OpenShift Virtualization

Avec le stockage externe Red Hat Ceph Storage 5

Optimisation et performances à grande échelle

Architecture de référence

Boaz Ben Shabat

Sommaire

Public

Ce document vise à aider les responsables des services d'infrastructure, notamment les clients, les ingénieurs commerciaux, les consultants sur le terrain et les architectes de solutions. Vous y trouverez un exemple de déploiement à grande échelle d'OpenShift Virtualization, une fonction de la solution Red Hat OpenShift® Container Platform, avec Red Hat Ceph Storage (RHCS) comme solution externe de stockage réseau à haute disponibilité.

Synthèse

Ce document reprend les conclusions de l'équipe chargée des performances et de la mise à l'échelle de Red Hat OpenShift Virtualization, réunies lors d'un déploiement réussi à grande échelle qui intégrait un cluster externe Red Hat® Ceph® Storage 5 (47 nœuds) et la fonction Red Hat OpenShift Virtualization (100 nœuds). Les machines virtuelles d'OpenShift Virtualization ont été stockées dans le cluster externe Ceph, pour un total de 3 000 machines virtuelles et 21 400 pods.

Cette architecture de référence explique les étapes nécessaires à l'optimisation de RHCS et Red Hat OpenShift Virtualization, qui a permis de créer un cluster OpenShift de 100 nœuds résilient.

Nous présenterons également le raisonnement qui a conduit à ces étapes et indiquerons la manière dont ces recommandations peuvent s'appliquer à tout autre cluster.

Le tableau ci-dessous fournit les résultats de performances pour chacun des principaux scénarios qui pourraient avoir lieu dans un environnement de production.

Scénario	Description	Résultat
Déploiement de machines virtuelles	Jusqu'à 800 machines virtuelles déployées simultanément	D'après les résultats des tests, il est possible d'atteindre la vitesse de déploiement maximale en clonant les machines virtuelles par groupes de 100.
« Boot storms » de machines virtuelles	Jusqu'à 1 000 machines virtuelles démarrées en même temps	Les démarrages quasi linéaires ont commencé à 1 minute 42 secondes pour 100 machines virtuelles et ont pris fin à 17 minutes 45 secondes pour 1 000 machines virtuelles.
Latence des machines virtuelles	Comparaison de la latence au repos à long terme et de la latence des charges de travail en lecture et en écriture	Le nombre de threads d'entrée/sortie (E/S) qui accèdent à RHCS n'a aucun effet sur la latence au repos des machines virtuelles, en comptant 1 million d'entrées/sorties par seconde (IOPS). Jusqu'à 30 % de baisse pour la latence en lecture et jusqu'à 88 % de hausse pour la latence en écriture.
Migration des machines virtuelles	Migration de 1 000 machines virtuelles	La migration de 1 000 machines virtuelles et l'éviction de 7 000 pods ont demandé environ 118 minutes.
Augmentation de la latence due à la migration des machines virtuelles	Migration de 1 000 machines virtuelles Red Hat Enterprise Linux® (RHEL) avec leur charge de travail	La latence d'E/S au cours de la migration a augmenté de 9 % en lecture et de 13 % en écriture ; la durée de migration a augmenté de 3 % et le taux réel des IOPS est resté le même.
Mise à niveau du cluster OpenShift	Mise à jour de la version du cluster OpenShift	La mise à niveau mineure a demandé 35 minutes et la mise à niveau majeure a pris 136 minutes.

Composants logiciels

Produit	Version	Description
Red Hat OpenShift	4.9.15	Plateforme Kubernetes de pointe pour les entreprises qui offre une expérience digne du cloud, partout où elle est déployée. Que ce soit dans le cloud, sur site ou en périphérie du réseau, Red Hat OpenShift vous donne la possibilité de choisir où créer, déployer et exécuter vos applications grâce à une expérience cohérente.
Red Hat Ceph Storage	5	Solution de stockage Open Source simplifiée et extrêmement évolutive pour les pipelines de données modernes. Conçue pour l'analyse des données, l'intelligence artificielle/apprentissage automatique (IA/AA) et les charges de travail émergentes, la solution Red Hat Ceph Storage fournit un stockage logiciel sur le matériel standard de votre choix.
Red Hat OpenShift Data Foundation	4.9.2	Solution de stockage logiciel pour les conteneurs. Plateforme de services de données et de stockage créée pour Red Hat OpenShift, elle aide les équipes à développer et déployer des applications rapidement et efficacement dans différents clouds et hôtes bare metal.
Red Hat OpenShift Virtualization	4.9.2	Solution de Red Hat pour l'exécution de machines virtuelles dans un cluster Kubernetes. D'une part, elle vise à aider tous les utilisateurs, quel que soit leur niveau d'expérience avec les machines virtuelles, à consolider leurs charges de travail sur une seule plateforme afin de réduire les coûts d'exploitation liés à la gestion d'une plateforme de virtualisation en plus d'une plateforme de conteneurs. D'autre part, cette solution est conçue pour s'appuyer sur la puissance du moteur et de l'écosystème Kubernetes pour aider les utilisateurs à moderniser les capacités, l'orchestration et l'architecture qu'ils utilisent habituellement pour leurs charges de travail.

Composants physiques

Cluster RHCS

10 serveurs rack DELL PowerEdge R640 :

Composant	Caractéristiques	Remarques
Processeur	40 cœurs	2 processeurs Intel(R) Xeon(R) Gold 6230, 2,10 GHz
Mémoire	384 Go de RAM ECC	12 SK Hynix, 1 barrette dual rank DDR4-3200 RDIMM PC4-25600R de 32 Go, 4 modules
SSD (disque racine)	446,63 Go, 6 Gb/s	SSD MICRON MTFDDAK480TDT
SSD (stockage)	3 574 Go, 12 Gb/s	2 SSD TOSHIBA KPM5XVUG1T92 de 1 787,88 Go
NVMe (stockage)	2 980,82 Go, 8 GT/s	NVMe Samsung S5CXNA0N607551

37 serveurs rack DELL PowerEdge R650 :

Composant	Caractéristiques	Remarques
Processeur	56 cœurs	2 processeurs Intel(R) Xeon(R) Gold 6330, 2 GHz
Mémoire	384 Go de RAM ECC	12 SK Hynix, 1 barrette dual rank DDR4-3200 RDIMM PC4-25600R de 32 Go, 4 modules
SSD (disque racine)	446,63 Go, 6 Gb/s	SSD MICRON MTFDDAK480TDT
SSD (stockage)	3 574 Go, 12 Gb/s	2 SSD TOSHIBA KPM5XVUG1T92 de 1 787,88 Go
NVMe (stockage)	2 980,82 Go, 8 GT/s	NVMe Samsung S5CXNA0N607551

Remarque : le matériel utilisé pour RHCS n'était pas entièrement adapté à la tâche, car l'hétérogénéité de la taille des disques et de l'architecture au sein du cluster RHCS a réduit les performances de Ceph. Cependant, Ceph prouve une fois de plus sa polyvalence.

