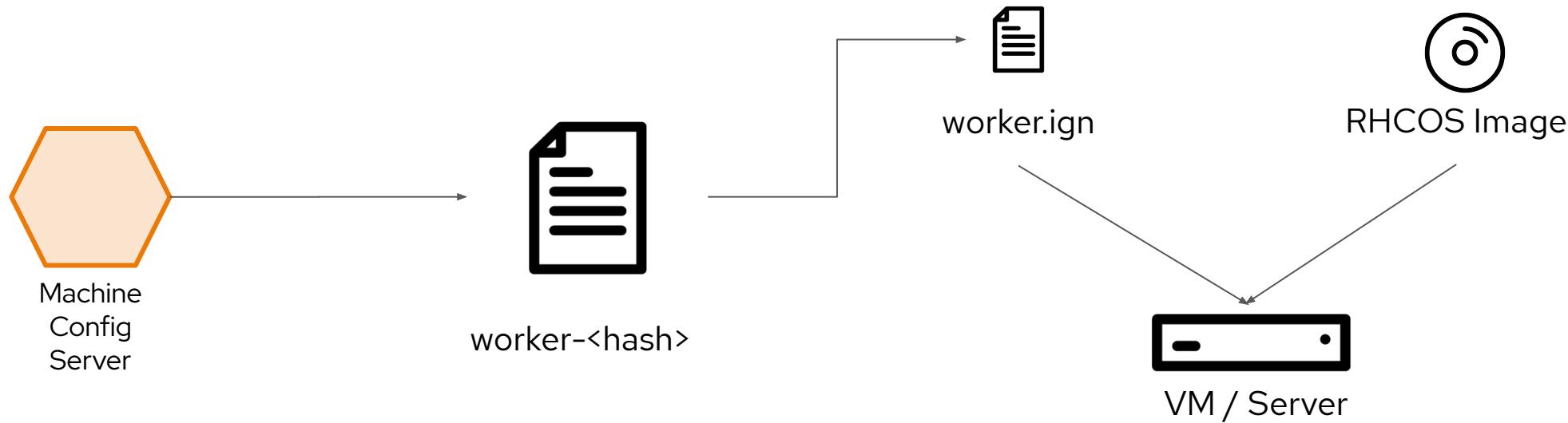
# Machine Config Operator

CONFIDENTIAL Designator

- **Ignition configs** fournira la **Machine Config Pool** correspondante lors du provisionnement de RHEL CoreOS. De cette façon, lorsque le nœud est provisionné, il a la configuration correcte dès le début.

## Machine Config Server

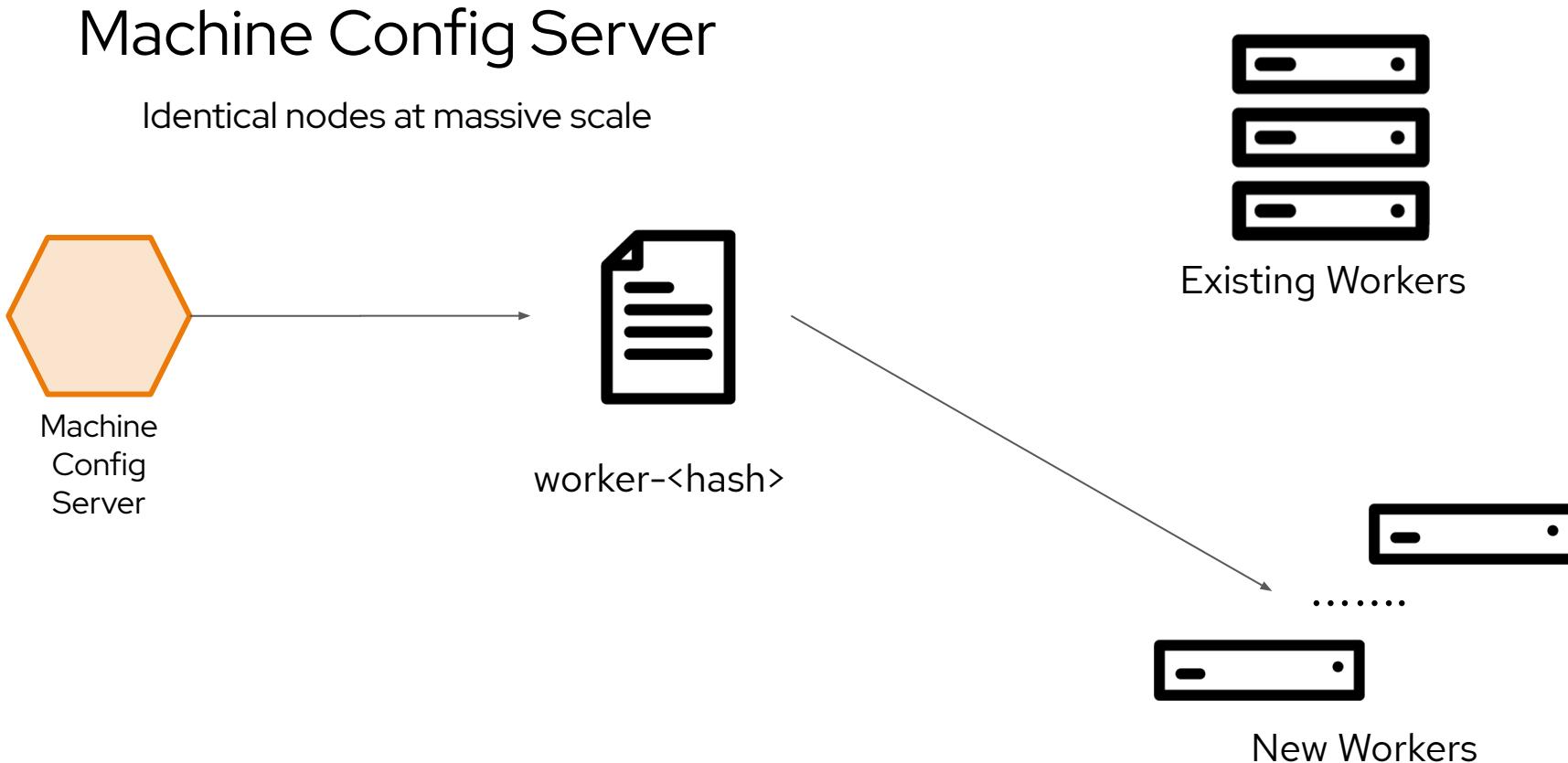
Providing Ignition configuration for provisioning



# Machine Config Operator

CONFIDENTIAL Designator

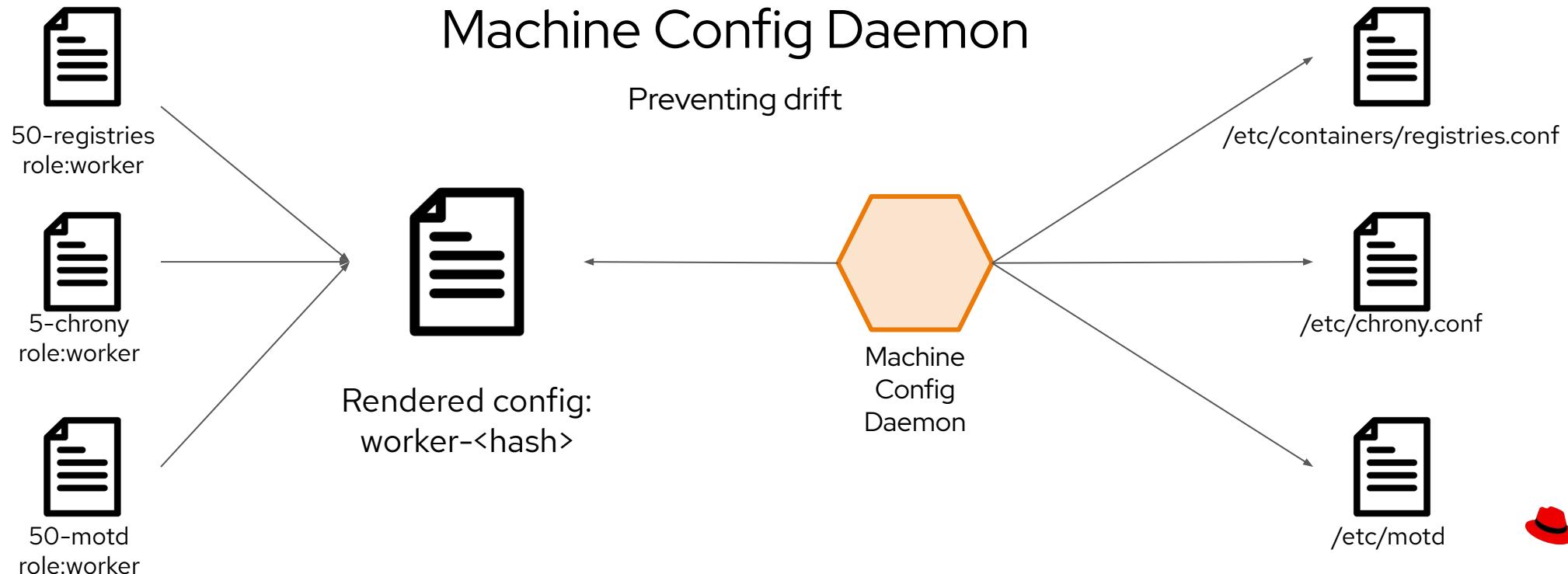
- Au fur et à mesure que le cluster s'agrandi, cette méthodologie d'utilisation d'Ignition pour configurer les hôtes au moment du provisionnement et la configuration d'Ignition rendue qui provient du serveur de configuration de la machine assure la cohérence.



# Machine Config Operator

CONFIDENTIAL Designator

- Le **Machine Config Operator** (MCO) veille à ce qu'un **Machine Config Daemon** (MCD) fonctionne sur chaque nœud. Le **MCD** extrait l'objet **MachineConfig** rendu pour le **MachineConfigPool** qui correspond au nœud, et s'assure ensuite que les fichiers définis correspondent à l'état et au contenu souhaités. Si des modifications sont apportées aux **MachineConfigs**, le MCD mettra à jour les fichiers et les configurations ciblés pour qu'ils correspondent à l'état souhaité. De cette façon, le **MCD** empêche la dérive de la configuration.



# Machine Config Operator

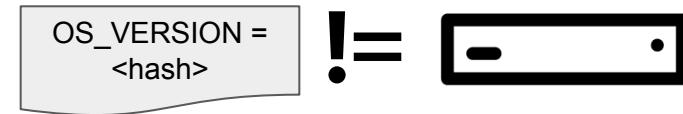
CONFIDENTIAL Designator

- Le MCO se coordonne avec le MCD pour effectuer les actions suivantes, de manière continue, lorsque des mises à jour du système d'exploitation et/ou des changements de configuration sont appliqués :
  - Nœuds consistant / inconsistants
  - Drainage des pods
  - Changements des noeuds
    - Mise à niveau du système d'exploitation
    - changements de configuration
    - systemd units
  - Redémarrage

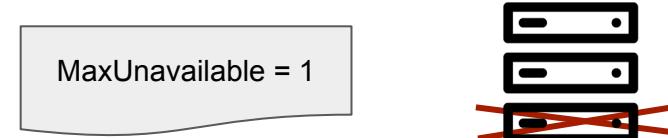
## Machine Config Daemon

Acting on drift

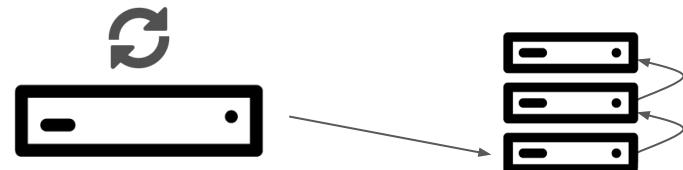
- Validates node state matches desired state



- Validate cluster state & policy to apply change



- Change is rolled across cluster

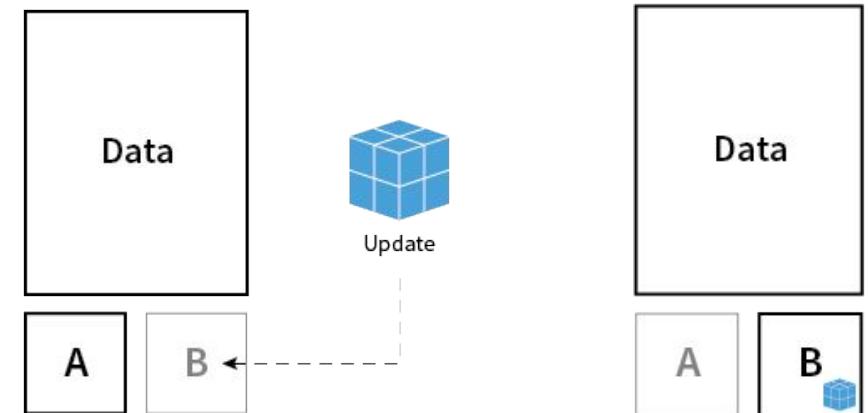


# Machine Config Operator

CONFIDENTIAL Designator

- Les mises à jour transactionnelles garantissent que RHEL CoreOS n'est jamais altéré pendant l'exécution. Il est plutôt démarré directement dans une version toujours "bonne".
  - Chaque mise à jour de l'OS est versionnée et testée comme une image complète.
  - Les binaires de l'OS (/usr) sont en lecture seule
  - Les mises à jour du système d'exploitation encapsulées dans les images des Containers
  - système de fichiers et couches de paquets disponibles pour les correctifs et le débogage

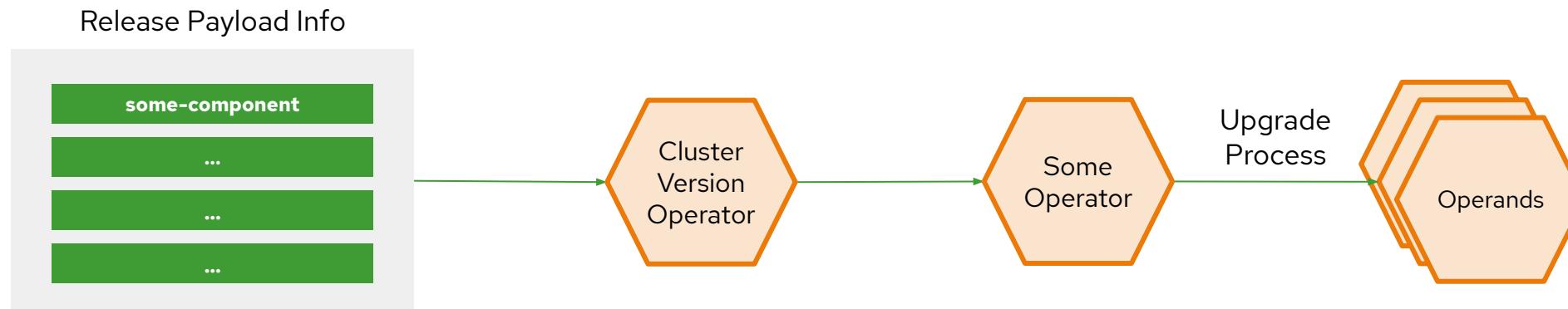
## Transactional updates with rpm-ostree



# Machine Config Operator

CONFIDENTIAL Designator

- Red Hat fournit des mises à jour de clusters via une **release payload data** (une pile de données). Cette pile de données utiles contient des références aux images des Containers pour les opérateurs du cluster.
- Le CVO (Cluster Version Operator) consomme cette pile de données utiles, puis évalue si les opérateurs ciblés correspondent ou non aux images définies dans la pile de données utile. Les opérateurs eux-mêmes savent alors quoi faire de leurs opérandes, si des modifications sont nécessaires.