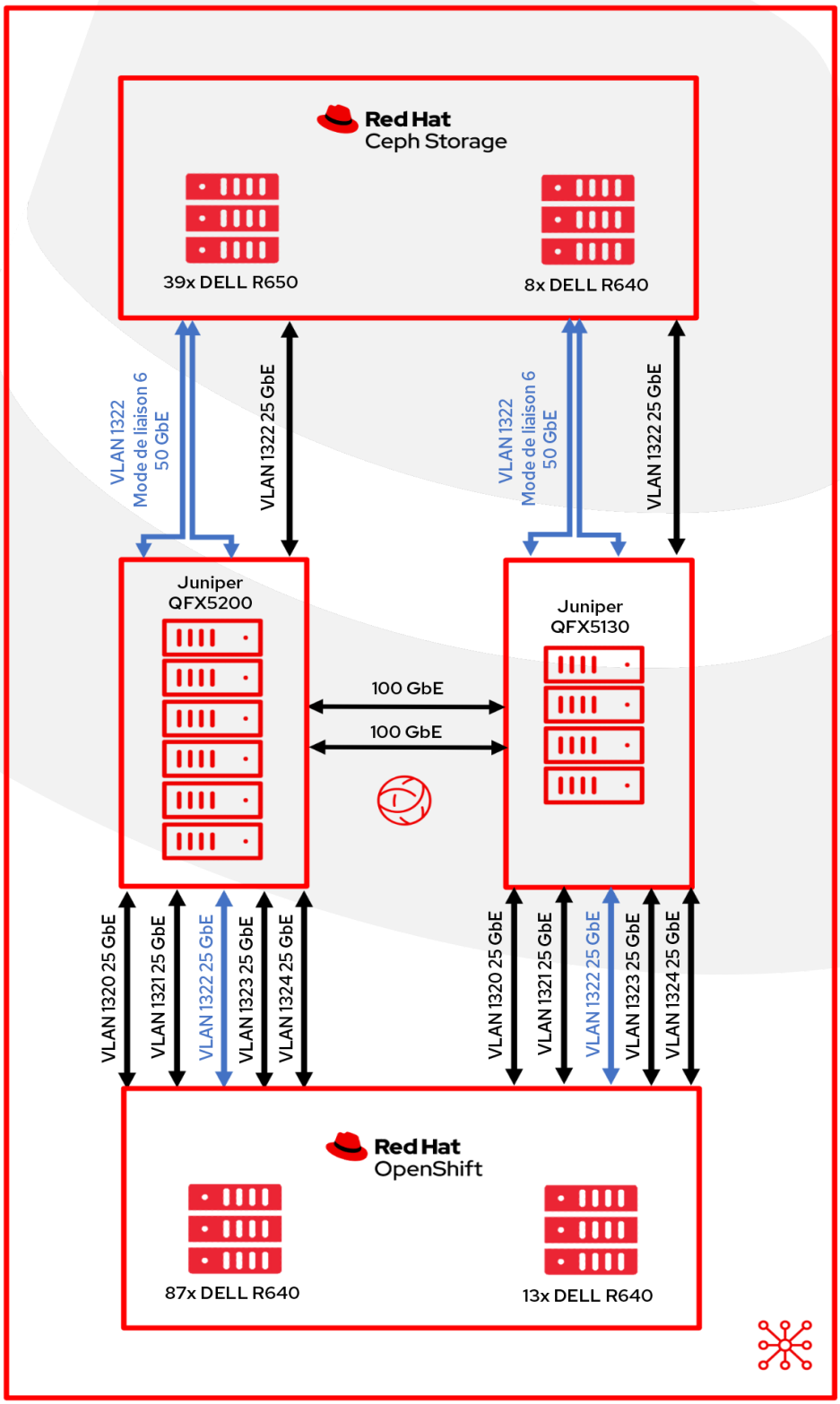
Cluster OpenShift

100 serveurs rack DELL PowerEdge R640 :

Composant	Caractéristiques	Remarques
Processeur	40 cœurs	2 processeurs Intel(R) Xeon(R) Gold 6230, 2,10 GHz et 20 cœurs
Mémoire	384 Go de RAM ECC	12 SK Hynix, 1 barrette dual rank DDR4-3200 RDIMM PC4-25600R de 32 Go, 4 modules
SSD (disque racine)	446,63 Go, 6 Gb/s	SSD MICRON MTFDDAK480TDT

Architecture

Ce diagramme montre l'architecture réseau des clusters OpenShift et Ceph. Au sein d'un VLAN privé, le chemin de données entre les clusters Ceph et Red Hat OpenShift Container Platform (OCP) utilise la liaison balance-alb avec deux ports de 25 GbE.



Réglage réseau RHCS

Cette section décrit les réglages du réseau Linux à effectuer sur les nœuds Ceph afin de s'adapter à cet environnement à grande échelle.

Réglage ARP (Address Resolution Protocol)

Flux ARP

Tout hôte Linux qui compte plusieurs interfaces réseau sur le même sous-réseau peut être confronté à des problèmes de flux ARP. Ces derniers peuvent survenir lorsqu'un hôte répond à une demande ARP pour des interfaces sur le même sous-réseau. Si ce comportement ne pose pas toujours problème, il arrive que le flux ARP entraîne un dysfonctionnement de certaines applications en raison d'une mise en correspondance incorrecte entre les adresses IPv4 et MAC.

Sur les hôtes basés sur RHEL, il est possible d'éliminer ce comportement en ajoutant les lignes suivantes au fichier /etc/sysctl.d/99.8-arp.conf de tous les hôtes RHCS :

```
net.ipv4.conf.all.arp_filter=1 #default value 0
net.ipv4.conf.all.arp_ignore=1 #default value 0
net.ipv4.conf.all.arp_announce=1 #default value 0
```

- **filter=1** : cet élément vous permet de multiplier les interfaces réseau sur le même sous-réseau et de vous assurer que les réponses aux ARP de chaque interface dépendent du fait que le noyau achemine ou non un paquet de l'IP ARP à partir de cette interface (pour cela, vous devez utiliser un routage basé sur la source). En d'autres termes, vous pouvez choisir les cartes (généralement une) qui répondront à une demande ARP.
- **ignore=1** : une réponse est envoyée uniquement si l'adresse IP cible correspond à une adresse locale configurée sur l'interface entrante.
- **arp_announce=1** : essayez d'éviter les adresses locales non présentes dans le sous-réseau de la cible pour cette interface. Ce mode se révèle utile lorsque les hôtes cibles accessibles via cette interface ont besoin que l'adresse IP source des demandes ARP soit intégrée à leur réseau logique configuré sur l'interface réceptrice.

Veillez à charger les nouveaux paramètres réseau en exécutant la ligne suivante :

```
$ sysctl -p /etc/sysctl.d/99.8-arp.conf
```

Cache ARP

Le cache ARP conserve une liste des entrées ARP, générées lorsqu'une adresse IP devient une adresse MAC. Pour éviter les situations où le cache ARP ne peut pas contenir l'ensemble des entrées en raison de leur grand nombre, vous devez augmenter la taille du cache en ajoutant les lignes suivantes au fichier **/etc/sysctl.d/99.7-arpcachesize.conf** :

```
net.ipv4.neigh.default.gc_thresh1 = 4096 #default value 128
net.ipv4.neigh.default.gc_thresh2 = 16384 #default value 512
net.ipv4.neigh.default.gc_thresh3 = 32768 #default value 1024
```

La valeur numérique marque le seuil à partir duquel nous commencerons à nettoyer le cache pour la destination IPv4. Si cette valeur double, le système refuse toute nouvelle allocation.

Réglage TCP/IP

Mise à l'échelle de la fenêtre TCP

Les paramètres réseau par défaut de RHEL n'apportent pas toujours un débit et une latence suffisants pour les tâches volumineuses exécutées en parallèle, généralement dans des configurations à grande échelle. Voici comment régler le réseau Linux et certains périphériques réseau pour améliorer les performances des tâches exécutées en simultané :

Pour tirer le meilleur parti des réseaux à bande passante élevée, il faut utiliser une grande fenêtre TCP.
Nous nous sommes donc assurés d'activer la mise à l'échelle de la fenêtre TCP.
Il est possible de vérifier ce paramètre avec : - **cat /proc/sys/net/ipv4/tcp_window_scaling**

```
$ sysctl -w net.ipv4.tcp_window_scaling=1
```

Pour garantir la persistance après chaque redémarrage :

```
$ echo "net.ipv4.tcp_window_scaling=1" >> /etc/sysctl.conf
```

Réglage de la mémoire tampon

Étape suivante : calculer le socket « envoyer la mémoire tampon » et « recevoir la mémoire tampon ».
De manière générale, la mémoire tampon de lecture et d'écriture pour chaque socket peut contenir soit 2 paquets au minimum, soit 4 paquets par défaut, soit 10 paquets au maximum. Si la mémoire tampon du socket réseau est trop réduite, elle risque de se remplir et de réduire le débit effectif, ce qui diminuera les performances. En revanche, si elle est suffisamment grande, elle peut améliorer les performances dans une certaine mesure.

Quelques définitions avant de commencer :

- rmem_max : valeur maximale pour « recevoir la mémoire tampon ».
- wmem_max : valeur maximale pour « envoyer la mémoire tampon ».
- wmem_default : valeur par défaut pour « envoyer la mémoire tampon ».
- max_backlog : taille maximale pour la file d'attente de la réception.
- Netdev_budget : nombre maximal de paquets récupérés dans toutes les interfaces en un cycle d'attente active.

Nous avons calculé la mémoire tampon optimale en fonction de la taille des paquets. Néanmoins, nous conseillons de prendre en compte la latence pour les configurations à latence élevée.

Méthode basée sur la latence

Il s'agit d'optimiser en s'appuyant sur le calcul du débit maximal d'une connexion TCP unique en fonction de la latence.

Taille optimale = (délai d'aller-retour en microsecondes) × (taille du lien en Mb/s) × 1 024^2

Par exemple, notre liaison s'exécute à 50 000 Mb/s. La latence du nœud Ceph au cluster OpenShift est de 0,208. Divisez ce résultat par 1 000 pour obtenir une valeur en microsecondes, puis multipliez par 50 000. Résultat : 10,4, soit 10 905 190 octets.

Ou :

```
ping -Ibond0 -c 60 -q 192.168.216.90|grep avg|awk -F"/" '{printf "%f", ($5
/1000) * 50000 * 1024^2}'
```

Il suffit maintenant sw définir le paramètre wmem_default qui contient quatre paquets, c'est-à-dire un quart de 10 905 190. Voici ce que l'on obtient :

```
net.core.rmem_max=10905190 #default value 212992
net.core.wmem_max=10905190 #default value 212992
net.core.wmem_default=4362076 #default value 212992
```

Méthode basée sur les paquets

Autre méthode possible : l'optimisation en fonction de la taille des paquets. Il s'agit de baser la taille optimale de chaque socket sur la taille moyenne des paquets par fichier, par exemple 512 Ko, et d'effectuer le calcul suivant : taille maximale = (taille du paquet en Mb/s) × 1 024^2.

```
net.core.rmem_max=5242880 #default value 212992
net.core.wmem_max=5242880 #default value 212992
net.core.wmem_default=2097152 #default value 212992
```

Gardez à l'esprit que l'optimisation en fonction de la latence est mieux adaptée aux réseaux qui fonctionnent souvent avec une latence élevée.

Réglage de l'adaptateur

Mémoire tampon de la carte réseau

Dans les configurations à grande échelle qui contiennent plusieurs hôtes, le taux de trafic entrant peut être si élevé que le noyau n'arrive pas à nettoyer la mémoire tampon suffisamment vite. Dans ce cas, la mémoire tampon de la carte réseau se retrouve submergée et le trafic est perdu. Il est considéré comme une perte due à une interruption logicielle (softirq). Une solution consiste à augmenter le temps de processeur que l'interruption logicielle peut consommer, ce qui correspond au paramètre netdev_budget, et à augmenter le budget en conséquence. Dans notre configuration, nous avons fait monter le budget à 1 000 afin que l'interruption logicielle élimine 1 000 messages de la carte réseau avant de quitter le processeur.

```
net.core.netdev_budget=1000 #default value 300
```

Si la troisième colonne de **/proc/net/softnet_stat** augmente progressivement :

```
01877e29 00000000 00000022 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000005d
0c4a6107 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000005e
01d05820 00000000 00000012 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000005f
092b933a 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000060
```

Cela signifie que l'interruption logicielle ne disposait pas d'un temps de processeur suffisant. Il faut alors augmenter le budget, de préférence petit à petit.

File d'attente du backlog

Au sein du noyau Linux, une file d'attente stocke le trafic entre sa réception par la carte réseau et son traitement par l'une des piles de protocoles (TCP/IP/iSCSI).
Chaque cœur de processeur possède une file d'attente de backlog qui stocke le trafic. Si sa capacité maximale est atteinte, tout paquet supplémentaire est abandonné.

Si la deuxième colonne de **/proc/net/softnet_stat** augmente progressivement :

```
04f88d2c 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
023a354d 00000000 00000018 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000001
10df99e1 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000002
01ba2dec 00000000 00000011 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000003
```

Cela indique le débordement de la file d'attente du backlog netdev et le besoin d'augmenter la valeur netdev_max_backlog, là encore progressivement.

Nous avons défini une valeur de 5 000 pour ce paramètre de mise à l'échelle.

```
net.core.netdev_max_backlog=5000 #default value 1000
```

Réglage RHCS

Cette section décrit les réglages Ceph à effectuer sur les nœuds Ceph afin de s'adapter à cet environnement à grande échelle.

Réglage des groupes de placement

Les groupes de placement sont un ensemble d'objets répliqués par un périphérique de stockage d'objets. Chaque objet est un conteneur de stockage pour les données et métadonnées.

Pour réunir un nombre optimal de groupes de placement par pool, il faut paramétrer la cible sur 100 groupes de placement par périphérique de stockage d'objets (conformément aux meilleures pratiques pour RBD et librados), la multiplier par la capacité d'utilisation maximale du pool (85 % par défaut), la diviser par le nombre de répliques, puis arrondir le résultat à la puissance la plus proche de 2 - 2^(round(log2(x))) :

```
( Groupes de placement cibles par périphérique de stockage d'objets ) x ( périphériques de stockage d'objets ) x ( %Data (capacité d'utilisation maximale du pool))
----------------------------------------------------------------------
                              ( Trois répliques )
```

Dans notre configuration : (100 × 141 × 0,85) / 3 = 3 995, arrondi à une puissance de 2, soit 4 096 groupes de placement au total.

Nous en avons fait un script avec une calculatrice de base :

```
$ echo "x=l(100*141*0.85/3)/l(2); scale=0; 2^((x+0.5)/1)" | bc -l
```

Par ailleurs, vous pouvez augmenter davantage le nombre de groupes de placement par périphérique de stockage d'objets pour tenter de réduire la variance de la charge par périphérique au sein de votre cluster. Toutefois, comme chaque groupe consomme une plus grande part des ressources du processeur et de la mémoire des périphériques de stockage d'objets qui les stocke, il faut tester et régler le nombre de périphériques au cas par cas.

Étape suivante : appliquer ces paramètres au cluster :

```
$ ceph osd pool set pool_name pg_autoscaler_mode off
$ ceph osd pool set pool_name pg_num 4096
```

Réglage Prometheus

Pour surveiller le cluster Ceph, il est possible d'afficher les statistiques du pool dans le tableau de bord Ceph. Pour ce faire :

```
$ ceph config set mgr mgr/prometheus/rbd_stats_pools pool_name
```

Pour atténuer la charge sur le système des clusters volumineux, il est possible de plafonner les statistiques du pool avec la commande suivante :

```
$ ceph config set mgr mgr/prometheus/rbd_stats_pools_refresh_interval 600
#Default value 300
```

De plus, il peut être utile de diminuer la fréquence d'attente active de Prometheus afin d'éviter que le gestionnaire Ceph ne devienne un goulet d'étranglement et que d'autres plug-ins ceph-mgr n'aient pas le temps de s'exécuter pour cette raison.

Dans ce cas, la commande définit l'intervalle de capture d'indicateurs, ou intervalle de « scraping », sur 60 secondes :

```
$ ceph config set mgr mgr/prometheus/scrape_interval 60 #Default value 15
```

Red Hat

OpenShift Virtualization

Introduction

Nous allons présenter les capacités et la stabilité d'OpenShift Virtualization à grande échelle à travers la démonstration des workflows suivants :

- Déploiement de machines virtuelles
- « Boot storm » de machines virtuelles
- Latence supplémentaire des machines virtuelles avec et sans charge de travail
- Migration de machines virtuelles avec et sans charge de travail

Cette configuration visait à atteindre une densité de 3 000 machines virtuelles et 21 400 pods dans tout le cluster. Voici les systèmes qui ont été nécessaires :

- 1 500 machines virtuelles RHEL 8.5 à stockage persistent
- 500 machines virtuelles Windows 10 à stockage persistent
- 1 000 machines virtuelles Fedora à stockage éphémère
- 21 400 pods inactifs

Plus simplement, la densité comprend 30 machines virtuelles et 214 pods par nœud.

KubeletConfig

Pour parvenir à l'échelle indiquée en introduction, nous devions contourner la limite par défaut de pods par nœud en appliquant ce paramètre KubeletConfig :

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: enabled
  kubeletConfig:
    maxPods: 500
    kubeAPIBurst: 200
```

```
    kubeAPIQPS: 100
```

En plus de faire monter la valeur `maxPods` à 500 (250 par défaut), nous avons augmenté les valeurs `kubeAPIBurst` et `kubeAPIQPS` à 200 et 100 (respectivement 100 et 50 par défaut) pour les adapter à une activité beaucoup plus élevée. Pour vous faire une idée générale, le nombre maximal de pods par défaut pour une instance Kubernetes standard est de 110 par nœud.

Notez qu'aucun des changements décrits ci-dessus n'est indispensable. Actuellement, il n'est pas recommandé de dépasser les 250 pods par nœud, en raison du manque de tests à long terme. Toutefois, nous n'avons pas remarqué de problème lié à la densité lors de nos tests.

Nous avons également appliqué un paramètre Kubeletconfig lié à BZ#1984442 afin d'équilibrer la distribution des pods des machines virtuelles dans tous les nœuds :

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-scheduling
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: enabled
  kubeletConfig:
    nodeStatusMaxImages: -1
```

Pour les nœuds de calcul, il est possible d'activer les deux paramètres Kubeletconfig personnalisés avec l'étiquette suivante :

```
oc label machineconfigpool worker custom-kubelet=enable
```

Remarque : les modifications apportées à KubeletConfig entraînent le redémarrage des nœuds associés.