# Machine Config Operator

CONFIDENTIAL Designator

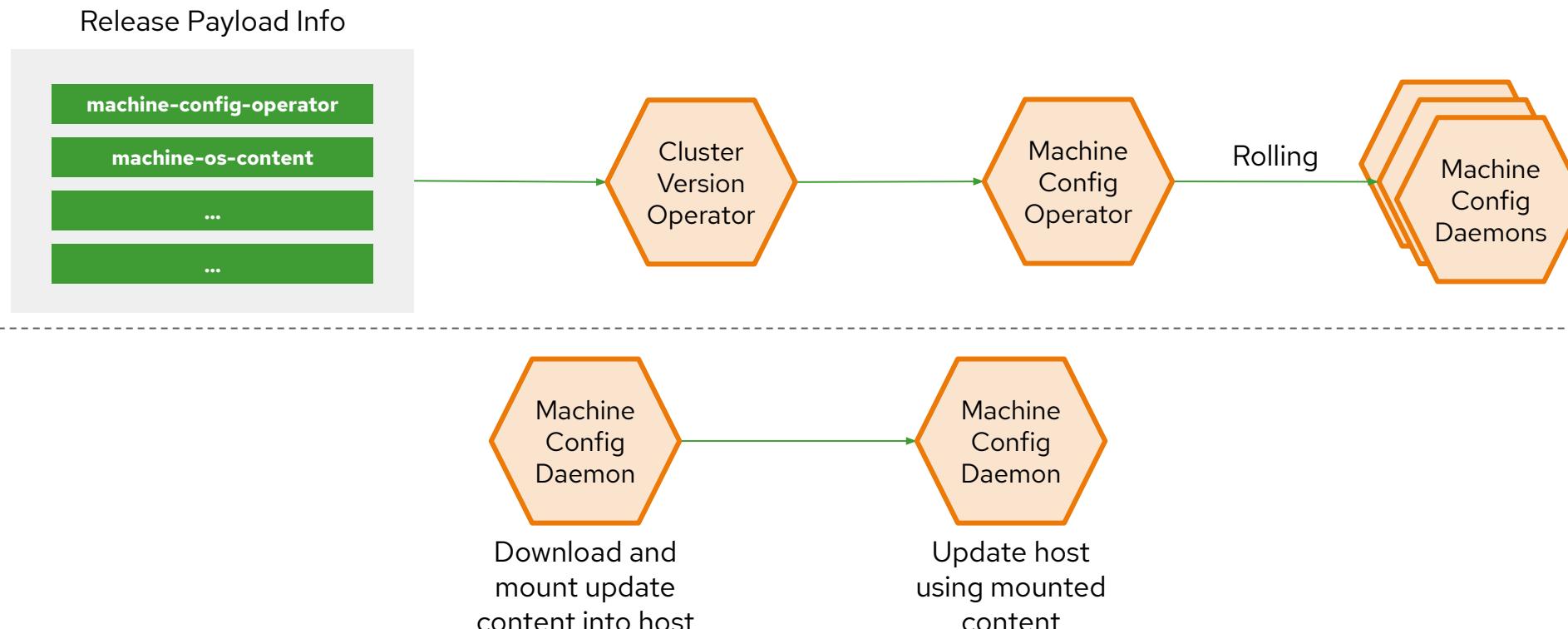
- Liste des opérateurs upgradés par CVO (Cluster Version Operator)

```
[lcolagio-redhat.com@clientvm 1 ~]$ oc get co
NAME          VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE
authentication 4.3.1    True       False        False      46h
cloud-credential 4.3.1    True       False        False      46h
cluster-autoscaler 4.3.1    True       False        False      46h
console         4.3.1    True       False        False      46h
dns             4.3.1    True       False        False      46h
image-registry 4.3.1    True       False        False      46h
ingress          4.3.1    True       False        False      3h33m
insights         4.3.1    True       False        False      46h
kube-apiserver 4.3.1    True       False        False      46h
kube-controller-manager 4.3.1    True       False        False      46h
kube-scheduler 4.3.1    True       False        False      46h
machine-api     4.3.1    True       False        False      46h
machine-config  4.3.1    True       False        False      46h
marketplace     4.3.1    True       False        False      3h32m
monitoring       4.3.1    True       False        False      3h32m
network          4.3.1    True       False        False      46h
node-tuning      4.3.1    True       False        False      3h33m
openshift-apiserver 4.3.1    True       False        False      46h
openshift-controller-manager 4.3.1    True       False        False      46h
openshift-samples 4.3.1    True       False        False      46h
operator-lifecycle-manager 4.3.1    True       False        False      46h
operator-lifecycle-manager-catalog 4.3.1    True       False        False      46h
operator-lifecycle-manager-packageserver 4.3.1    True       False        False      3h33m
service-ca       4.3.1    True       False        False      46h
service-catalog-apiserver 4.3.1    True       False        False      46h
service-catalog-controller-manager 4.3.1    True       False        False      46h
storage          4.3.1    True       False        False      46h
[lcolagio-redhat.com@clientvm 0 ~]$ █
```

# Machine Config Operator

CONFIDENTIAL Designator

- Dans le cas d'une mise à jour de nœud, le payload contient une image spécifiée pour l'opérateur Machine Config Operator (MCO).
- Le CVO s'assure que cette nouvelle version du MCO fonctionne dans le cluster. La nouvelle version du MCO garantit ensuite que la dernière version du Machine Config Daemon (MCD) fonctionne sur tous les nœuds.
- Le MCD examine le contenu de la machine pour déterminer si le système de fichiers du nœud a le bon contenu. Si ce n'est pas le cas, le processus de mise à jour a lieu.
- Le MCD télécharge le machine-os-content, qui n'est qu'un contenu rpm-ostree. Le MCD monte le contenu qui se trouve dans l'image de la machine à contenu, puis effectue une mise à jour rpm-ostree en utilisant le contenu monté.



# Use Case Machine Config

The OpenShift operating  
system

# Add SSH Key

CONFIDENTIAL Designator

- Dans cette exemple, on va ajouter une nouvelle clé SSH au worker et au master MachineConfigs. Un modèle similaire pourrait être utilisé pour faire tourner les clés SSH en utilisant les MachineConfigs.

```
#Générer une nouvelle clef ssh
ssh-keygen -t rsa -f $HOME/.ssh/node.id_rsa -N ''
cat $HOME/.ssh/node.id_rsa.pub

# Ajouter la clef ssh au machine config
oc patch machineconfig 99-worker-ssh --type=json --patch="[{\"op\":\"add\",
\"path\":\"/spec/config/passwd/users/0/sshAuthorizedKeys/-\", \"value\":\"\$(cat
\$HOME/.ssh/node.id_rsa.pub)\"}]"

# Visualiser le reboot des nodes worker
oc get node
```

# Add SSH Key

CONFIDENTIAL Designator

- Dans cet exemple, on va créer un Pod exécuté dans le cluster avec un accès SSH aux nœuds. Dans l'environnement OpenStack utilisé, l'hôte bastion est capable d'établir des connexions SSH directes avec les nœuds. Chez de nombreux fournisseurs de cloud et réseaux plus isolés, la topologie ou les pare-feu peuvent empêcher cet accès direct. Si vos nœuds se trouvent sur un réseau routé auquel vous avez accès ou si vous disposez d'un hôte de saut comme c'est le cas pour votre bastion, cette partie du Lab sera inutile.

```
# Creation d'un projet node-ssh
oc new-project node-ssh
# on utilise une image ubi8
oc new-build registry.access.redhat.com/ubi8/ubi:latest --name=node-ssh --dockerfile
- <<EOF
FROM unused
RUN dnf install -y openssh-clients
CMD ["sleep", "infinity"]
EOF
```

# Add SSH Key

CONFIDENTIAL Designator

- on ajoute la clef privé SSH générée dans un secret

```
# Creation d'un projet node-ssh
oc new-project node-ssh
# Ajouter la clef privé SSH généré dans un secret
oc create secret generic node-ssh --from-file=id_rsa=$HOME/.ssh/node.id_rsa -n node-ssh

# on utilise une image ubi8 pour lancer un ssh sur un node OCP via une la privé
chargée via le secret généré précédemment.
oc new-build registry.access.redhat.com/ubi8/ubi:latest --name=node-ssh --dockerfile
- <<EOF
FROM unused
RUN dnf install -y openssh-clients
CMD ["sleep", "infinity"]
EOF
```

# Add SSH Key

- On déploie le template basé sur l'image UBI

```
# On deploy le pod UBI8
oc apply -f $HOME/node-ssh.deployment.yaml

# On récupère le nom du pod UBI8
NODE_SSH_POD=$(oc get pod -l app=node-ssh -o
jsonpath='{.items[0].metadata.name}')

# On se connecte en ssh au Node via son IP via le Pod UBI8
oc get node -l node-role.kubernetes.io/worker -o wide

oc exec -it $NODE_SSH_POD -- ssh core@<your-node-IP>
```

```
# node-ssh.deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: node-ssh
    name: node-ssh
    namespace: node-ssh
spec:
  replicas: 1
  selector:
    matchLabels:
      app: node-ssh
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: node-ssh
    spec:
      containers:
        - image: image-registry.openshift-image-registry.svc:5000/node-ssh/node-ssh:latest
          name: node-ssh
          resources: {}
          volumeMounts:
            - name: node-ssh
              mountPath: /.ssh
        volumes:
          - name: node-ssh
            secret:
              secretName: node-ssh
              defaultMode: 0600
status: {}
```

# Add SSH Key

CONFIDENTIAL Designator

- Dans cette section, on va ajouter la clef SSH qui a servi à l'installation OCP

```
# Copier la clef publique SSH qui a servi à l'installation OCP
sudo cp /home/ec2-user/.ssh/id_rsa $HOME/.ssh/${GUID}key.pem
sudo chmod 755 $HOME/.ssh/${GUID}key.pem
```

```
oc create secret generic node-ssh --from-file=id_rsa=$HOME/.ssh/${GUID}key.pem -n node-ssh
```



# Projets et namespace Kubernetes

# Projects & namespace

The OpenShift operating  
system

# Projets et namespace Kubernetes

CONFIDENTIAL Designator

## Multitenancy

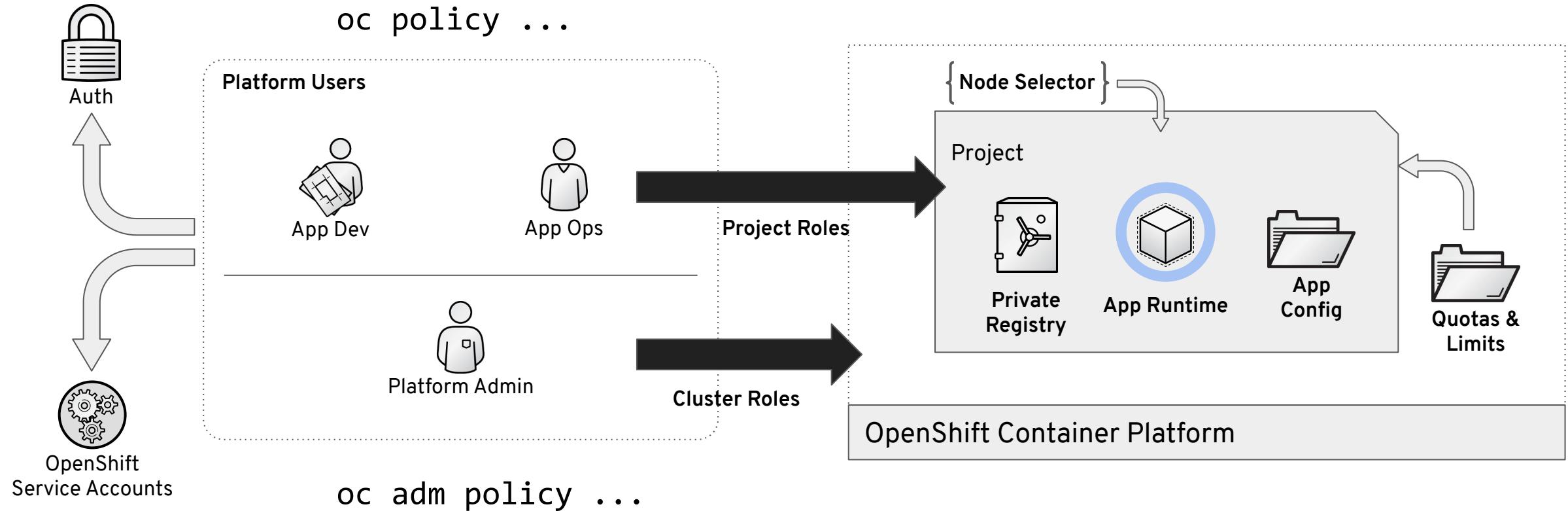
- OpenShift permet le **multi-tenancy** dans un cluster grâce à l'utilisation de l'isolation des containers au niveau de l'hôte, de l'application et du service via les projets OpenShift, en combinaison avec le contrôle d'accès basé sur les rôles (RBAC) et les politiques de réseau.

## Projets et namespace Kubernetes

- Les projets OpenShift sont des namespaces Kubernetes avec des annotations supplémentaires telles que le label MCS (Multi- Category Security) fourni dans SELinux.
- Les projets OpenShift permettent à un groupe d'utilisateurs d'organiser et de gérer leurs ressources de clusters (objets, politiques, contraintes et comptes de service) indépendamment des autres groupes ou ressources de clusters.
- Chaque projet couvre son propre ensemble d'objets, de politiques, de contraintes et de comptes de service.  
Les projets OpenShift sont le véhicule central par lequel l'accès aux ressources pour les utilisateurs réguliers est géré. Contrairement aux Kubernetes en amont, les projets OpenShift (namespace de Kubernetes) sont configurés et déployés par défaut pour permettre le **Multitenancy**.

# Projets et namespace Kubernetes

CONFIDENTIAL Designator

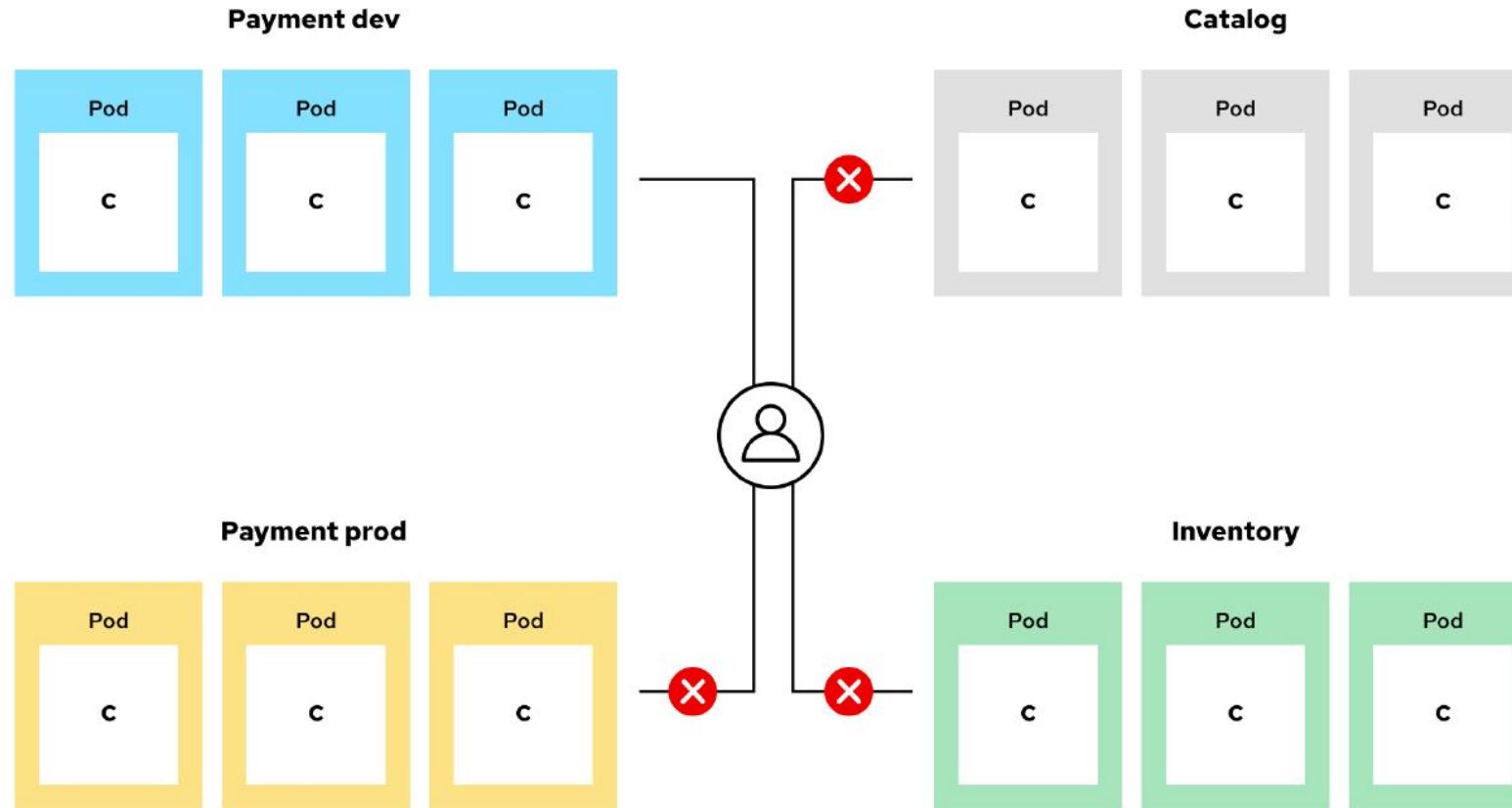


# Projets et namespace Kubernetes

CONFIDENTIAL Designator

## OpenShift Project/Namespace Isolation

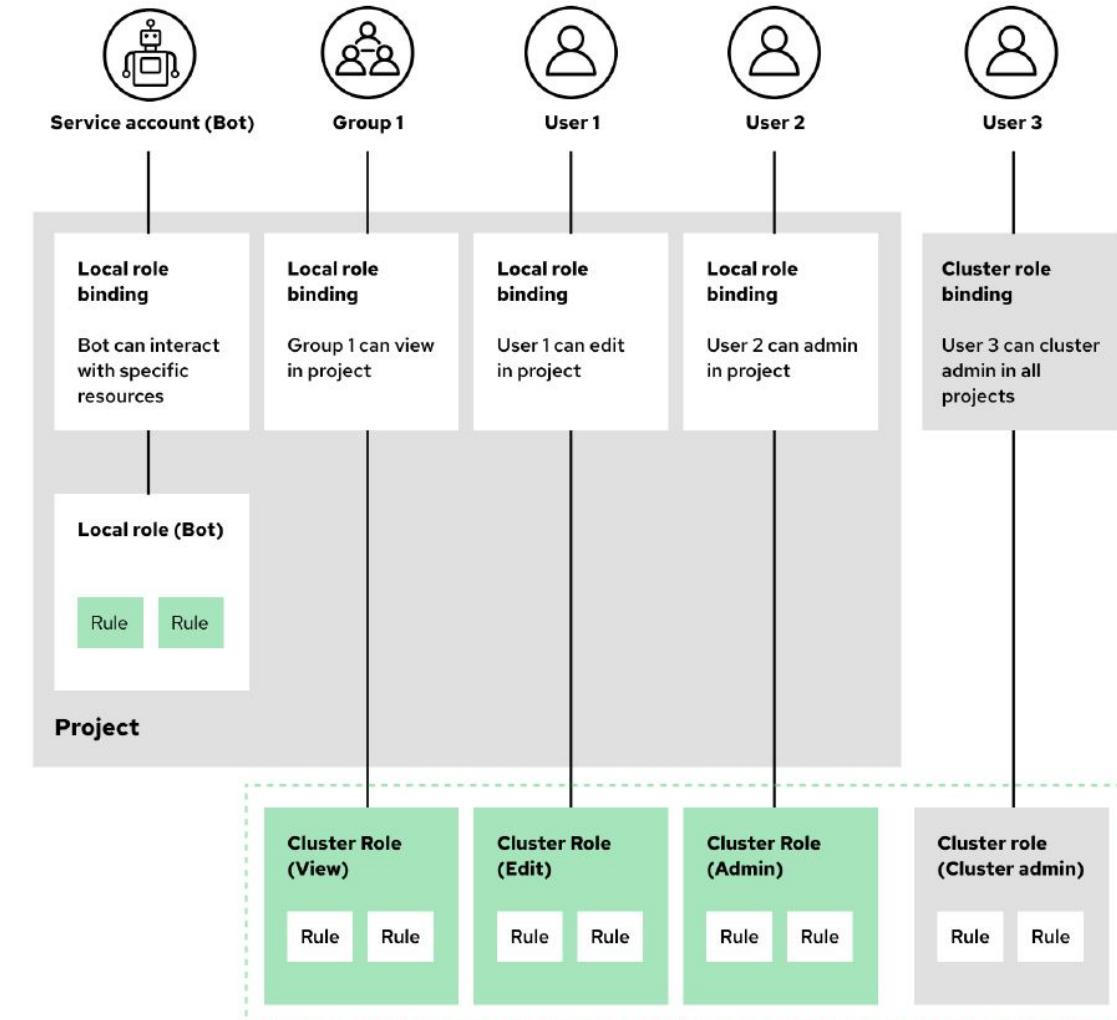
- La communication entre les pods d'un projet ou entre les pods de projets distincts est gérée par des politiques de réseau. Les politiques de réseau sont examinées en détail dans le chapitre sur la sécurité du réseau.



# Projets et namespace Kubernetes

CONFIDENTIAL Designator

- Les utilisateurs ou les groupes peuvent se voir accorder l'accès aux projets par les administrateurs de clusters ou les administrateurs de projets, mais ils peuvent également se voir déléguer la création de leurs propres projets.
- Les utilisateurs ou groupes ne sont autorisés à voir que le contenu des projets auxquels ils sont affectés et se voient attribuer des rôles spécifiques au sein des projets.
- Ces rôles sont appelés "**local role binding**" et déterminent les actions autorisées au sein de ces projets.



# Projets et namespace Kubernetes

CONFIDENTIAL Designator

## Quotas de projets

- Un quota de ressources, défini par un objet **ResourceQuota**, fournit des contraintes qui limitent la consommation globale de ressources par projet.
  - Il peut limiter la quantité d'objets qui peuvent être créés dans un projet par type, ainsi que la quantité totale de ressources de calcul et de stockage qui peut être consommée par les ressources de ce projet.
  - Les administrateurs de clusters peuvent fixer et gérer des quotas de ressources par projet, et les développeurs et les administrateurs de clusters peuvent les consulter.
- Les administrateurs de clusters peuvent également gérer des quotas de ressources sur plusieurs projets avec un quota multi-projets.
  - Les ressources utilisées dans chaque projet sélectionné sont agrégées et cet agrégat est utilisé pour limiter les ressources dans tous les projets sélectionnés.

# Projets et namespace Kubernetes

CONFIDENTIAL Designator

## Default Project Template

- Dans OpenShift Container Platform, les projets sont utilisés pour regrouper et isoler des objets connexes. Lorsqu'une demande est faite pour créer un nouveau projet en utilisant la console web ou la commande `oc new-project`, un **template** dans OpenShift Container Platform est utilisé pour fournir le projet selon un modèle, qui peut être personnalisé.
- En tant qu'administrateur de cluster, On peut autoriser et configurer la manière dont les développeurs et les comptes de service peuvent créer ou fournir eux-mêmes leurs propres projets.

```
oc get template -n openshift-config -o yaml
```

# Projets et namespace Kubernetes

CONFIDENTIAL Designator

- On déploie le template basé sur l'image UBI

```
# Creation projet en tant que admin cluster
oc login -u opentlc-mgr -p r3dh4t1!
oc get project
oc status
```

```
# Creation projet en tant que admin du projet
oc login -u andrew -p r3dh4t1!
oc get project
oc status
```

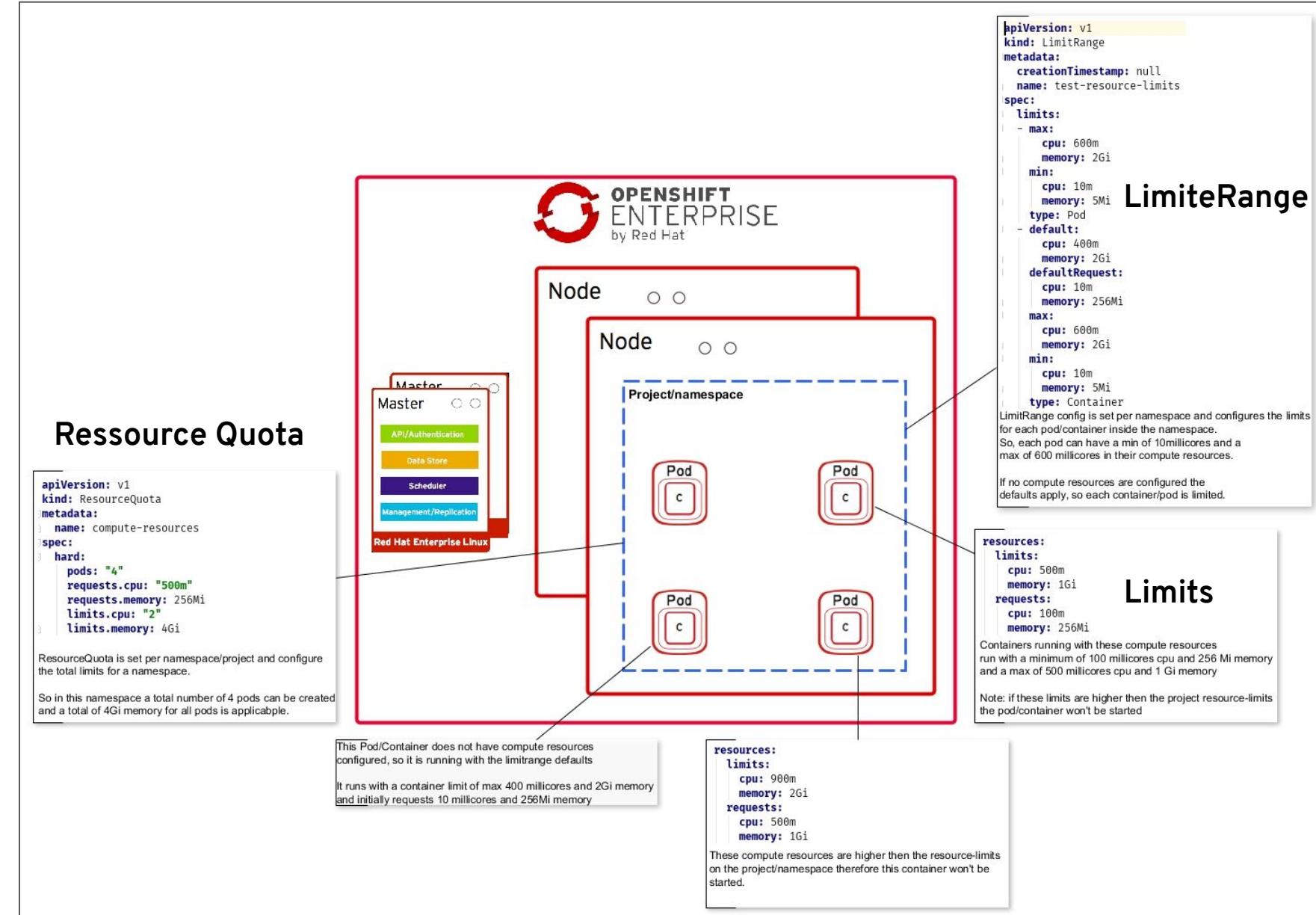
```
# Visualiser les quotas et limites
oc get limitranges -A
oc get quota -A
oc get networkpolicy -A
```

# Cloisonnement par ressource compute (Quota)

The OpenShift operating  
system

# Limites et Quota

- Il y a 3 types de limites et de restrictions disponibles dans Openshift.
  - Quotas
  - Limite Ranges
  - Compute ressources



# Les Basiques

CONFIDENTIAL Designator

## Containers

- Les unités de base des applications OpenShift sont appelées **Containers**. Les technologies de **Containers** Linux sont des mécanismes légers permettant d'isoler les processus en cours d'exécution de sorte qu'ils soient limités à l'interaction avec les seules ressources désignées

## Pods

- OpenShift s'appuie sur le concept de **Kubernetes** d'un **pod**, qui est un ou plusieurs **Containers** déployés ensemble sur un node, et la plus petite unité de calcul qui peut être définie, déployée et gérée.

## Namespaces

- Un **namespace Kubernetes** fournit un mécanisme permettant de délimiter les ressources dans un cluster. Dans OpenShift, un **projet** est un namespace Kubernetes avec des **annotations supplémentaires**.

Les namespaces offrent un cadre d'isolation pour :

- Nommer les ressources pour éviter les collisions de noms.
- Déléguer de l'autorité de gestion à des utilisateurs de confiance.
- Limiter la capacité / la consommation des ressources au sein du cluster.

La plupart des objets dans le système sont classés par namespace, mais certains sont exemptés et n'ont pas de namespace, par exemple les noeuds et les utilisateurs.

# OCP Cluster Limits

CONFIDENTIAL Designator

Maximum Pods par Cluster / Pods attendus par Node = Total nombre de Nodes

Maximum type	3.x tested maximum	4.x tested maximum
Number of Nodes	2,000	2,000
Number of Pods [1]	150,000	150,000
Number of Pods per node	250	500 [2]
Number of Pods per core	There is no default value.	There is no default value.
Number of Namespaces [3]	10,000	10,000
Number of Builds	10,000 (Default pod RAM 512 Mi) - Pipeline Strategy	10,000 (Default pod RAM 512 Mi) - Source-to-Image (S2I) build strategy
Number of Pods per namespace [4]	25,000	25,000
Number of Services [5]	10,000	10,000
Number of Services per Namespace	5,000	5,000
Number of Back-ends per Service	5,000	5,000
Number of Deployments per Namespace [4]	2,000	2,000

Dans OpenShift Container Platform 4.4, la moitié d'un cœur de CPU (500 millicore) est réservée par le système par rapport à OpenShift Container Platform 3.11 et aux versions précédentes.



# Quota

CONFIDENTIAL Designator

- Les quotas sont des limites configurées par **namespace** et servent de limite supérieure pour les ressources dans ces namespaces particuliers.
- Ce quota indique que le namespace peut avoir un maximum de 5 pods, et/ou un maximum de 2 cœurs et 2 Go de mémoire, la " demande " initiale de ce namespace est de 500 millicores et 512 Mo de mémoire.
- Il existe également d'autres scope comme
  - best effort
  - guaranteed

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: namespace-quota
spec:
  hard:
    pods: "5"
    requests.cpu: "500m"
    requests.memory: 512Mi
    limits.cpu: "2"
    limits.memory: 2Gi
  scopes:
  - NotTerminating
```



# Cluster Ressource Quota

CONFIDENTIAL Designator

- Un quota multi-projets, défini par un objet ClusterResourceQuota, permet de partager les quotas entre plusieurs projets. Les ressources utilisées dans chaque projet sélectionné sont agrégées et cet agrégat est utilisé pour limiter les ressources de tous les projets sélectionnés.
- Lors de la création de quotas, vous pouvez sélectionner plusieurs projets en fonction via un **selector** basé sur les annotations, de label ou les deux.

```
1.  apiVersion: v1
2.  kind: ClusterResourceQuota
3.  metadata:
4.    name: for-user
5.  spec:
6.    quota:
7.      hard:
8.        pods: "10"
9.        secrets: "20"
10.   selector:
11.     annotations:
12.       openshift.io/requester: andrew
13.     labels: null
14.   status:
15.     namespaces:
16.       - namespace: ns-one
17.         status:
18.           hard:
19.             pods: "10"
20.             secrets: "20"
21.           used:
22.             pods: "1"
23.             secrets: "9"
24.         total:
25.           hard:
26.             pods: "10"
27.             secrets: "20"
28.           used:
29.             pods: "1"
30.             secrets: "9"
```

Red Hat

Cluster Resource Quotas > Cluster Resource Quota Details

**CRQ clusterquota-andrew**

Actions ▾

[Overview](#) [YAML](#)

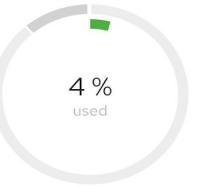
### Cluster Resource Quota Overview

CPU Request



25 %  
used

CPU Limit



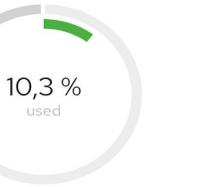
4 %  
used

Memory Request



50 %  
used

Memory Limit



10,3 %  
used

User Management
Users
Groups
Service Accounts
Roles
Role Bindings
Administration
Cluster Settings
Namespaces
Resource Quotas
Limit Ranges
Custom Resource Definitions

### Cluster Resource Quota Details ⓘ

Resource Type	Capacity	Used	Max
configmaps	⊖	3	25
limits.cpu	⊖	1	25
limits.memory	⊖	4306277Ki	40Gi
persistentvolumeclaims	⊖	4	25
pods	⊖	14	25
requests.cpu	⊖	1250m	5
requests.memory	⊖	3223741824	6Gi
services	⊖	13	25

121

REDACTED

# Limit ranges

CONFIDENTIAL Designator

- Un autre type de limite est la "limit range". Une plage de limites est également configurée sur un **namespace**, mais une **limit range** définit des limites par **pod** et/ou **Container** dans ce **namespace**. Elle fournit essentiellement des limites de **CPU** et de **mémoire** pour les **containers** et les **pods**.
- La **limit range** définit la portée de ces limites.
- Dans l'exemple chaque **pod** du **namespace** aura **initiallement** une capacité de 200 millicores et 6 Mo de mémoire et pourra fonctionner avec un **maximum** de 1 Go de mémoire et 2 cœurs de processeur.
- Les limites réelles du **pod** ou du **container** peuvent être définies dans les spécifications du pod ou du Container
- Il convient également de noter les limites par **default** et par **defaultRequest** dans la plage de limites du Container. Il s'agit des limites appliquées à un Container qui ne spécifie pas d'autres limites et se voit donc attribuer la valeur par défaut.