Modèles

Tous les modèles de système d'exploitation que nous avons utilisés sont les modèles par défaut proposés par l'assistant de modèles OpenShift Virtualization. Nous avons seulement apporté quelques modifications à notre réseau personnalisé.

Red Hat Linux

Voici comment récupérer le modèle utilisé :

```
oc get templates -n openshift rhel8-server-medium -o yaml
```

Le voici en entier avec nos changements :

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: node-os-vm
  name: node-os-vm
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm:
    spec:
      terminationGracePeriodSeconds: 60
      evictionStrategy: LiveMigrate
      domain:
        cpu:
          cores: 1
          model: host-passthrough
          sockets: 1
          threads: 1
        devices:
          disks:
          - disk:
              bus: virtio
            name:
          interfaces:
          - bridge: {}
            model: virtio
            name: nic-0
          networkInterfaceMultiqueue: true
          rng: {}
        machine:
          type: pc-q35-rhel8.4.0
        resources:
          requests:
            cpu: "1"
            memory: 4G
      networks:
```

```
      - multus:
          networkName: linux-bridge
        name: nic-0
      volumes:
      - dataVolume:
          name:
        name:
  dataVolumeTemplates:
  - metadata:
      annotations
      name:
    spec:
      pvc:
        accessModes:
        - ReadWriteMany
        resources:
          requests:
            storage: 40Gi
        volumeMode: Block
        storageClassName: ocs-external-storagecluster-ceph-rbd
      source:
        pvc:
          namespace: "default"
          name: "rhel-dv"
```

Fedora

Voici comment récupérer le modèle utilisé :

```
oc get templates -n openshift fedora-desktop-medium -o yaml
```

Le voici en entier avec nos changements :

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    app:
    kubevirt-vm:
  name:
spec:
  annotations:
    descheduler.alpha.kubernetes.io/evict: "true"
    kubevirt.io/provisionOnNode:
  terminationGracePeriodSeconds: 0
  evictionStrategy: Restart
  running: true
  template:
    metadata:
      labels:
        kubevirt-vm: node-os-vm
    spec:
      domain:
```

```
        cpu:
          cores: 1
          sockets: 1
          threads: 1
        devices:
          disks:
            - disk:
                bus: virtio
              name: containerdisk
            - disk:
                bus: virtio
              name: cloudinitdisk
        machine:
          type: pc-q35-rhel8.4.0
        resources:
          requests:
            memory: 256Mi
            cpu: 100m
          limits:
            cpu: 100m
      terminationGracePeriodSeconds: 0
      volumes:
        - containerDisk:
            image: quay.io/kubevirt/fedora-container-disk-images:35
          name: containerdisk
        - cloudInitNoCloud:
            userData: |-
              #cloud-config
              Password: "password"
              chpasswd: { expire: False }
              runcmd:
                - sed -i -e "s/PasswordAuthentication.*/PasswordAuthentication yes/" /etc/ssh/sshd_config
                - systemctl restart sshd
          name: cloudinitdisk
status: {}
```

Windows

Voici comment récupérer le modèle utilisé :

```
oc get templates -n openshift windows10-desktop-medium -o yaml
```

Le voici en entier avec nos changements :

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm:
  name:
spec:
  running: false
  template:
    metadata:
      labels:
```

```
      kubevirt.io/vm:
  spec:
    terminationGracePeriodSeconds: 0
    evictionStrategy: LiveMigrate
    domain:
      clock:
        timer:
          hpet:
            present: false
          hyperv: {}
          pit:
            tickPolicy: delay
          rtc:
            tickPolicy: catchup
        utc: {}
      cpu:
        cores: 1
        model: host-passthrough
        sockets: 1
        threads: 1
      devices:
        blockMultiQueue: false
        disks:
        - disk:
            bus: virtio
          name:
        interfaces:
        - bridge: {}
          model: virtio
          name: nic-0
      features:
        acpi: {}
        apic: {}
        hyperv:
          frequencies: {}
          ipi: {}
          reenlightenment: {}
          relaxed: {}
          reset: {}
          runtime: {}
          spinlocks:
            spinlocks: 8191
          synic: {}
          synictimer:
            direct: {}
          vapic: {}
          vpindex: {}
      machine:
        type: pc-q35-rhel8.4.0
      resources:
        requests:
          cpu: "1"
          memory: 4G
        limits:
    networks:
    - multus:
        networkName: linux-bridge
      name: nic-0
    volumes:
    - dataVolume:
        name:
      name:
  dataVolumeTemplates:
  - metadata:
```

```
    annotations:
    name:
  spec:
    pvc:
      accessModes:
      - ReadWriteMany
      resources:
        requests:
          storage: 40Gi
      volumeMode: Block
      storageClassName: ocs-external-storagecluster-ceph-rbd
    source:
      pvc:
        namespace: "default"
        name: "win10-dv"
```

Pod

```
kind: Pod
apiVersion: v1
metadata:
  name: vdpod-pod-name
  namespace: pods-space
  labels:
    name: vdpod-density
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  restartPolicy: "Always"
  containers:
  - name: vdpod-pod-name
    image: gcr.io/google_containers/pause-amd64:3.0
    ports:
    imagePullPolicy: IfNotPresent
    securityContext:
      privileged: false
```

Déploiement de machines virtuelles

Le déploiement de machines virtuelles constitue la base de tout environnement virtuel. À grande échelle, la vitesse à laquelle nous pouvons déployer directement de nombreuses machines virtuelles affecte l'efficacité de la production.

Dans cette section, nous allons présenter les performances potentielles offertes par la méthode CSI (Container Storage Interface) de Ceph pour le clonage de plusieurs images de machines virtuelles à partir d'une source d'image maître.

Attention, si vous clonez plus de 100 machines virtuelles avec la stratégie de clonage CSI de Ceph, celle-ci risque de ne pas purger les clones et leur suppression manuelle peut également échouer (BZ#2055595).
Dans ce cas, il est plus sûr d'éviter le clonage d'instantané et d'utiliser la commande `cloneStrategy: copy` à la place.

Pour activer le clonage d'instantané CSI, vous devez modifier le profil de stockage OpenShift Virtualization :

```
oc edit -n openshift-cnv storageprofile <storage class name>
```

Vous devez aussi ajouter cette ligne :