```
apiVersion: v1
kind: LimitRange
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "200m"
        memory: "6Mi"
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
      defaultRequest:
        cpu: "200m"
        memory: "100Mi"
```

Horizontal Pod Autoscalers

Networking &gt;

Storage &gt;

Builds &gt;

Monitoring &gt;

Compute &gt;

User Management &gt;▼

Users

Groups

Service Accounts

Roles

Role Bindings

Administration &gt;▼

Cluster Settings

Namespaces

Resource Quotas

Limit Ranges

Custom Resource Definitions

Project: usecase3-app ▾

**Namespace** usecase3-app**Labels**

No labels

**Annotations**0 Annotations **Created At** Jun 9, 5:29 pm**Owner**

No owner

**Limits**

Type	Resource	Min	Max	Default Request	Default Limit	Max Limit/Request Ratio
Container	cpu	-	2	50m	500m	-
Container	memory	-	6Gi	256Mi	1536Mi	-
Pod	cpu	-	2	-	-	-
Pod	memory	-	12Gi	-	-	-

# Compute resources

CONFIDENTIAL Designator

- La dernière des limites est probablement la plus facile à comprendre, les ressources de compute sont définies sur les spécifications du **Pod** ou du **Container**, dans un **deploymentconfig**.
- Dans cet exemple, on définit les limites de cpu et de mémoire pour un **pod** en particulier.
- Le **Pod** demandera initialement 100 millicores et 200 Mo de mémoire et atteindra un maximum de 200 millicores et 400 Mo de mémoire.
- Notez que si une **limitRange** est également fournie dans le **namespace** où le Pod s'exécutera et que les limites des ressources de compute sont comprises dans la plage de limites, le Pod fonctionnera correctement.
- Si toutefois les limites sont supérieures aux limites de la plage de limites, le pod ne démarrera pas

```
apiVersion: v1
kind: Pod
spec:
  containers:
    - image: nginx
      name: nginx
  resources:
    requests:
      cpu: 100m
      memory: 200Mi
    limits:
      cpu: 200m
      memory: 400Mi
```

# SCOPE / QoS

CONFIDENTIAL Designator

- Toutes les **limites** ont un **request** (demande) et un **maximum** qui définissent d'autres plages sur lesquelles le **Pod** peut fonctionner. Lorsque la demande est de toute façon intense et les objectifs "garantis" (tant que le nœud sous-jacent a la capacité). Cela donne la possibilité de définir implicitement différents niveaux de **QoS**.
  - **BestEffort** - aucune limite n'est prévue définit. Le Pod réclame tout ce dont il a besoin, mais il est le premier à être réduit ou tué lorsque d'autres Pods demandent des ressources.
  - **Burstable** - Les limites de **request** (demande) sont inférieures aux limites **maximales**. La limite initiale est garantie, mais le **Pod** peut, si des ressources sont disponibles, exploser jusqu'à son maximum.
  - **Garantie** - la **request** et le **maximum** sont identiques, donc le pod réclame directement le maximum de ressources, même si le pod n'utilise pas toutes les ressources au départ ; elles sont déjà réclamées par le cluster, et donc garanties.

Vous trouverez ci-dessous une vue d'ensemble des trois différentes limites d'Openshift.

# Default Project Template - exemple

CONFIDENTIAL Designator

```
apiVersion: v1
items:
- apiVersion: template.openshift.io/v1
  kind: Template
  metadata:
    name: project-request
    namespace: openshift-config
  objects:
- apiVersion: v1
  kind: LimitRange
  metadata:
    name: ${PROJECT_NAME}-core-resource-limits
    namespace: ${PROJECT_NAME}
  spec:
    limits:
      - default:
          cpu: 500m
          memory: 1.5Gi
        defaultRequest:
          cpu: 50m
          memory: 256Mi
        max:
          cpu: 2
          memory: 6Gi
        type: Container
      - max:
          cpu: 2
          memory: 12Gi
        type: Pod
```

```
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-all-namespaces
  spec:
    ingress:
      - from:
          - namespaceSelector: {}
            podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-ingress-namespace
  spec:
    ingress:
      - from:
          - namespaceSelector:
              matchLabels:
                network-policy: global
            podSelector: null
```

```
- apiVersion: project.openshift.io/v1
  kind: Project
  metadata:
    annotations:
      openshift.io/description: ${PROJECT DESCRIPTION}
      openshift.io/display-name: ${PROJECT DISPLAYNAME}
      openshift.io/requester: ${PROJECT REQUESTING USER}
    kind: ClusterRole
    name: system:deployer
  subjects:
    - kind: ServiceAccount
      name: deployer
      namespace: ${PROJECT NAME}
- apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: admin
    namespace: ${PROJECT_NAME}
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: ClusterRole
    name: admin
  subjects:
    - apiGroup: rbac.authorization.k8s.io
      kind: User
      name: ${PROJECT_ADMIN_USER}
  parameters:
    - name: PROJECT NAME
    - name: PROJECT DISPLAYNAME
    - name: PROJECT DESCRIPTION
    - name: PROJECT ADMIN USER
    - name: PROJECT_REQUESTING_USER
  kind: List
  metadata:
    resourceVersion: ""
    selfLink: ""
```

# SCOPE / QoS

CONFIDENTIAL Designator

All limits have a request and a max which define further ranges the Pod can operate on. Where the request is by all intense and purposes “guaranteed” (as long as the underlying node has the capacity). This gives the option to implicitly set different QoS tiers.

**BestEffort** – no limits are provided whatsoever. The Pod claims whatever it needs, but is the first one to get scaled down or killed when other Pods request for resources.

**Burstable** – The request limits are lower than the max limits. The initial limit is guaranteed, but the Pod can, if resources are available, burst to its maximum.

**Guaranteed** – the request and max are identical, so it directly claims the max resources, even though the pod might not initially use all resources they are already claimed by the cluster, and therefore guaranteed.

Below is an overall view of the three different Openshift limits.

# Use Case Quota

The OpenShift operating system

# Cluster Quota

CONFIDENTIAL Designator

- Créer un cluster quota qui fixe les limites strictes suivantes pour les projets créé par Andrew

```
# Creation cluster quota

oc login -u system:admin
export OCP_USERNAME=andrew

oc create clusterquota clusterquota-${OCP_USERNAME} \
--project-annotation-selector=openshift.io/requester=${OCP_USERNAME} \
--hard pods=25 \
--hard requests.memory=6Gi \
--hard requests.cpu=5 \
--hard limits.cpu=25 \
--hard limits.memory=40Gi \
--hard configmaps=25 \
--hard persistentvolumeclaims=25 \
--hard services=25

oc get clusterresourcequota
NAME          AGE
clusterquota-andrew  4d23h
```

# Cluster Quota

CONFIDENTIAL Designator

- Décrivez le cluster resource quota. Notez que vous n'aurez pas encore de valeurs affichées, car le demandeur

```
# Description cluster quota

oc describe clusterresourcequota clusterquota-andrew
Name: clusterquota-andrew
Created: 18 seconds ago
Labels: <none>
Annotations: <none>
Namespace Selector: []
Label Selector:
AnnotationSelector: map[openshift.io/requester:andrew]
Resource     Used     Hard
-----      -----      -----
```

# Cluster Quota

CONFIDENTIAL Designator

- Décrivez le cluster resource quota. Notez que vous n'aurez pas encore de valeurs affichées, car le demandeur n'a pas encore de projets.

```
# Creation clusterquota

oc login -u andrew -p r3dh4t1!
oc new-project projet-andrew-1

apiVersion: v1
kind: Pod
metadata:
  name: example
  labels:
    app: hello-openshift
  namespace: projet-andrew-1
spec:
  containers:
    - name: hello-openshift
      image: openshift/hello-openshift
```

```
oc login -u opentlc-mgr -p r3dh4t1!
Login successful.

You have access to 78 projects, the list has been suppressed. You
can list all projects with 'oc projects'

Using project "projet-andrew-1".

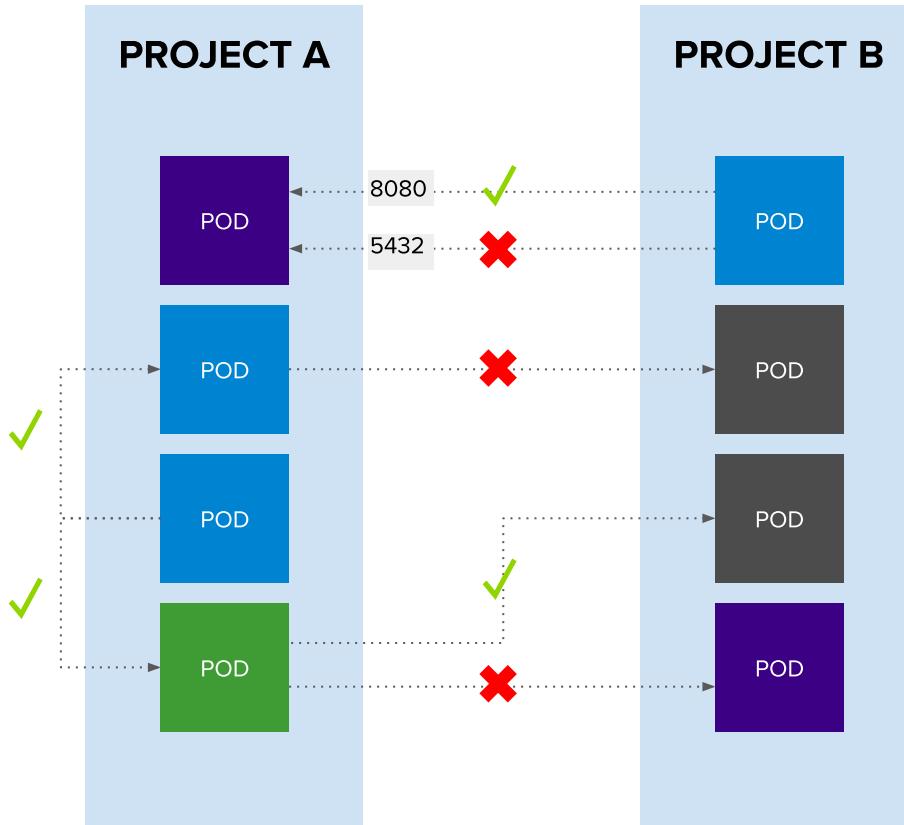
oc describe clusterresourcequota clusterquota-andrew -A
Name:           clusterquota-andrew
Created:        5 days ago
Labels:          <none>
Annotations:    <none>
Namespace Selector:  ["projet-andrew-1"]
Label Selector:
AnnotationSelector: map[openshift.io/requester:andrew]
Resource          Used Hard
-----  -----
configmaps        0   25
limits.cpu        500m 25
limits.memory     1536Mi 40Gi
persistentvolumeclaims 0   25
pods              1   15
requests.cpu      50m   5
requests.memory   256Mi 10Gi
services          0   25
```

# Network Policy

The OpenShift operating system

# Network Policy

- Le plug-in network policy permet aux administrateurs de projet de configurer des politiques d'isolation réseau à l'aide d'objets de type NetworkPolicy.
- Une network policy est une spécification qui décrit comment des groupes de pods (utilisant des label) sont autorisés à communiquer entre eux et avec d'autres endpoints réseau.



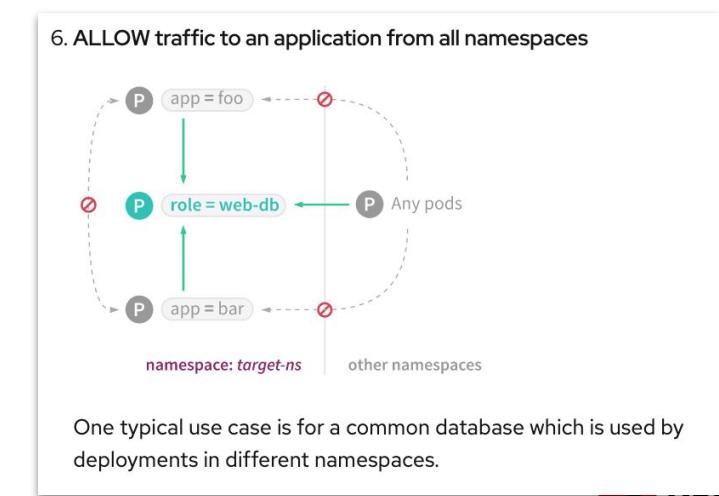
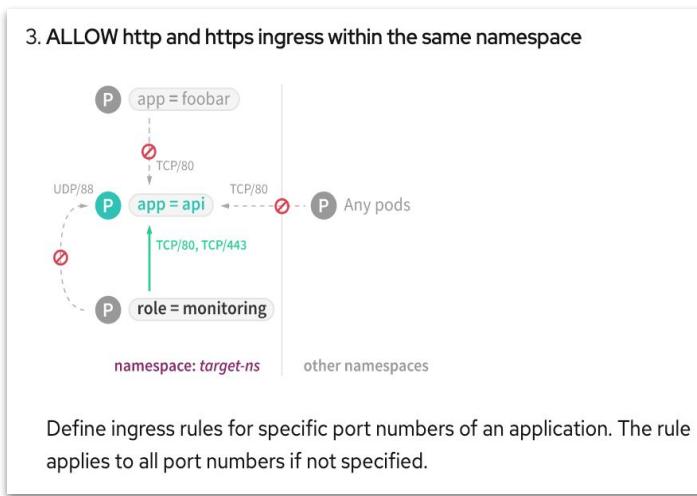
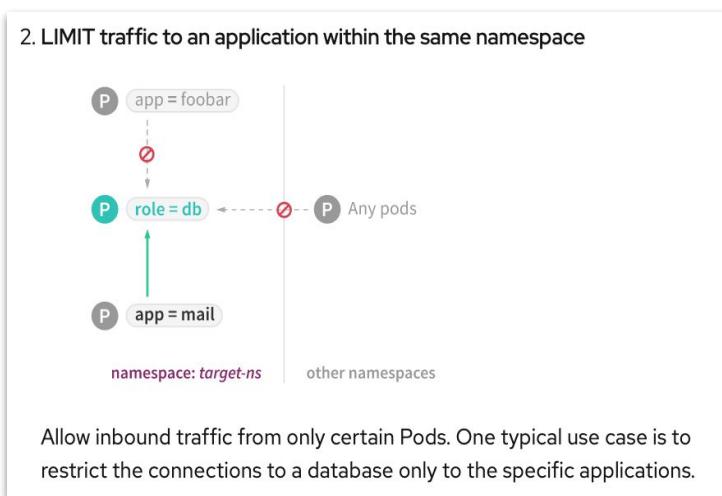
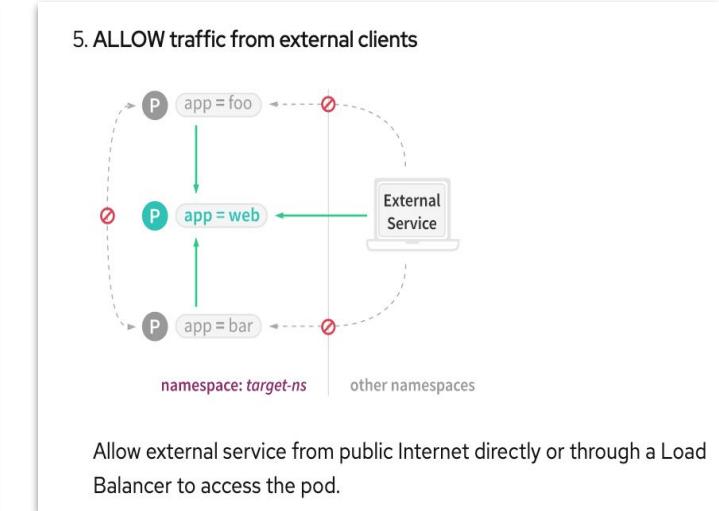
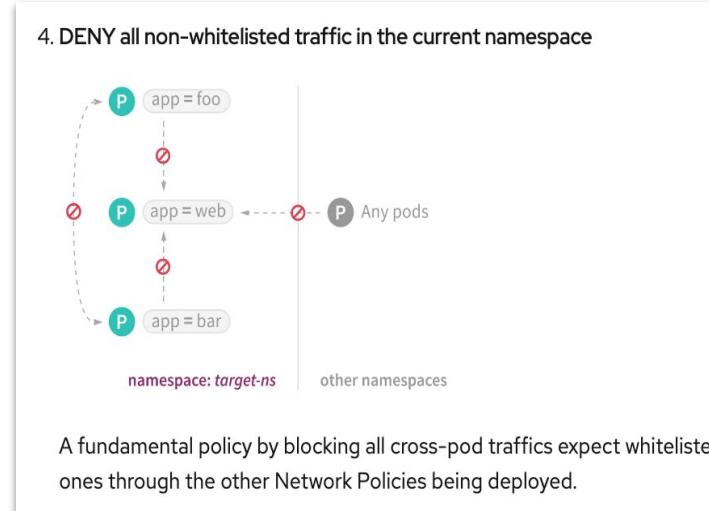
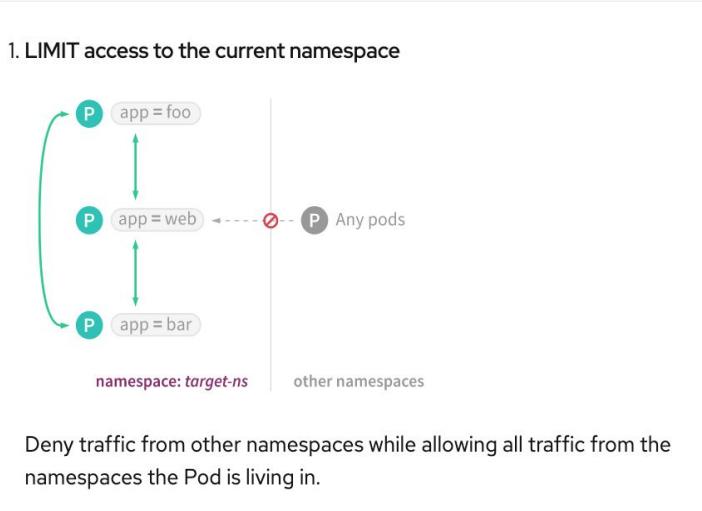
## Exemple Policies

- Permettre tout le traffic à l'intérieur du project
- Permettre le traffic de vert to gray
- Allow traffic to purple on 8080

```
apiVersion: extensions/v1beta1
kind: NetworkPolicy
metadata:
  name: allow-to-purple-on-8080
spec:
  podSelector:
    matchLabels:
      color: purple
  ingress:
  - ports:
    - protocol: tcp
      port: 8080
```

# Network Policy

- Le dashboard OCP 4 offre une assistance à la création de networkpolicy en proposant des templates



# Use Case

The OpenShift operating system

# Network Policy

CONFIDENTIAL Designator

- Créer un environnement backend avec un pod Bastion, web, app, db

```
export ZONE=zone-backend

oc create -f - <<EOF
apiVersion: v1
kind: Namespace
metadata:
  labels:
    app: test-network-policy
  name: $ZONE
EOF

oc project $ZONE

oc create -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: $ZONE-bastion
  labels:
    app: $ZONE-bastion
spec:
  containers:
    - name: $ZONE-bastion
      image: pragma/network-multitool
      args:
        - sh
      tty: true
EOF
```

```
for POD in web app db
do

  oc create -f - <<EOF
  apiVersion: v1
  kind: Pod
  metadata:
    name: $ZONE-$POD
    labels:
      app: $ZONE-$POD
  spec:
    containers:
      - name: $ZONE-$POD
        image: pragma/network-multitool
        command: ["/bin/bash"]
        args: ["-c", "while true; do { echo -e 'HTTP/1.1 200 OK\r\n'; hostname; ip a | grep 'global eth0';} | nc -l 4444; done"]
        tty: true
        ports:
          - containerPort: 8080
EOF

done
```

Red Hat

# Network Policy

CONFIDENTIAL Designator

- Test de connexion du bastion vers app-web

```
oc get pod -owide
```

NAME	NOMINATED NODE	READY	STATUS	RESTARTS	AGE	IP	NODE	
		READINESS	GATES					
zone-backend-app		1/1	Running	0	36m	10.131.0.38	ip-10-0-151-213.us-east-2.compute.internal	<none>
zone-backend-bastion		1/1	Running	0	36m	10.131.0.36	ip-10-0-151-213.us-east-2.compute.internal	<none>
zone-backend-db		1/1	Running	0	36m	10.131.0.37	ip-10-0-151-213.us-east-2.compute.internal	<none>
zone-backend-web		1/1	Running	0	36m	10.131.0.39	ip-10-0-151-213.us-east-2.compute.internal	<none>

```
oc rsh zone-backend-bastion
... nc 10.131.0.39 4444
... HTTP/1.1 200 OK
```

```
oc rsh zone-backend-web
ou
watch logs zone-backend-web
```

# Network Policy

CONFIDENTIAL Designator

- Créer une network policy qui isole toute communication entre les pods

```
# default-deny-all

oc create -f - <<EOF

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny-all

spec:
  podSelector: {}
  policyTypes:
    - Ingress

EOF
```

# Network Policy

CONFIDENTIAL Designator

- Créer une network policy qui autorise les pods labélisés bastion à communiquer avec les pods labélisés web dans le même namespace

```
# web-allow-bastion

oc create -f - <<EOF

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-bastion

spec:
  podSelector:
    matchLabels:
      app: zone-backend-web
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: zone-backend-bastion
  policyTypes:
    - Ingress

EOF
```

# Network Policy

CONFIDENTIAL Designator

- Créer une network policy qui autorise tous les namespaces à communiquer avec les pods labélisés web

```
# web-allow-all-ns

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all-ns
spec:
  podSelector:
    matchLabels:
      app: zone-backend-web
  ingress:
    - from:
        - namespaceSelector: {}
  policyTypes:
    - Ingress
```

# Cloisonnement par nodes

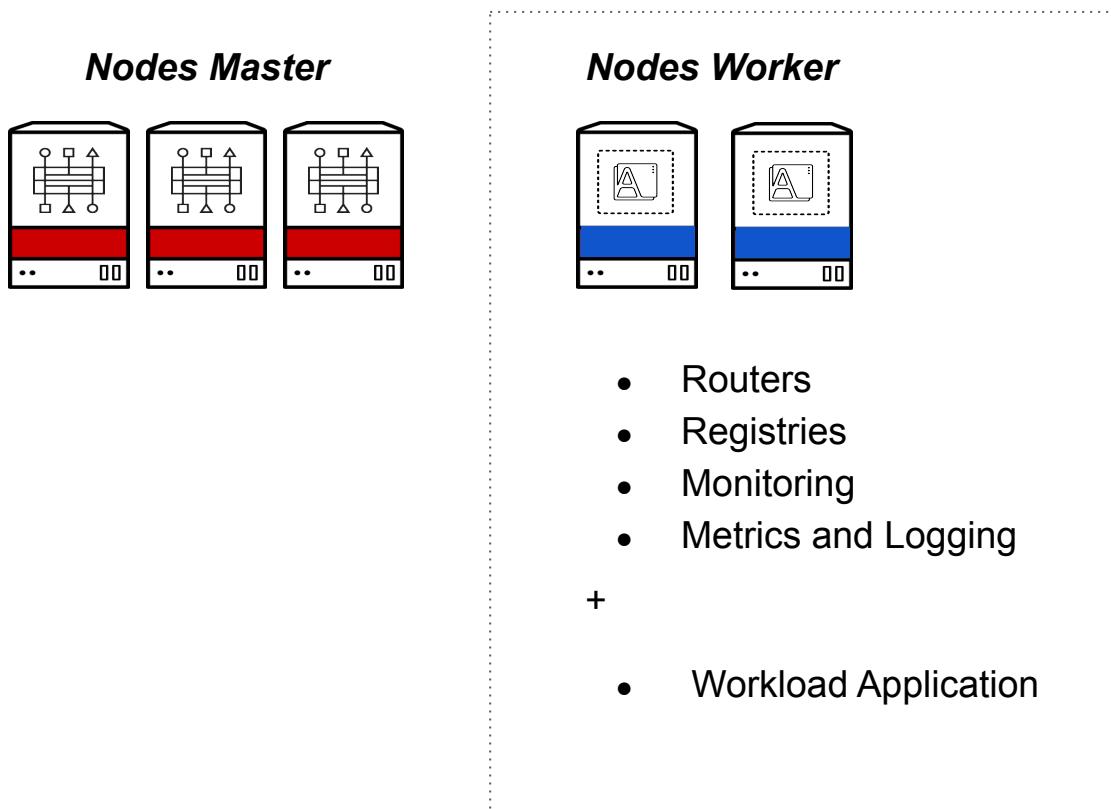
The OpenShift operating  
system

# Node sélection

CONFIDENTIAL Designator

## Step1 - Installation par default

- Au minimum, un cluster OpenShift contient 2 nodes de workers en plus de 3 nodes de control plane (Master)
- Bien qu'un grand nombre de composants critiques pour le fonctionnement du cluster soient isolés sur les masters, il y en a encore qui, par défaut, fonctionnent sur les nodes workers: Routers, Registries, Monitoring, Metrics et Logging  
Or ces nodes sont utilisés pour le workload des utilisateurs du cluster qui y déplient leurs applications.
- Par conséquent, pour garantir un environnement plus sécurisé et une disponibilité continue de ces ressources et du cluster, il peut être essentiel de configurer des nodes d'infrastructure dédiés et de les isoler des autres workload applicatifs.

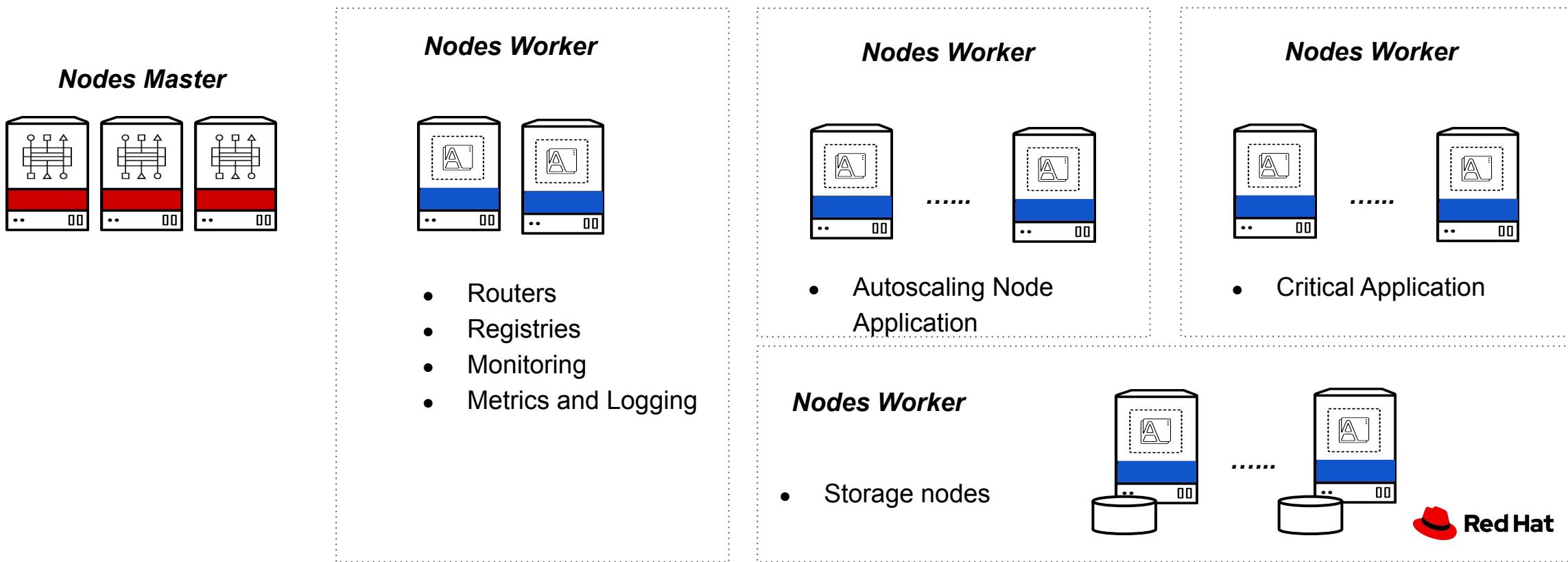


# Node sélection

CONFIDENTIAL Designator

## Step2 - Analyse du besoin

- Chaque organisation doit analyser quels ajouts dans le clusters sont considérés comme essentiels suivant ces besoins afin de pouvoir:
  - Dévier des ressources compute aux applications critiques
  - Autoscaler les applications
  - Isoler les nodes Stockage
  - Isoler l'accès au réseau

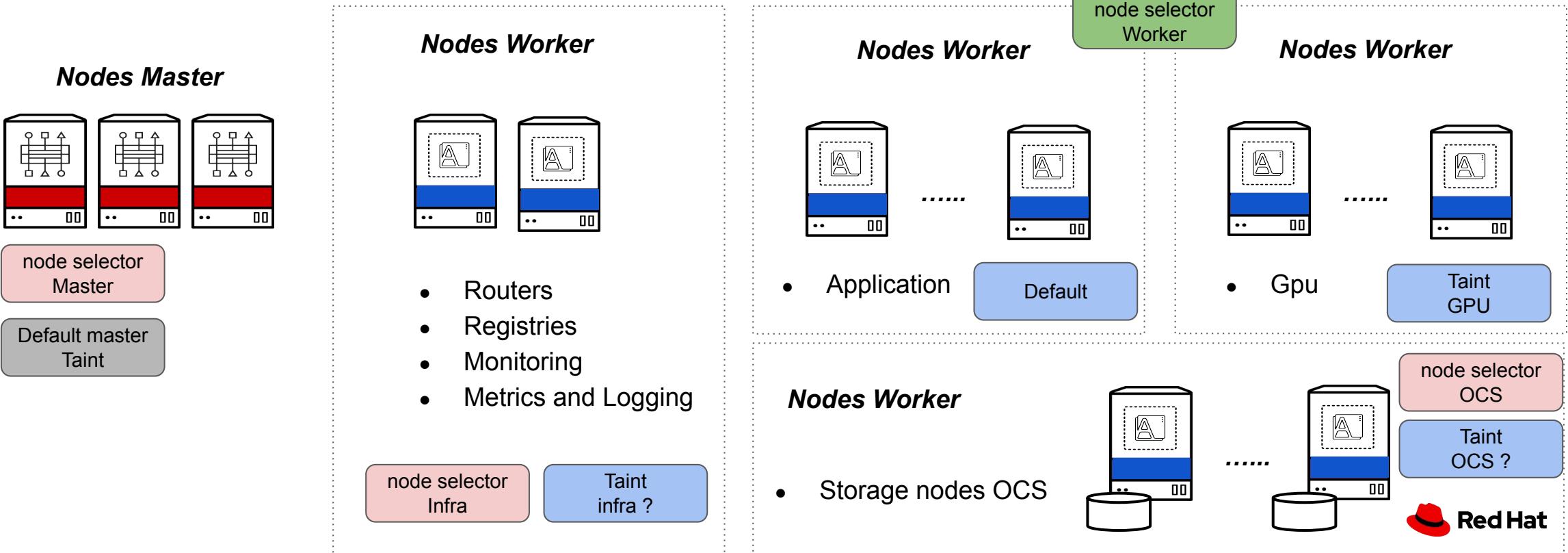


# Use Case Client

CONFIDENTIAL Designator

## Step2 - Analyse du besoin

- Chaque organisation doit analyser quels ajouts dans le clusters sont considérés comme essentiels suivant ces besoins afin de pouvoir:
  - Dévier des ressources compute aux applications critiques
  - Isoler les nodes GPU
  - Isoler les nodes Stockage
  - Isoler les nodes OCS



## Step3 - Comment configurer cette isolation

- Avant de plonger dans la configuration des **nodes** pour qu'ils soient vraiment isolés, voyons comment le comportement du **scheduler** par défaut dans OpenShift peut être influencé sans modifier directement la planification:
  - **Node Selectors**

Les node selectors sont utilisés lorsque des ressources spécifiques doivent être **scheduler** sur des nodes spécifiques. Si un node selector est spécifié en conjonction avec une ressource et qu'il n'y a pas de nodes disponibles avec ce label de selecteur, ces ressources ne pourront pas être **scheduler**. Au contraire, si un **node selector** n'est pas spécifié dans une ressource, **ces ressources peuvent toujours être planifiées sur les nodes avec node selectors.**
  - **Taints et Tolerations**

Les **Taints** et **Tolerations** sont utilisés lorsque le résultat souhaité est d'**empêcher la scheduling par défaut de ressources sur des nodes particuliers**, sauf nécessité contraire. Lorsqu'une **Taints** est appliquée à un node, toutes les ressources seront repoussées de ce node, à moins qu'il ne soit configuré avec une tolérance pour cette **Taints**, permettant de la scheduler à cet endroit.
  - **Affinity et Anti-Affinity**

L'**Affinity** et **Anti-Affinity** sont utilisées lorsque le résultat souhaité du **scheduling** des ressources est relatif à d'autres ressources. Certains cas d'utilisation consistent à empêcher le scheduling des pods sur un même node, ou à s'assurer qu'un pod est placé sur un node où un autre pod est déjà présent.  
Les règles d'affinité et d'anti-affinité sont souvent utilisées pour équilibrer les pods entre les nodes (souvent aussi entre les zones de disponibilité) afin d'assurer une haute disponibilité (HA).

# Node sélection

CONFIDENTIAL Designator

## Step4 - Principe de configuration

- Pour garantir la haute disponibilité (HA), chaque cluster doit disposer de trois nodes, idéalement répartis sur les zones de disponibilité.
- Dans un environnements sans cloud provider, lorsque qu'un node (worker) supplémentaires est ajouté au cluster, le label **Infra** est appliquée manuellement de cette façon :

```
oc label node <node-name> node-role.kubernetes.io/infra=
```

- Le label worker devra être retiré

```
oc label node <node-name> node-role.kubernetes.io/worker-
```

```
oc get node
```

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-129-241.us-east-2.compute.internal	Ready	master	12d	v1.16.2
ip-10-0-130-249.us-east-2.compute.internal	Ready	infra	108m	v1.16.2
ip-10-0-147-48.us-east-2.compute.internal	Ready	worker	130m	v1.16.2
ip-10-0-149-181.us-east-2.compute.internal	Ready	master	12d	v1.16.2
ip-10-0-151-213.us-east-2.compute.internal	Ready	worker	12d	v1.16.2
ip-10-0-153-7.us-east-2.compute.internal	Ready	infra	109m	v1.16.2
ip-10-0-161-76.us-east-2.compute.internal	Ready	worker	130m	v1.16.2
ip-10-0-163-73.us-east-2.compute.internal	Ready	master	12d	v1.16.2
ip-10-0-172-2.us-east-2.compute.internal	Ready	infra	108m	v1.16.2



# Node sélection

CONFIDENTIAL Designator

## Step4 - Principe de configuration

- La dernière étape pour faire de ces nodes des nodes d'infrastructure, consiste à créer un Machine Config Pool (MCP) d'infrastructure et à appliquer ensuite cette infra Machine Config (MC) aux nodes d'infrastructure.