```
spec:
  cloneStrategy: csi-clone
```

Commençons par importer une image QCOW RHEL dans un volume de données depuis l'un de nos hôtes :

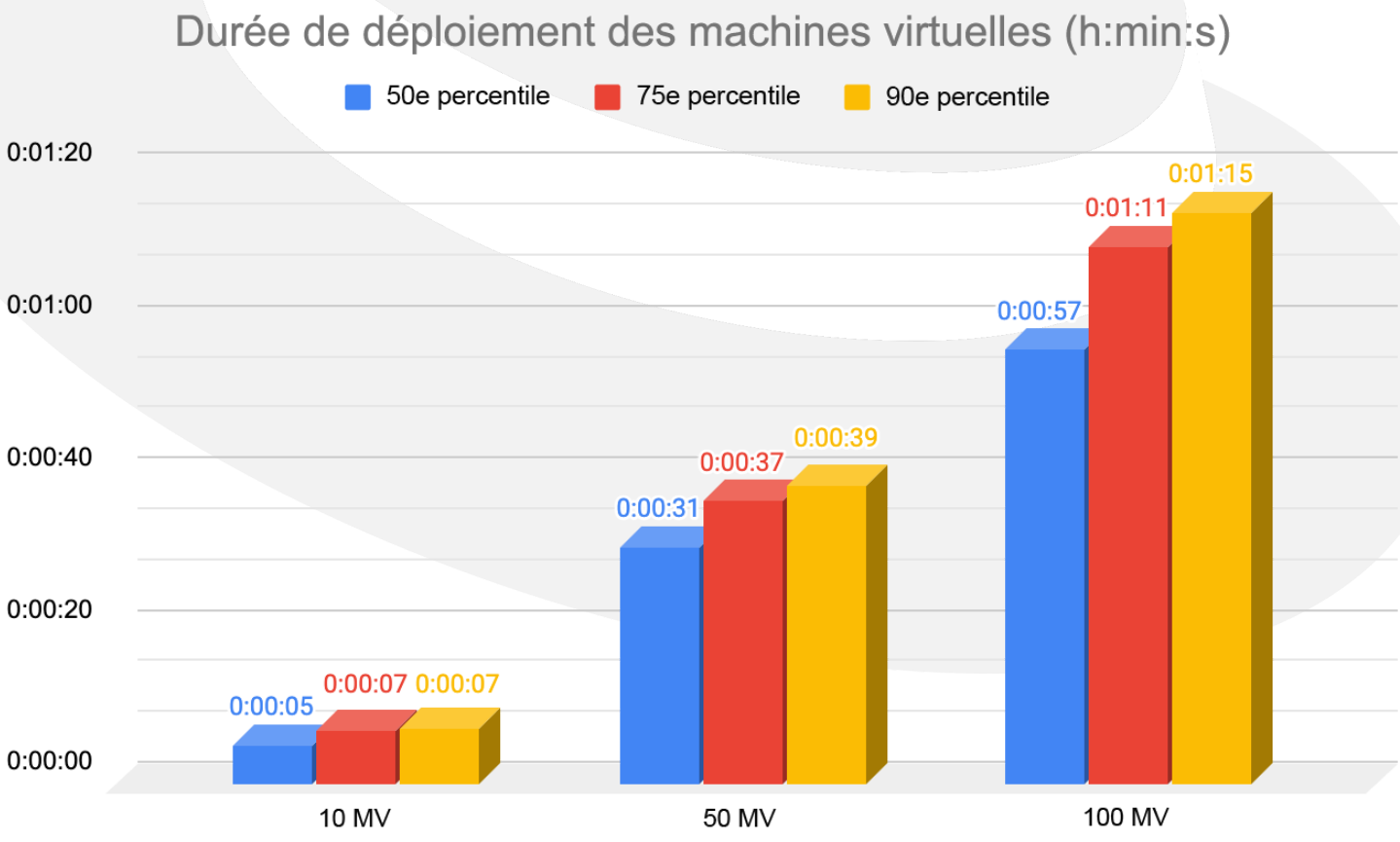
```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: rhel-clone-dv
spec:
  source:
      http:
         url: http://internel.server.com/ISO/rhel8.qcow2
  pvc:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 40Gi
    volumeMode: Block
    storageClassName: ocs-external-storagecluster-ceph-rbd
```

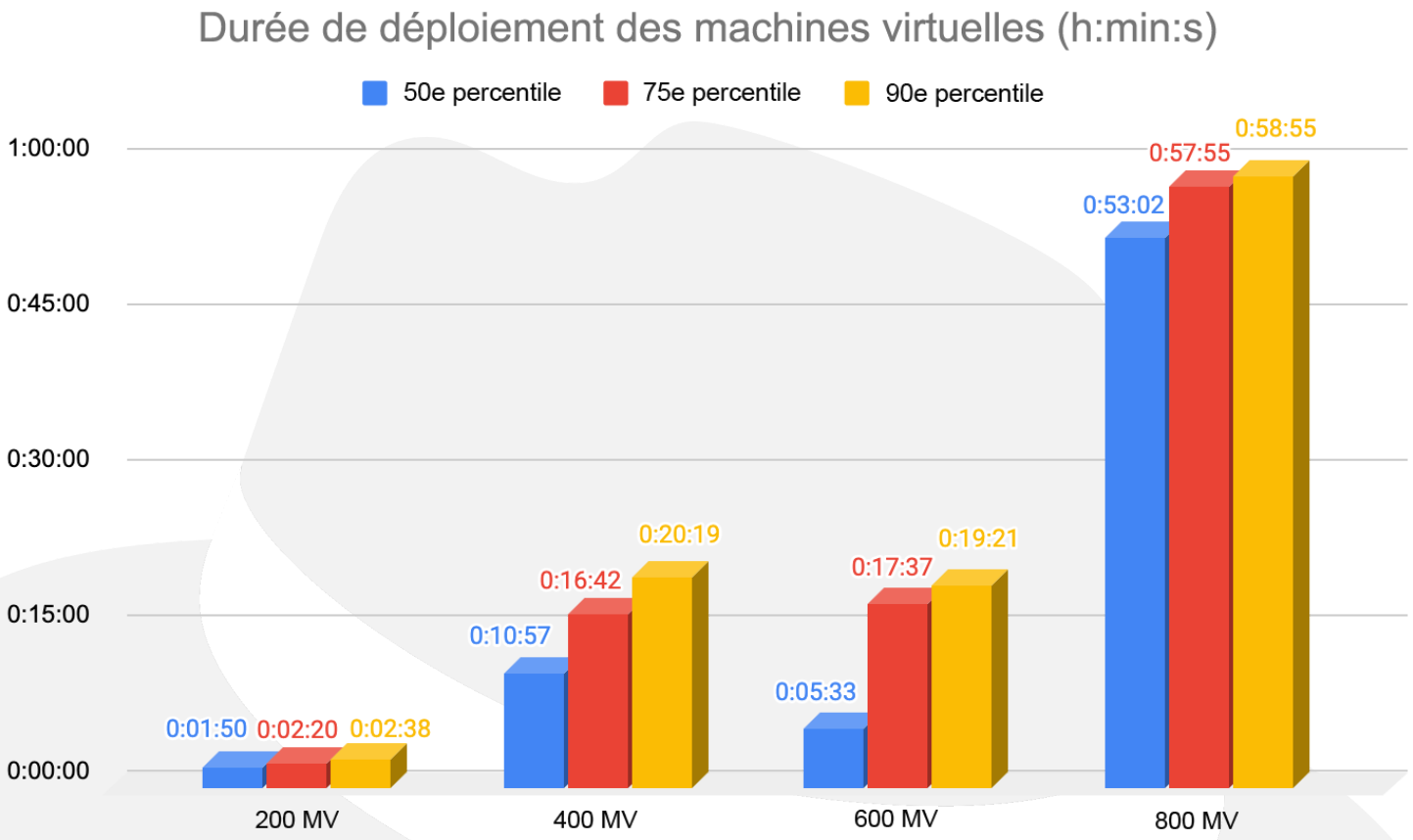
À la fin de l'importation, nous déployons en parallèle le nombre souhaité de machines virtuelles et mesurons la durée de clonage pour chacune d'entre elles. Pour ce faire, nous interrogeons chaque machine virtuelle toutes les 2 secondes jusqu'à la fin du clonage.

Les machines virtuelles se déploient de manière optimale par groupes de 100. Il faut donc déployer 100 machines virtuelles, patienter pendant toute la durée du clonage, puis en déployer 100 autres.

De manière générale, au-delà de 10 machines virtuelles déployées simultanément, le système CSI de Ceph applique une pénalité, car le paramètre vol-id de l'image parente se verrouille. Le clonage à partir d'une même image parente n'a donc pas lieu en parallèle, mais en série, et le provisionneur externe ne peut jamais envoyer plus de 10 appels gRPC parallèles au pilote CSI.

En outre, passé les 250 clones, les images RDB commencent à s'aplatir, ce qui accentue cette pénalité. Plus un instantané est volumineux, plus il faut du temps avant d'aplatir un clone.





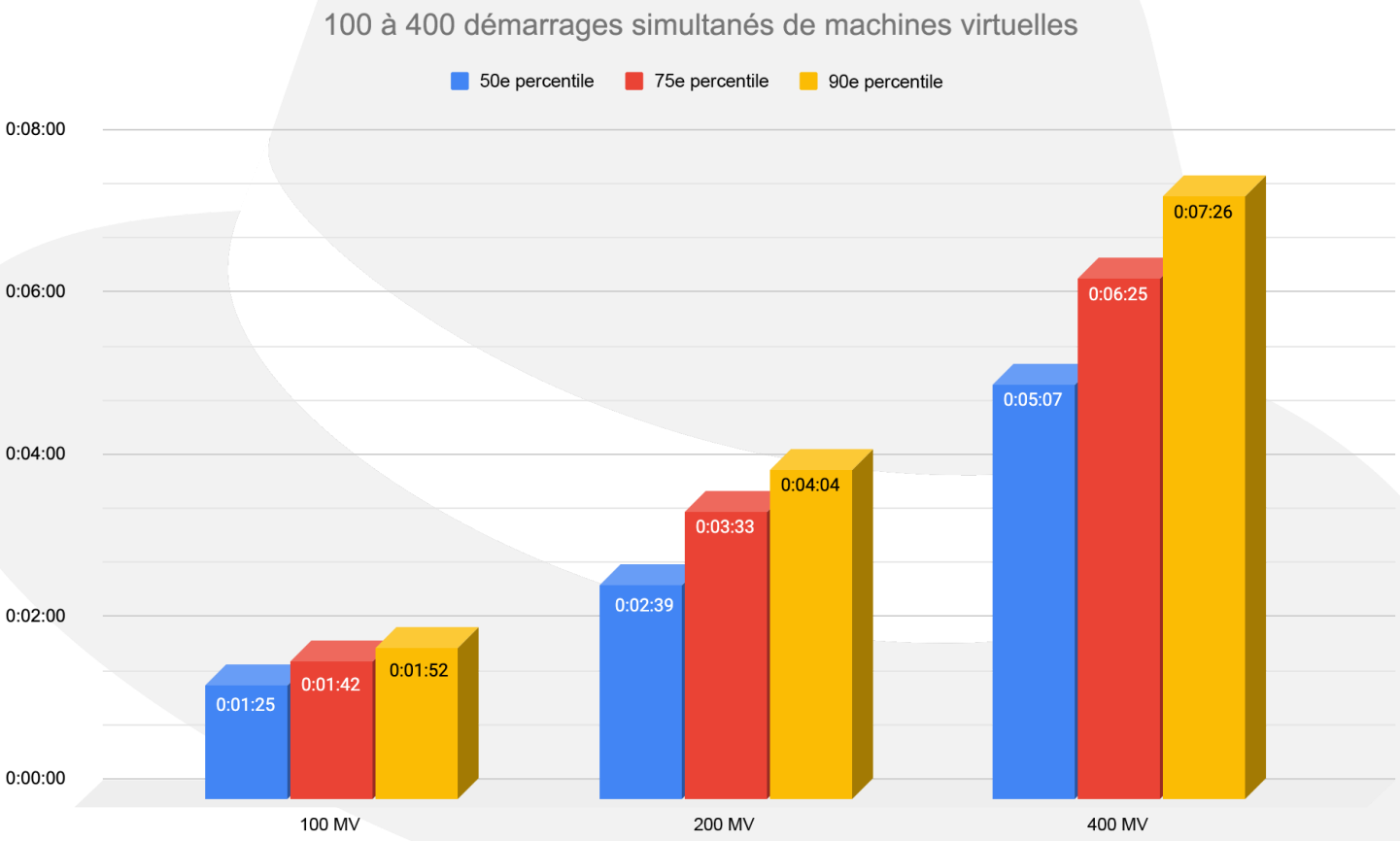
« Boot storm » de machines virtuelles

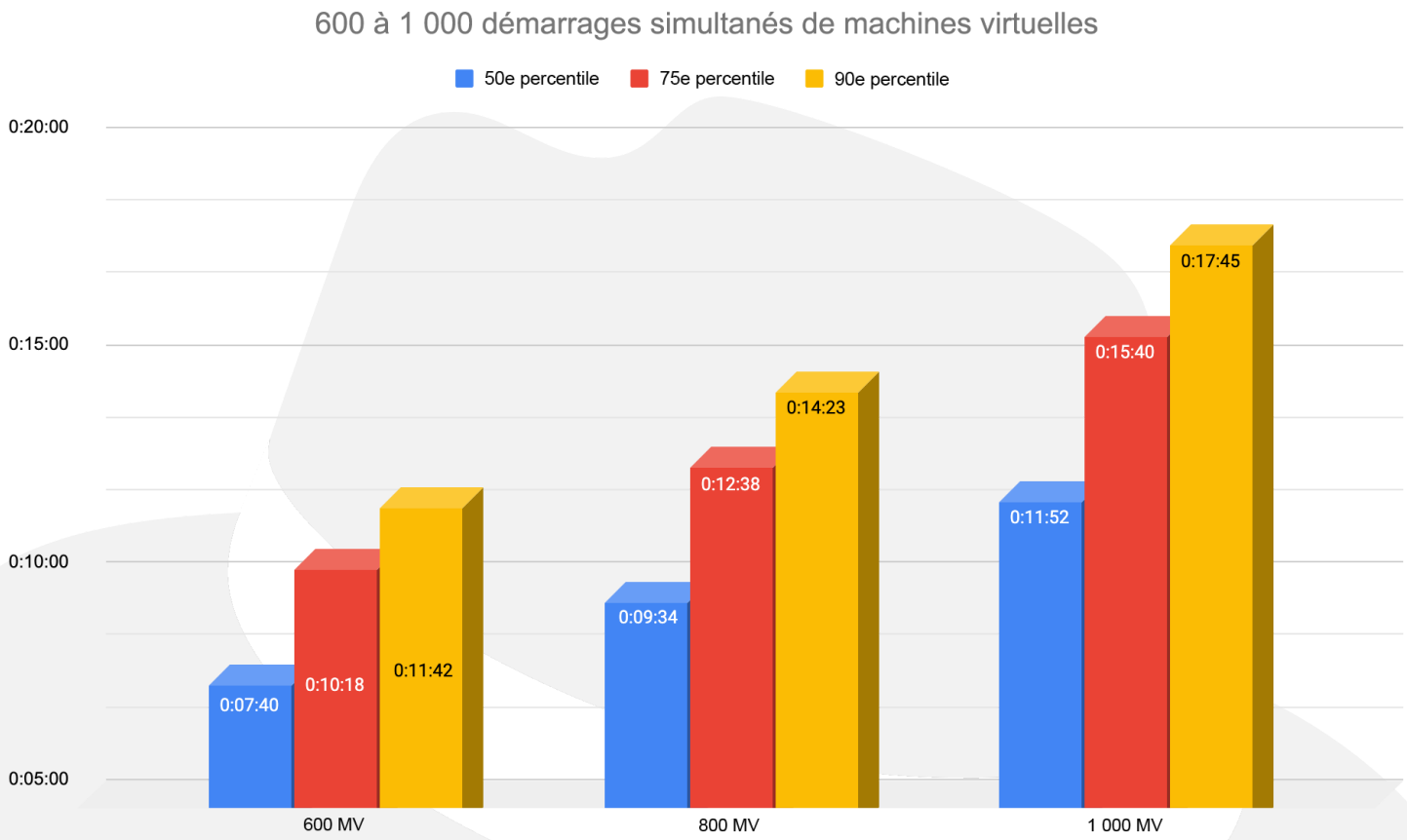
Dans ce scénario, nous avons testé le temps nécessaire au démarrage simultané d'un grand nombre de machines virtuelles afin de prouver la résilience d'OpenShift Virtualization et du plan de contrôle. Ce scénario intervient généralement lors de la récupération d'un environnement après un sinistre, par exemple une panne de courant.

La mesure commence au début de la demande et couvre chacune des machines virtuelles, jusqu'à ce qu'elles s'exécutent et soient accessibles via SSH (en d'autres termes, l'hôte a démarré correctement et le démon SSH est actif). Nous avons mesuré les durées en interrogeant chaque machine virtuelle. Lorsque son état bascule sur « Running », le test tente d'accéder à la machine virtuelle par SSH toutes les 2 secondes jusqu'à l'établissement de la connexion SSH.

Comme pour le scénario du déploiement, toutes les demandes de démarrage des machines virtuelles s'exécutent en même temps pour mettre sous pression toutes les variables du cluster.

Les graphiques ci-dessous montrent qu'avec notre cluster OCP homogène, les temps de démarrage sont presque linéaires jusqu'à 1 000 machines virtuelles. Au-delà, les durées s'allongent.




600 à 1 000 démarrages simultanés de machines virtuelles
50e percentile
75e percentile
90e percentile
0:20:00
0:15:00
0:10:00
0:05:00
0:07:40
0:10:18
0:11:42
0:09:34
0:12:38
0:14:23
0:11:52
0:15:40
0:17:45
600 MV
800 MV
1 000 MV

Latence des machines virtuelles

Chaque machine virtuelle dispose de son propre thread pour traiter les entrées/sorties, sauf s'il existe plusieurs revendications de volume persistant et si la valeur du paramètre `dedicatedIOThread:` est « true ». Dans ce cas, chaque revendication de volume persistant dispose de son propre thread d'E/S.