```
cat infra-mcp.yaml
```

```
---
```

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: infra
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,infra]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/infra: ""
```

```
oc create -f infra-mcp.yaml
```



# Node sélection

CONFIDENTIAL Designator

## Step4 - Principe de configuration (Taint)

- La Taint suivante doit être appliquée aux nodes d'infrastructure pour garantir que seuls les composants de l'infrastructure peuvent y être schedulé : assurant l'isolement des composants de l'infrastructure.

```
oc adm taint nodes -l node-role.kubernetes.io/infra infra=reserved:NoSchedule infra=reserved:NoExecute  
node/ip-10-0-130-249.us-east-2.compute.internal tainted  
node/ip-10-0-153-7.us-east-2.compute.internal tainted  
node/ip-10-0-172-2.us-east-2.compute.internal tainted
```

# Node sélection

CONFIDENTIAL Designator

## Step4 - placer un pod

- Pour déplacer des composants vers les nodes d'infrastructure, ils doivent maintenant avoir le Node Selector d'infrastructure et une Taint assignée aux nodes d'infrastructure.

```
apiVersion: v1
kind: Pod
metadata:
  name: network-multitool
  labels:
    app: network-multitool
spec:
  containers:
    - name: network-multitool
      image: praqma/network-multitool
      command: ["/bin/bash"]
      tty: true
      ports:
        - containerPort: 8080
  nodeSelector:
    node-role.kubernetes.io/infra: ""
  tolerations:
    - effect: NoSchedule
      key: infra
      value: reserved
    - effect: NoExecute
      key: infra
      value: reserved
```



# Node sélection

CONFIDENTIAL Designator

## Default cluster-wide node selectors

- On peut utiliser des node selectors par défaut sur les pods ainsi que des labels sur les nodes pour contraindre tous les pods créés dans un cluster à aller sur des nodes spécifiques.

### Procedure

To add a default cluster node selector:

- Edit the Scheduler Operator Custom Resource to add the cluster node selectors:

```
$ oc edit scheduler cluster
```

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
spec: {}
  policy:
    spec:
      defaultNodeSelector: type=user-node,region=east ①
```

- ① Add a node selector with the appropriate <key>:<value> pairs.

For example, to label a node:

```
$ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
```

- Lorsque on crée un pod, OpenShift Container Platform ajoute la <clé>:<valeur> appropriée et schédule le pod sur le node labelisé.

For example:

```
spec:
  nodeSelector:
    region: east
    type: user-node
```

# Node sélection

CONFIDENTIAL Designator

## Project-wide node selectors

- On peut utiliser des node selectors sur un projet ainsi que des labels sur les nodes pour contraindre tous les pods créés dans un namespace à aller sur les node labélisés

### Procedure

To add a default project node selector:

- Create a namespace or edit an existing namespace associated with the project to add the `openshift.io/node-selector` parameter:

```
$ oc edit namespace <name>
```

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/node-selector: "type=user-node,region=east" ①
    openshift.io/sa.scc.mcs: s0:c17,c14
    openshift.io/sa.scc.supplemental-groups: 1000300000/10000
    openshift.io/sa.scc.uid-range: 1000300000/10000
  creationTimestamp: 2019-06-10T14:39:45Z
  labels:
    openshift.io/run-level: "0"
  name: demo
  resourceVersion: "401885"
  selfLink: /api/v1/namespaces/openshift-kube-apiserver
  uid: 96ecc54b-8b8d-11e9-9f54-0a9ae641edd0
spec:
  finalizers:
  - kubernetes
  status:
    phase: Active
```

For example, to label a node:

```
$ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
```

- Lorsque qu'on crée un pod dans le namespace, OpenShift Container Platform ajoute la `<clé>:<valeur>` appropriée et schedule le pod sur le node labélisé.

For example:

```
spec:
  nodeSelector:
    region: east
    type: user-node
```

① Add `openshift.io/node-selector`` with the appropriate `<key>:<value>` pairs.

# Annexes

# Annexe - OpenShift Concepts

The OpenShift operating  
system

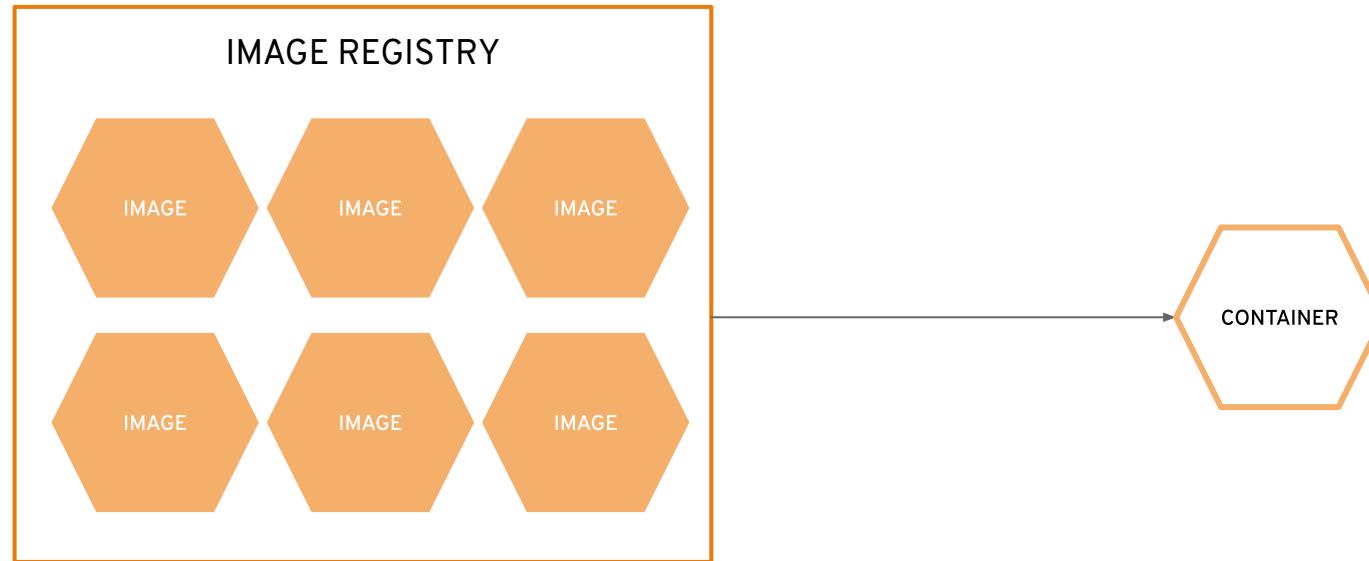
# a container is the smallest compute unit



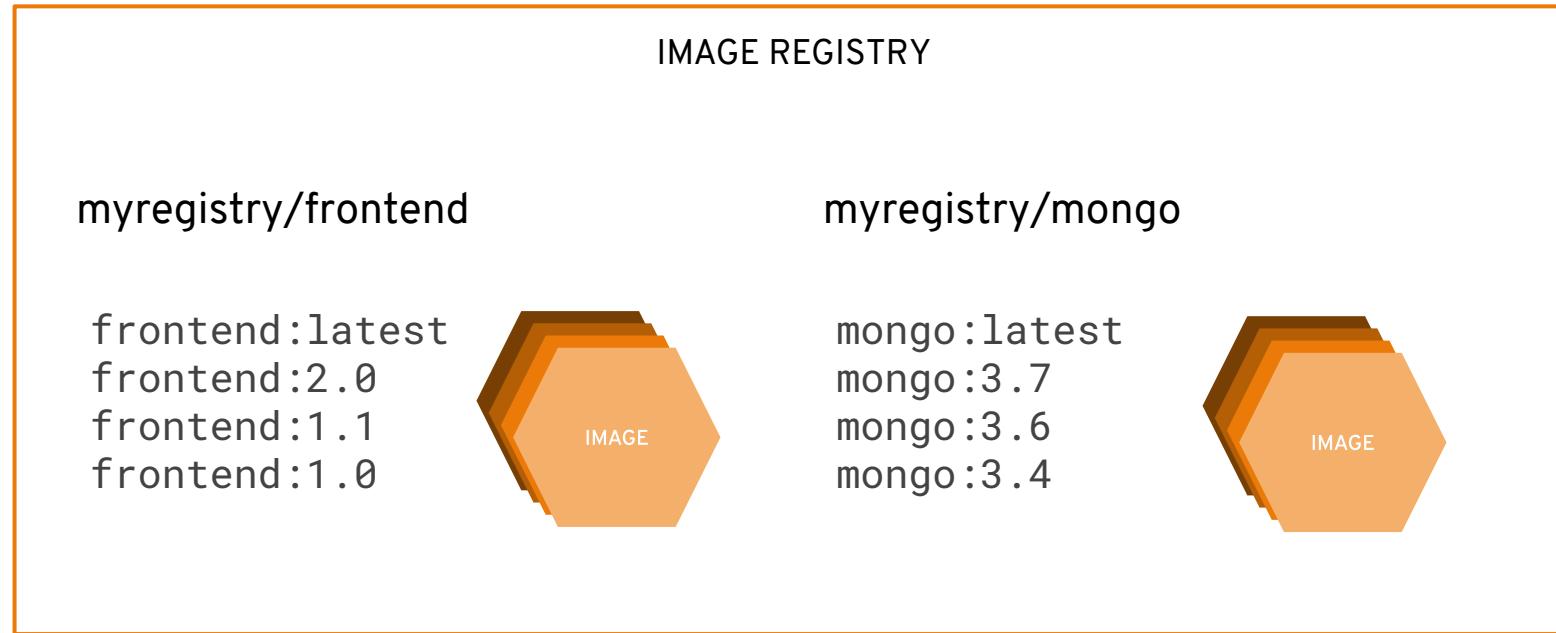
# containers are created from container images



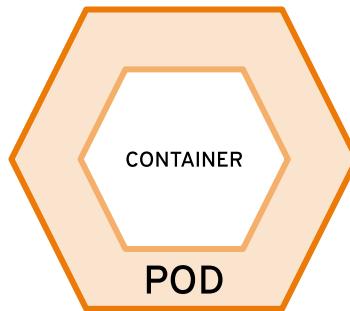
# container images are stored in an image registry



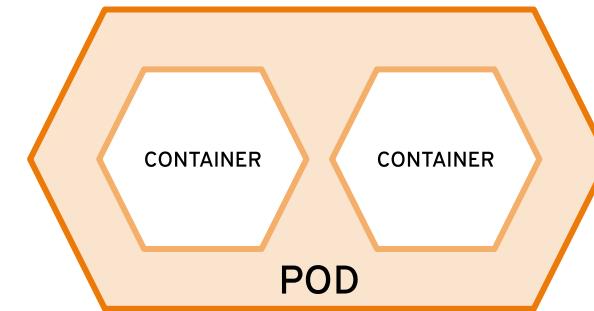
# an image repository contains all versions of an image in the image registry



containers are wrapped in pods which are units of deployment and management

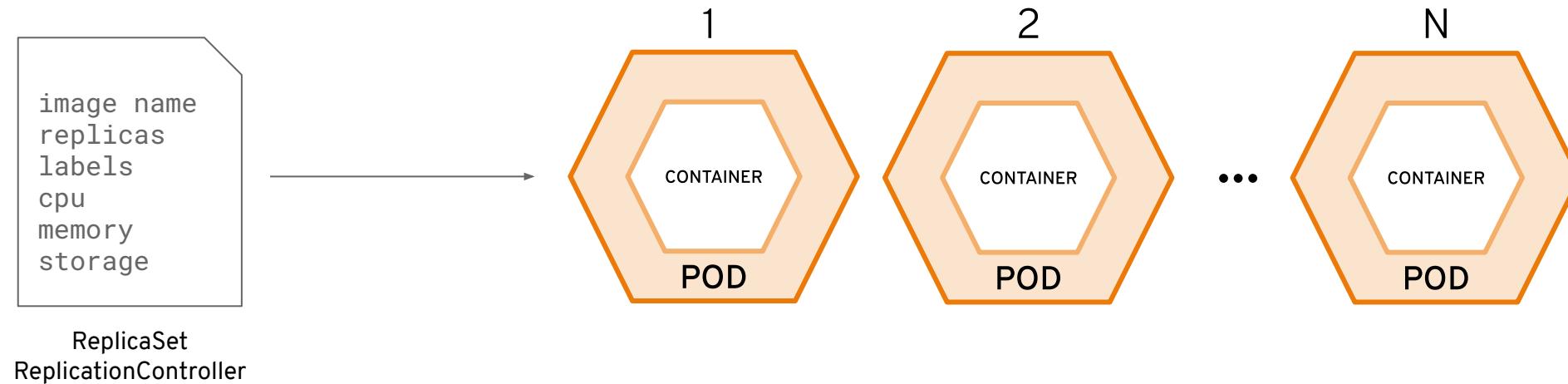


10.140.4.44

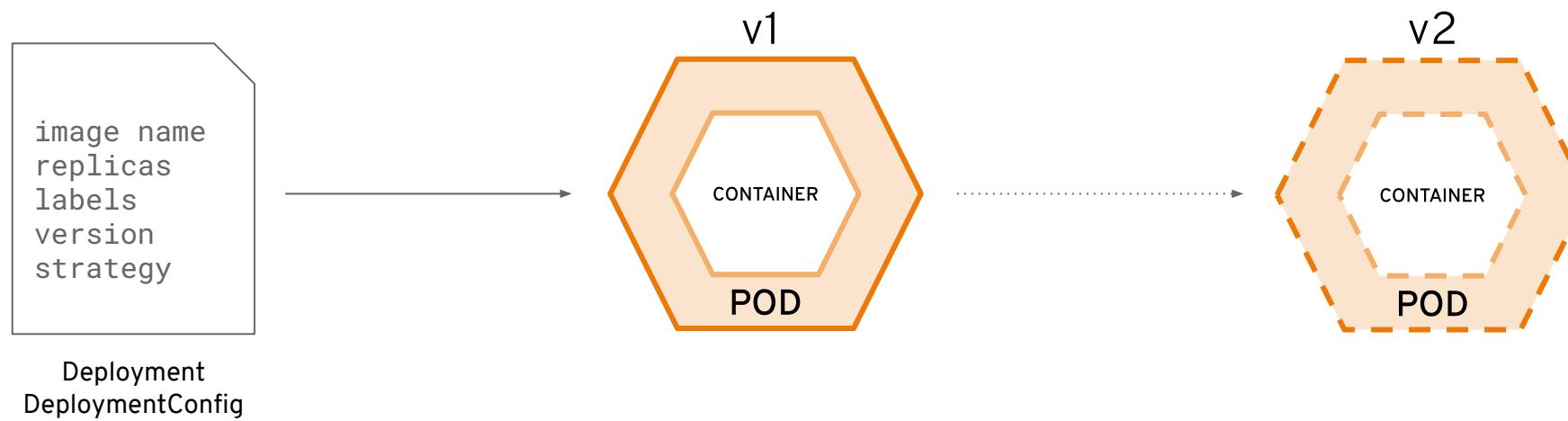


10.15.6.55

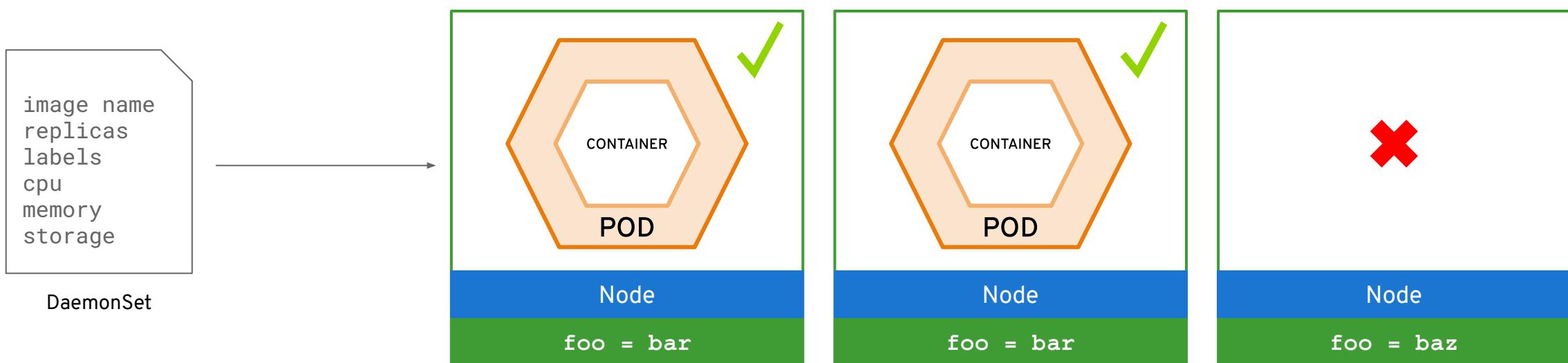
# ReplicationControllers & ReplicaSets ensure a specified number of pods are running at any given time