Dans le scénario suivant, nous avons utilisé 15 machines virtuelles par nœud de calcul et testé jusqu'à 64 nœuds de calcul, soit 960 machines virtuelles, pour démontrer que même en la présence de plusieurs threads qui accèdent en même temps au cluster RHCS, à eux seuls ils n'aggravent pas la latence.

Pour les besoins de ce scénario, nous avons utilisé des blocs de 4 Ko pour les lectures et les écritures aléatoires. Voici les tests que nous avons réalisés :

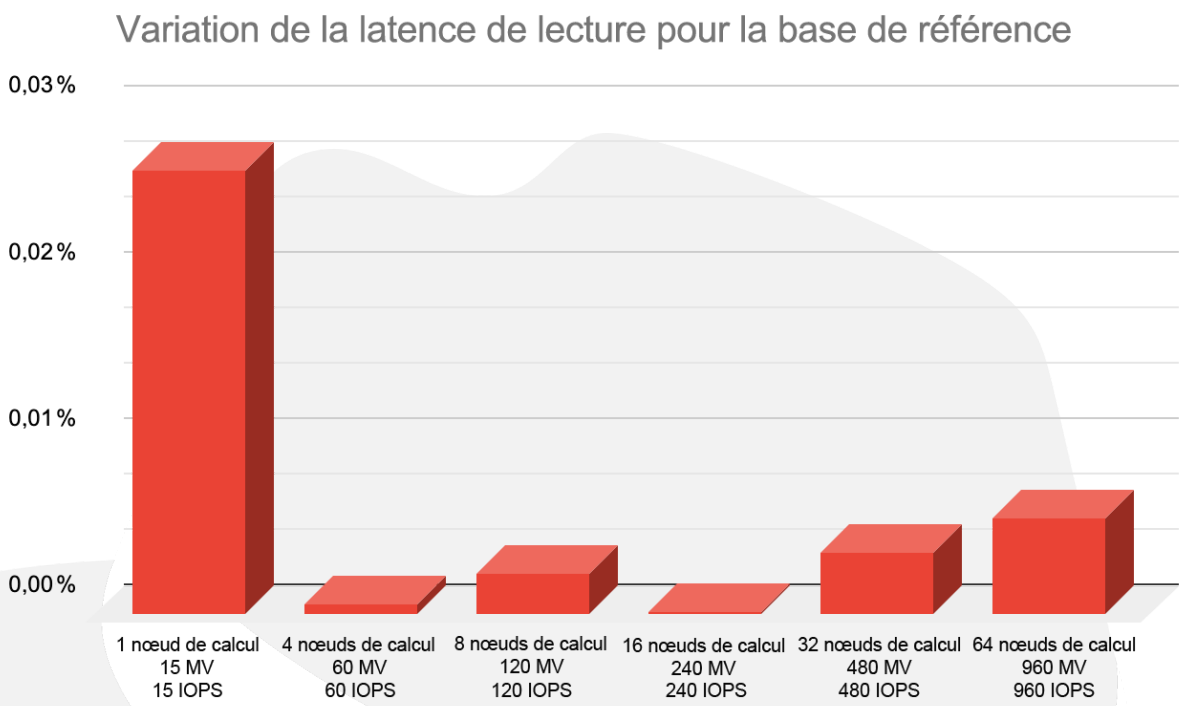
- Base de référence : nous avons utilisé 15 machines virtuelles par nœud de calcul qui généraient chacune une IOPS, en commençant à 15 machines virtuelles (15 IOPS, un nœud) et en terminant à 64 nœuds, pour un total de 960 machines virtuelles (960 IOPS).
- Charge de travail : nous avons utilisé exactement 15 machines virtuelles par nœud de calcul qui généraient chacune 1 000 IOPS, en commençant à 15 machines virtuelles (15 000 IOPS, un nœud) et en terminant à 64 nœuds, pour un total de 960 machines virtuelles (960 000 IOPS).

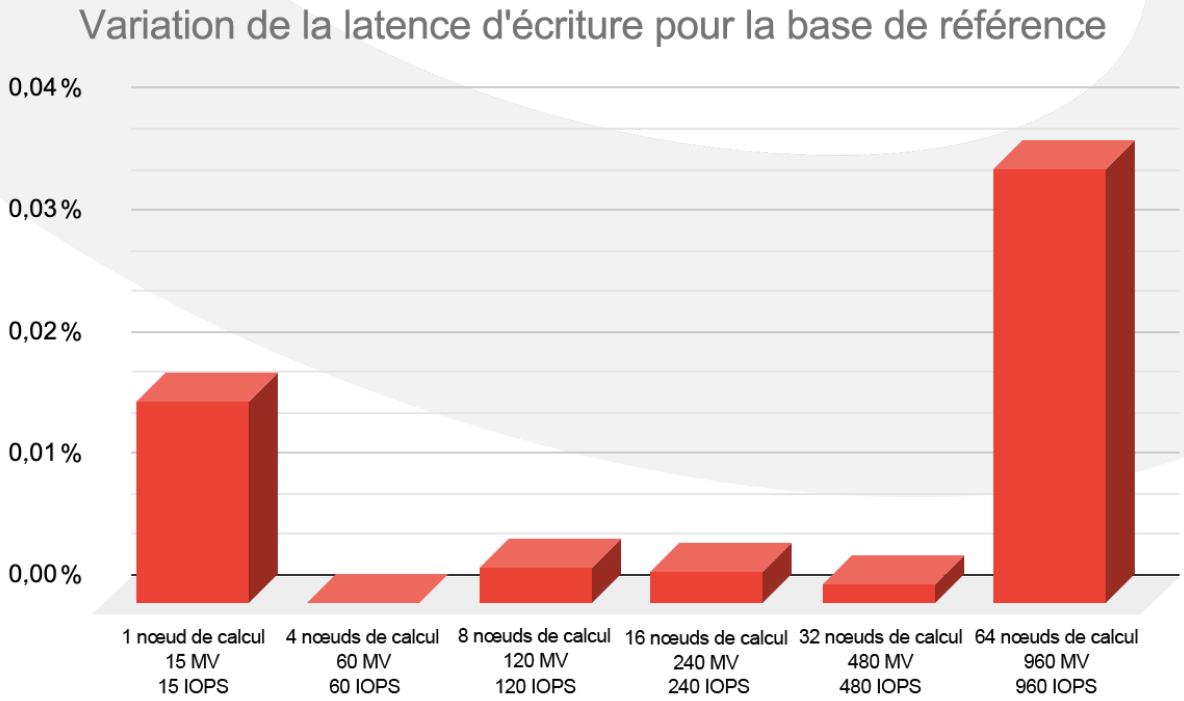
L'ensemble de données du système de fichiers de chaque machine virtuelle comprend 300 répertoires. Chaque répertoire contient 8 fichiers et chaque fichier fait 20 Mio. Autrement dit, toutes les machines virtuelles ont un ensemble de données de 4,8 Gio.

Nous avons choisi un petit bloc afin d'éviter autant que possible les incohérences dues à la mise en réseau et aux disques hétérogènes de Ceph dans le cluster RHCS.

À noter que bien que nous ayons utilisé jusqu'à 960 machines virtuelles lors de nos tests, il y avait au total 3 000 machines virtuelles qui s'exécutaient sur le cluster, ainsi que 21 400 pods.

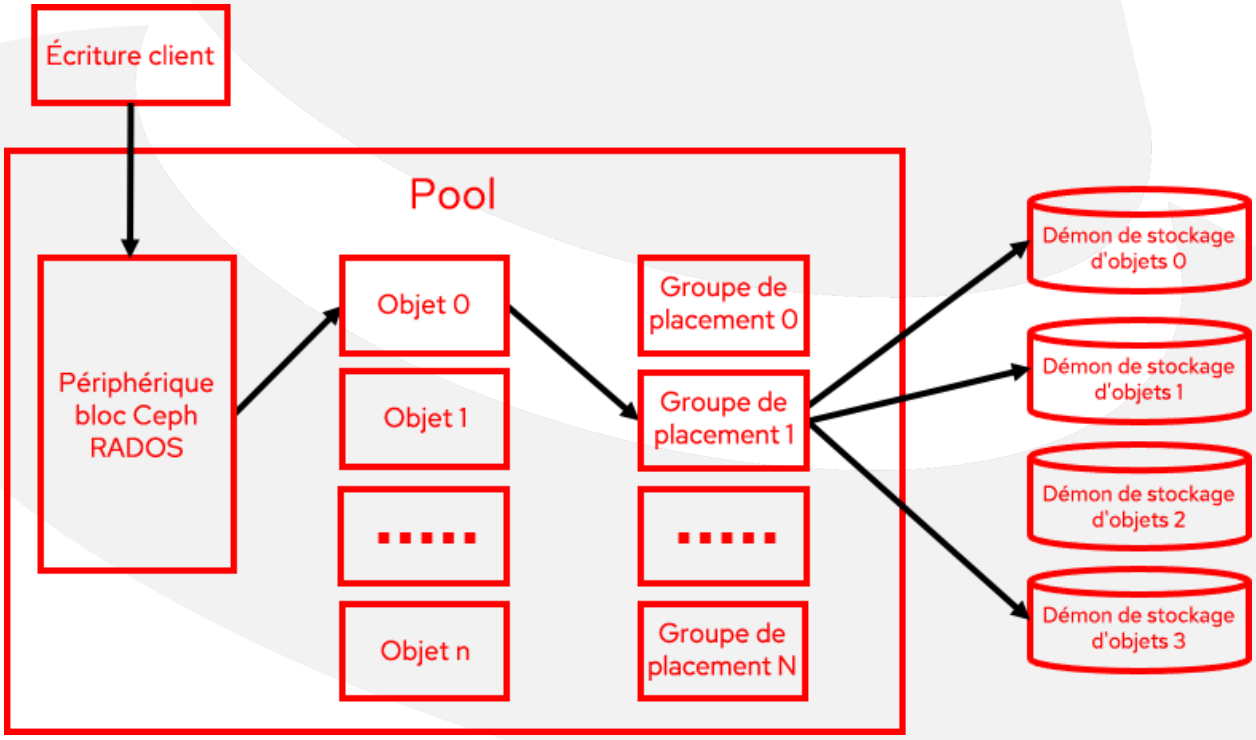