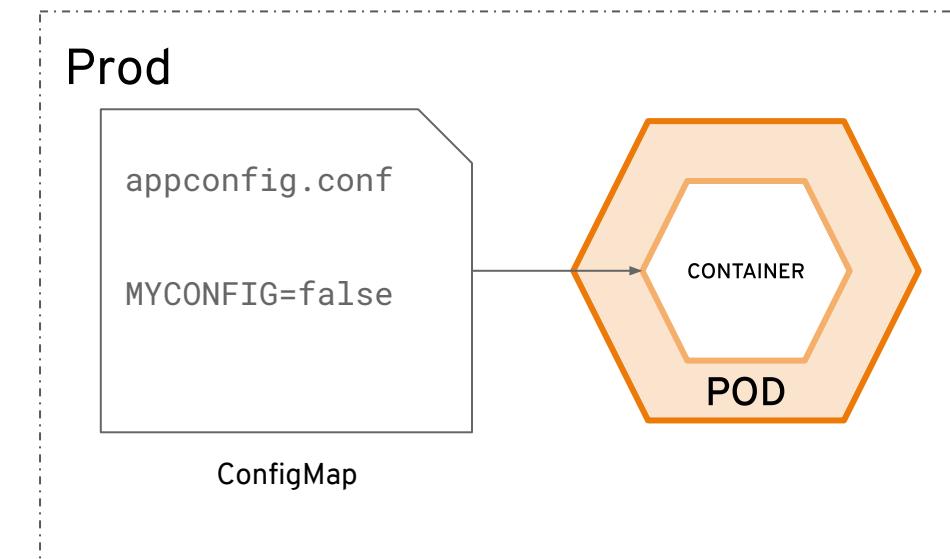
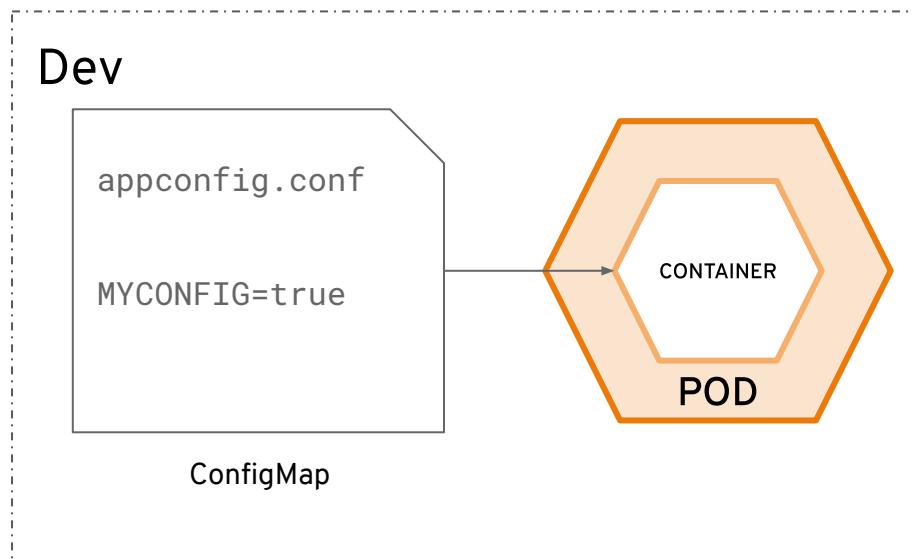
# Deployments and DeploymentConfigurations define how to roll out new versions of Pods



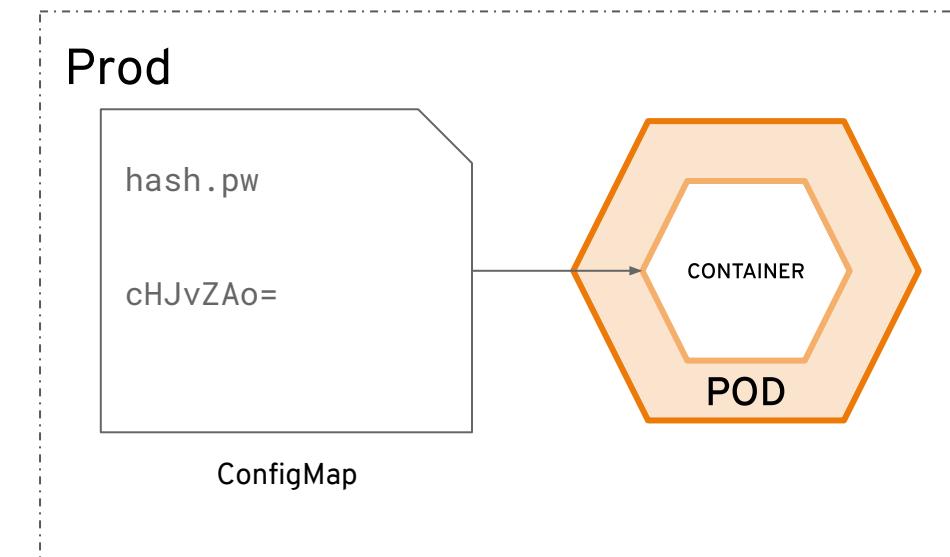
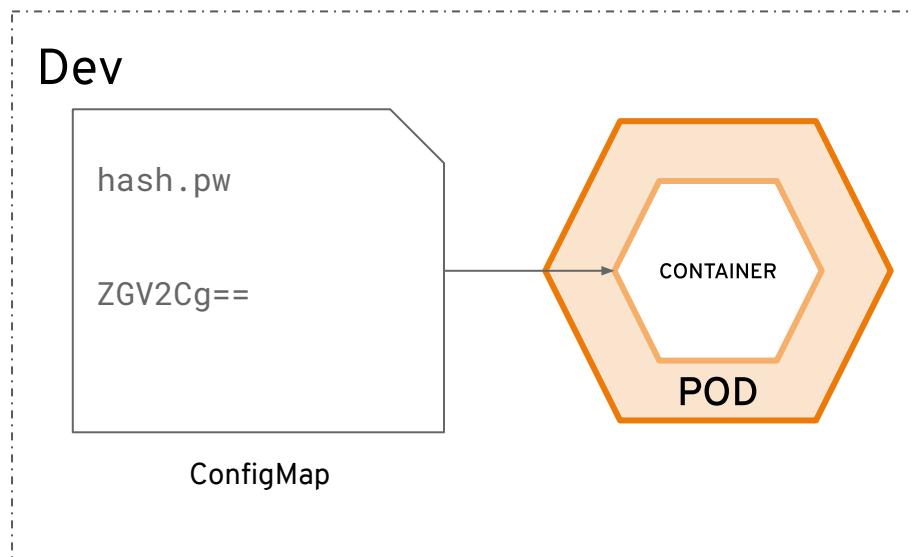
a daemonset ensures that all  
(or some) nodes run a copy of a pod



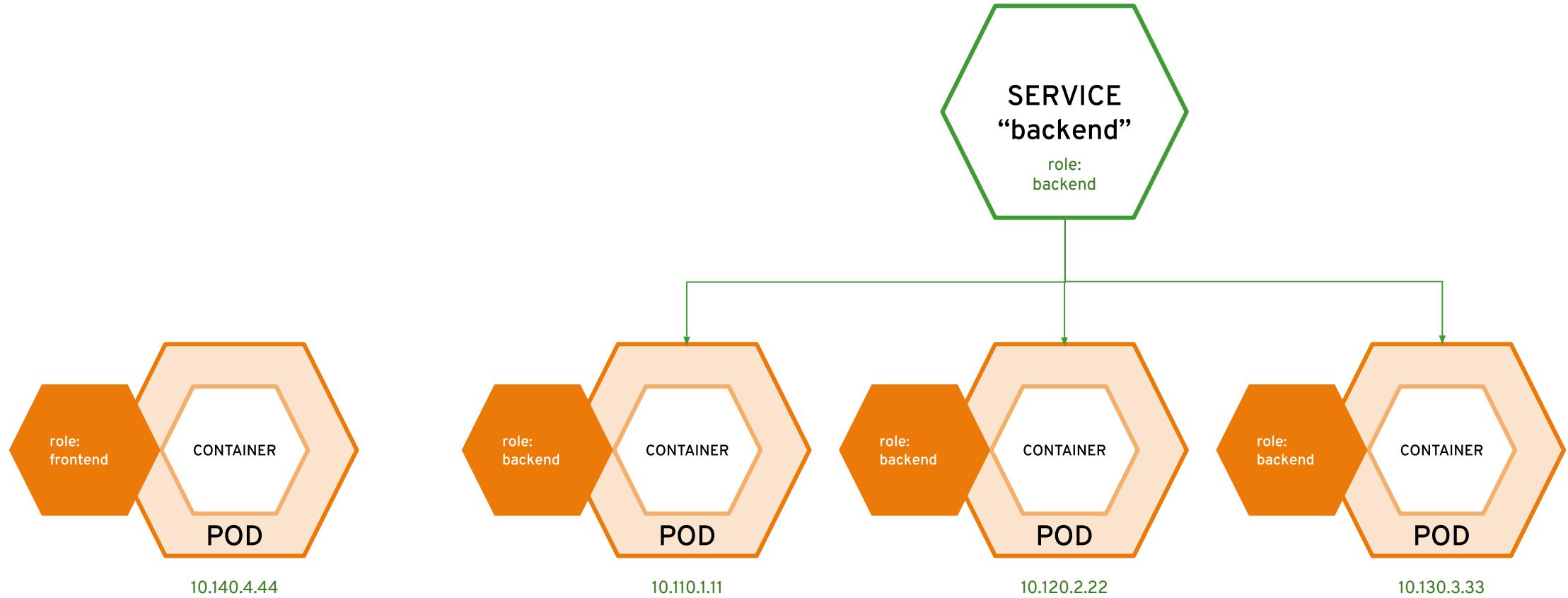
# configmaps allow you to decouple configuration artifacts from image content



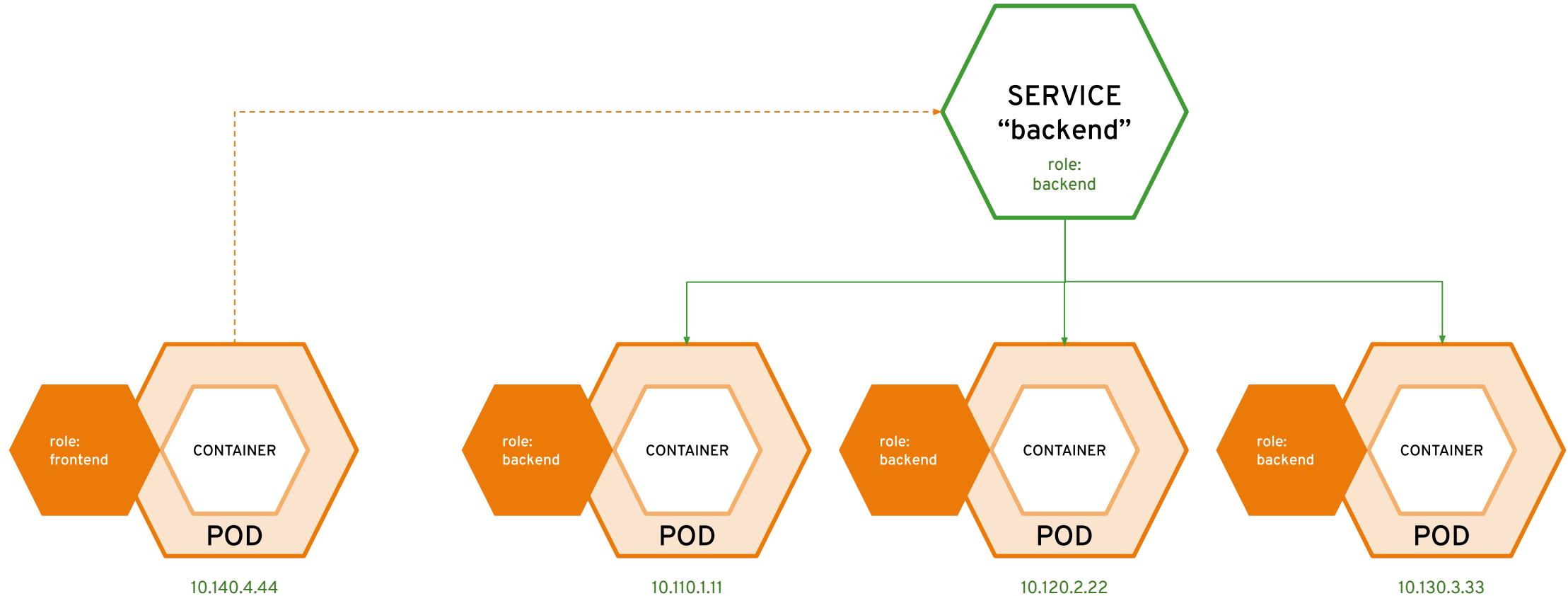
secrets provide a mechanism to hold sensitive information such as passwords



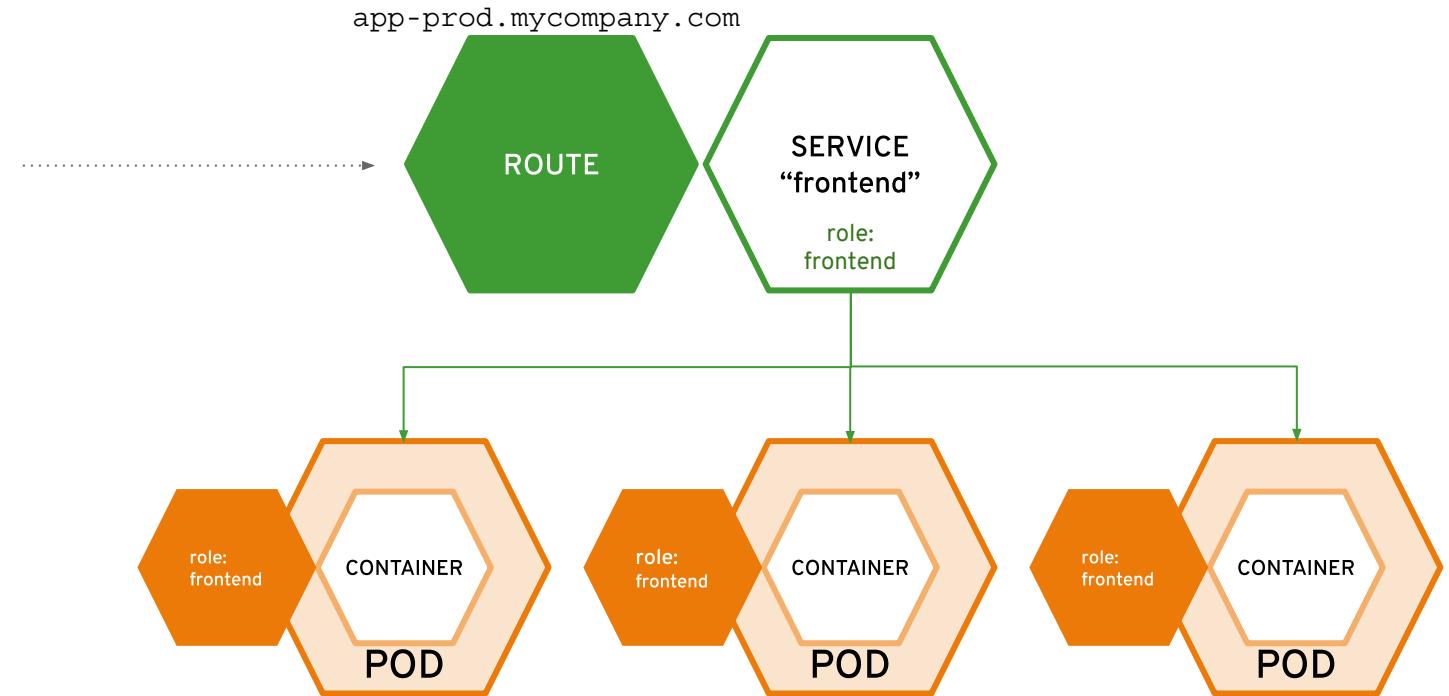
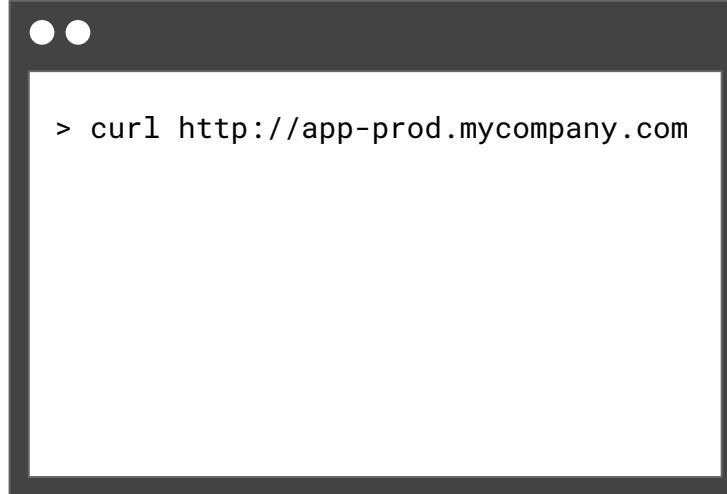
# services provide internal load-balancing and service discovery across pods



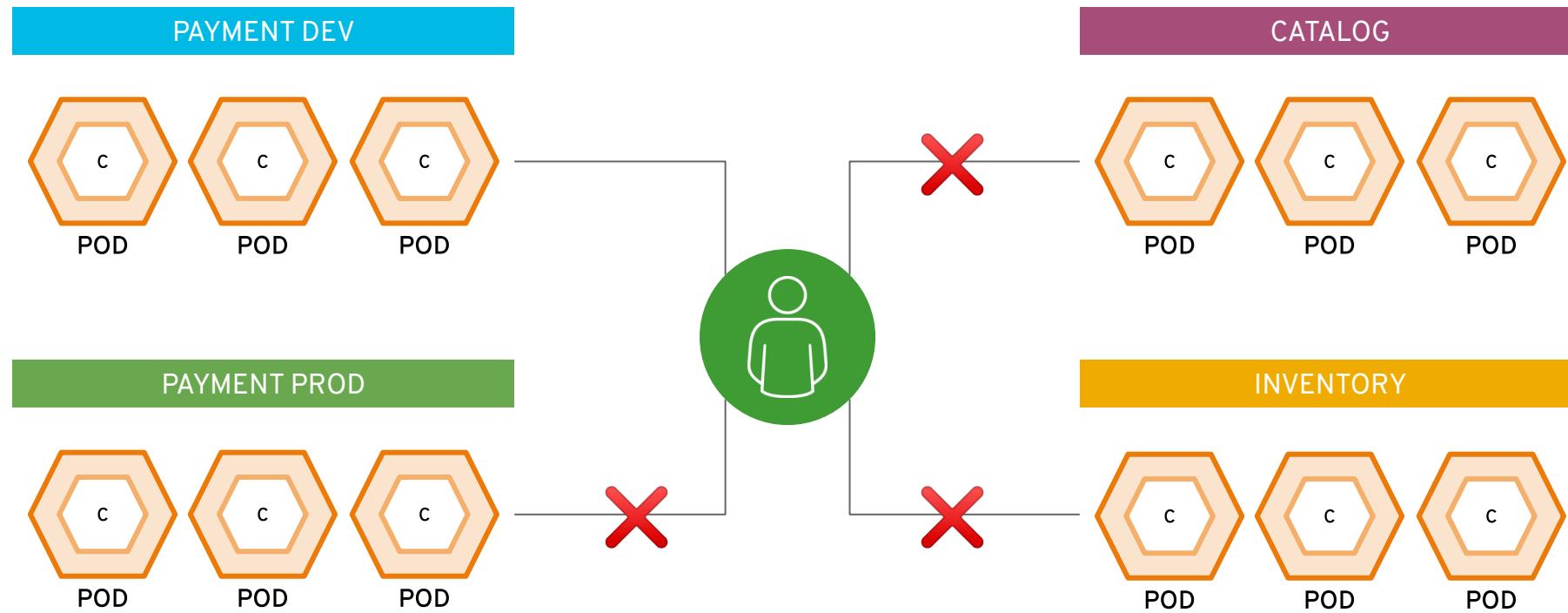
# apps can talk to each other via services



# routes make services accessible to clients outside the environment via real-world urls



projects isolate apps across environments,  
teams, groups and departments

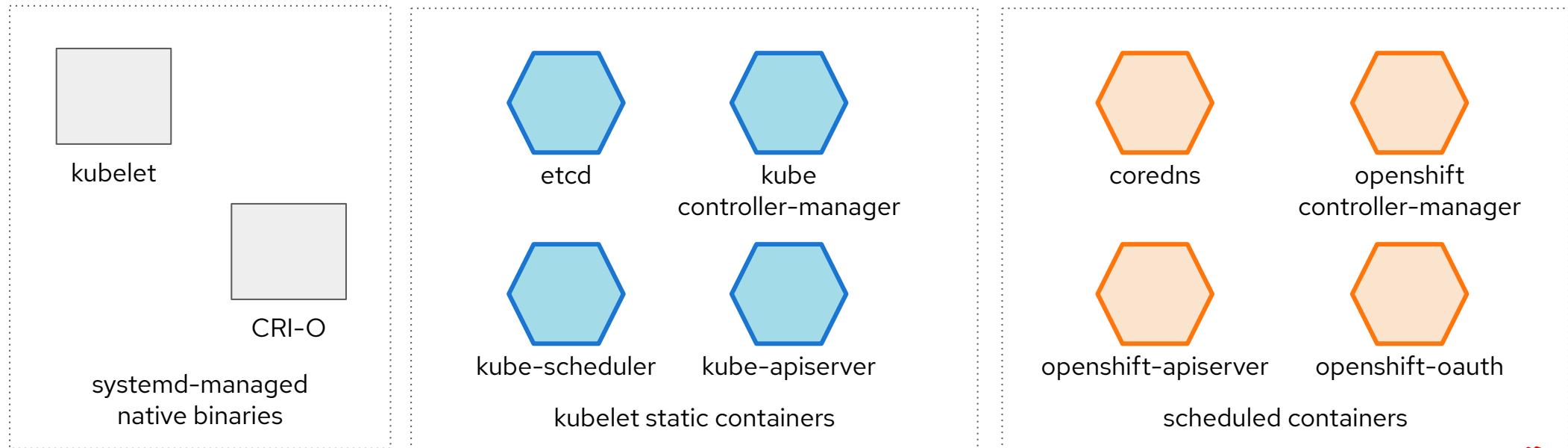


# Annexe - CoreOS

The OpenShift operating  
system

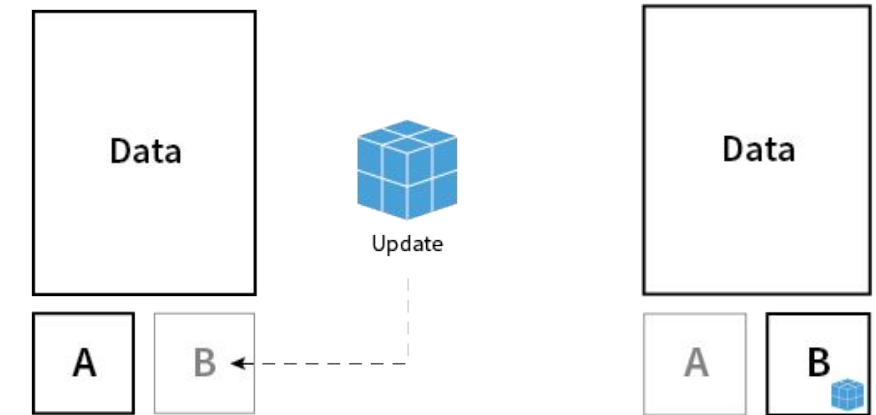
- CoreOS est un système d'exploitation de type appliance optimisé pour les Containers, conçu explicitement comme un composant d'OpenShift où tout fonctionne comme un Container.
- Cependant, tout ne fonctionne pas comme un Container \_orchestré\_ programmé par Kubernetes.
- Ce schéma montre ici la nature des pods "**static**" par rapport aux pods "**scheduled**" ou "**dynamic**" sur un hôte CoreOS.

## CoreOS “pod” architecture



- Les mises à jour transactionnelles garantissent que RHEL CoreOS n'est jamais altéré pendant l'exécution. Il est plutôt démarré directement dans une version toujours "bonne".
  - Chaque mise à jour de l'OS est versionnée et testée comme une image complète.
  - Les binaires de l'OS (/usr) sont en lecture seule
  - Les mises à jour du système d'exploitation encapsulées dans les images des Containers
  - système de fichiers et couches de paquets disponibles pour les correctifs et le débogage

## Transactional updates with rpm-ostree

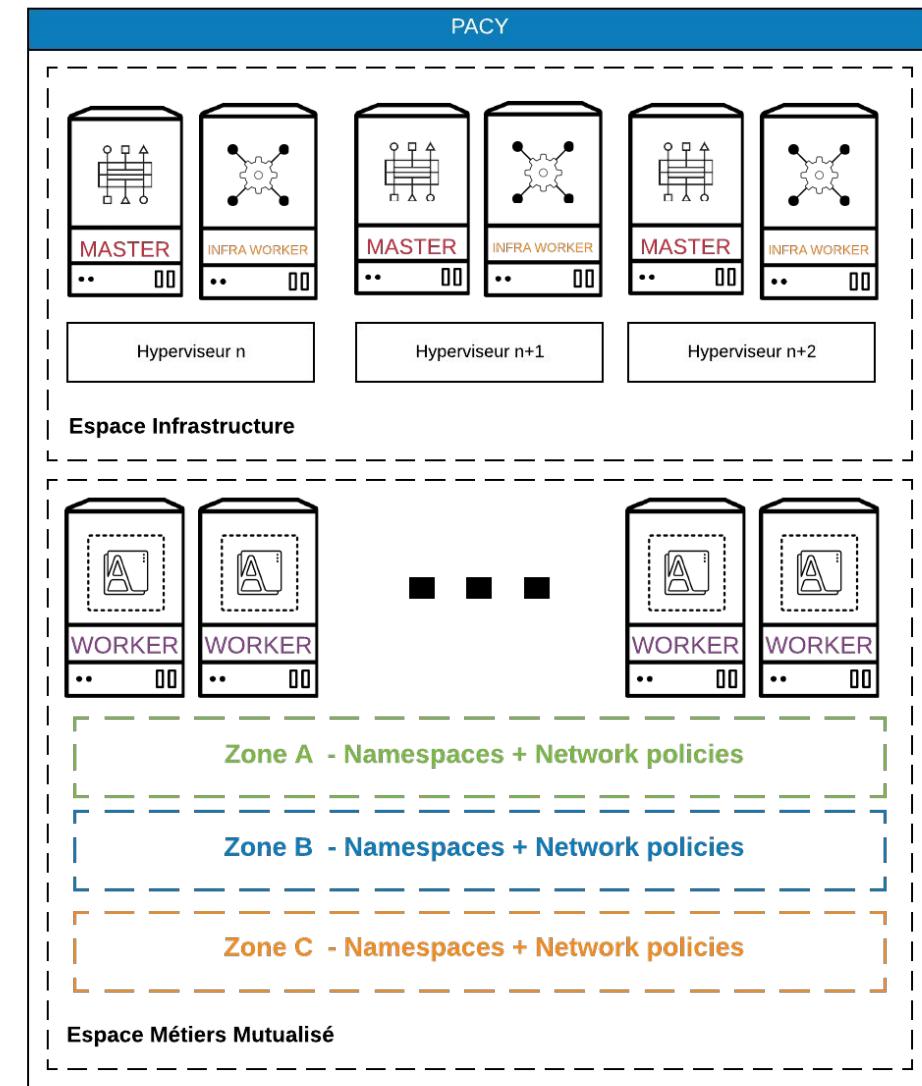
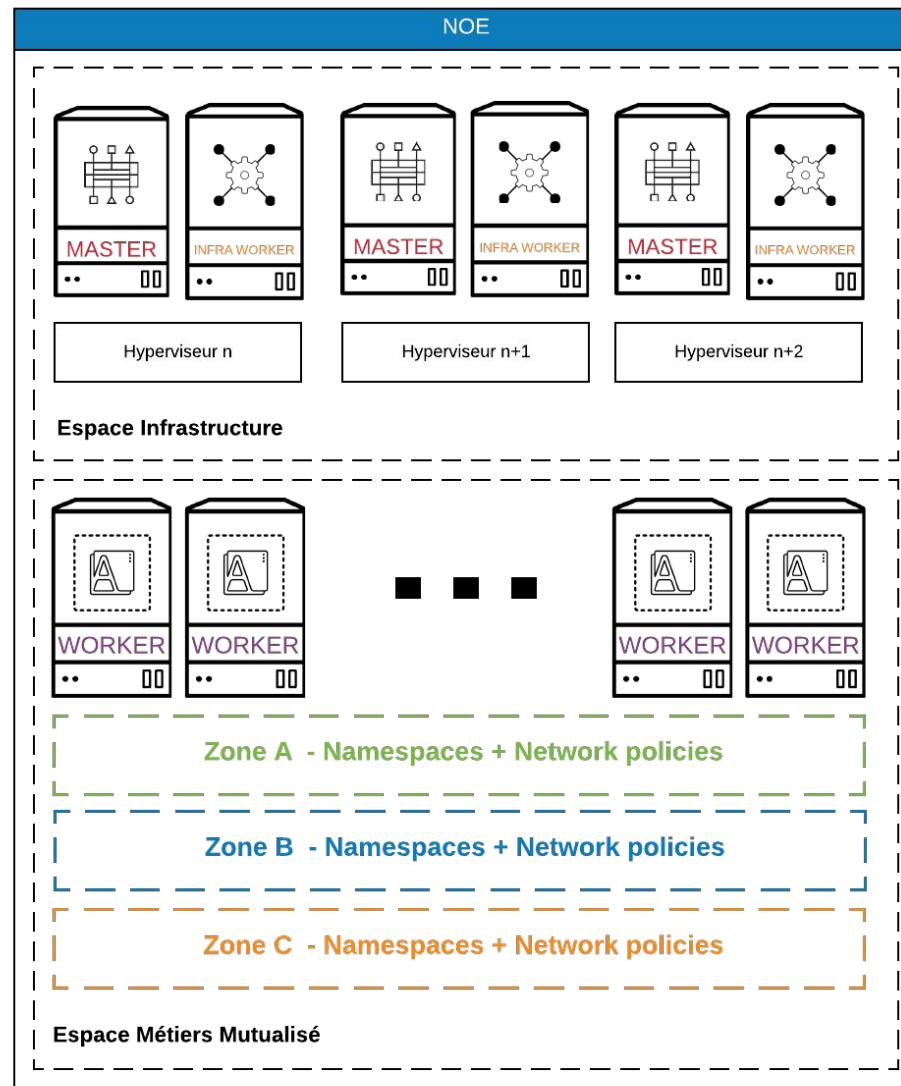


# Annexe - Namespace

The OpenShift operating  
system



## LB / GSLB



# Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



[twitter.com/RedHat](https://twitter.com/RedHat)