Comme le montrent les deux graphiques ci-dessous, la latence des lectures et des écritures aléatoires présentait une variation inférieure à 0,04 % durant l'exécution des tests de base.


Variation de la latence de lecture pour la base de référence
0,03 %
0,02 %
0,01 %
0,00 %
1 nœud de calcul
15 MV
15 IOPS
4 nœuds de calcul
60 MV
60 IOPS
8 nœuds de calcul
120 MV
120 IOPS
16 nœuds de calcul
240 MV
240 IOPS
32 nœuds de calcul
480 MV
480 IOPS
64 nœuds de calcul
960 MV
960 IOPS

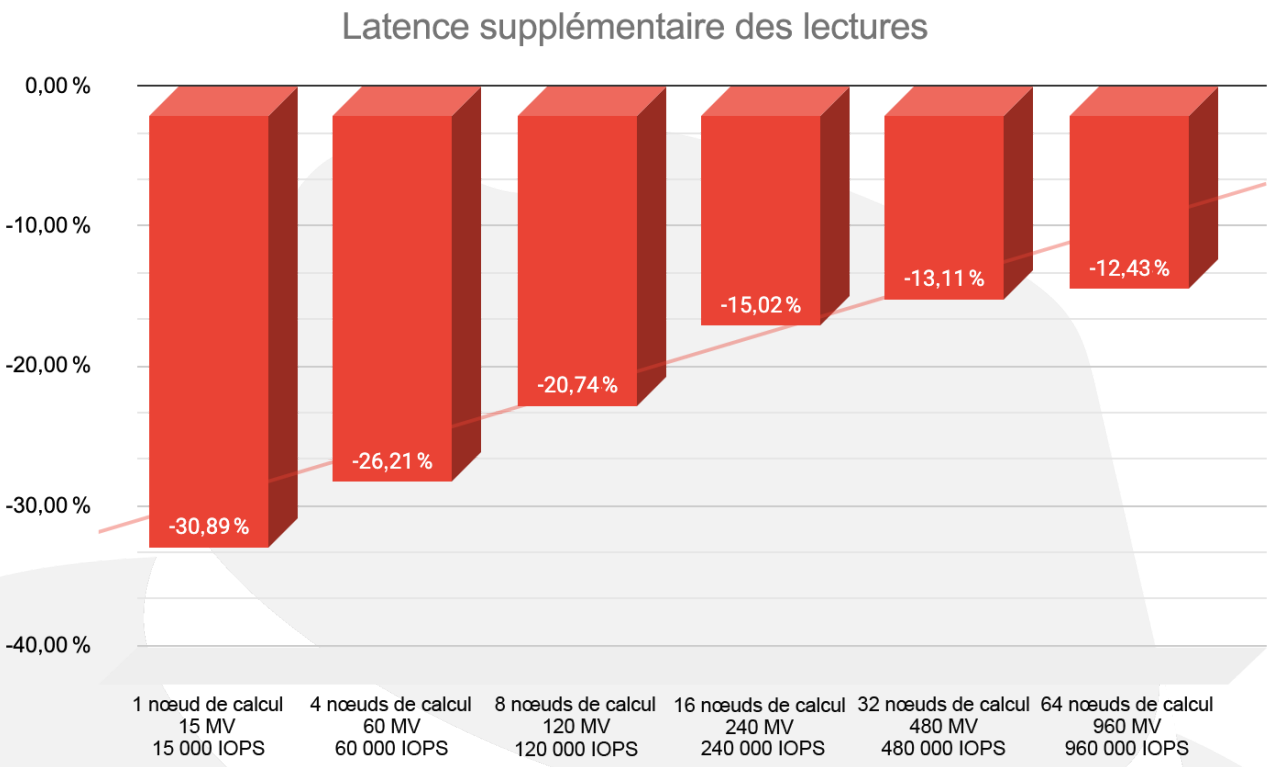

Variation de la latence d'écriture pour la base de référence
0,04 %
0,03 %
0,02 %
0,01 %
0,00 %
1 nœud de calcul
15 MV
15 IOPS
4 nœuds de calcul
60 MV
60 IOPS
8 nœuds de calcul
120 MV
120 IOPS
16 nœuds de calcul
240 MV
240 IOPS
32 nœuds de calcul
480 MV
480 IOPS
64 nœuds de calcul
960 MV
960 IOPS

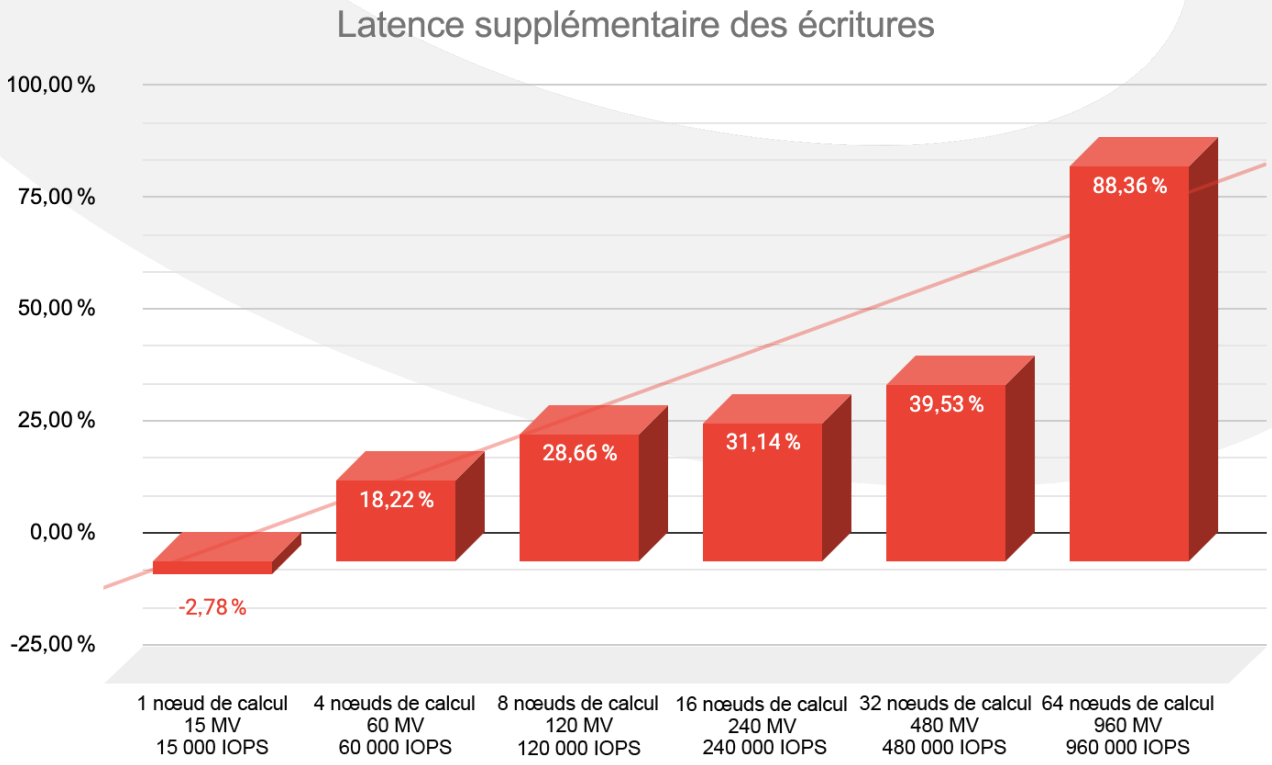
Le diagramme ci-dessous montre la tendance de la latence lors du scénario axé sur les charges de travail, par rapport aux résultats de base (plus la valeur est basse, mieux c'est). Pour les performances de lecture, un taux d'IOPS élevé fait diminuer la latence dans une certaine mesure, en raison de la différence d'allocation des ressources entre les pics d'utilisation et la charge de travail inactive ou faible des IOPS sur la machine virtuelle.

Le flux de données pour les écritures est différent, ce qui est nécessaire pour garantir une haute disponibilité. Comme le montre le diagramme, pour chaque écriture générée par Ceph, trois copies des données parcourent le réseau vers leurs périphériques de stockage d'objets respectifs. Ce phénomène influe sur la latence des écritures.



Remarque : pour les applications sensibles à la latence, chaque réplique de données réduite peut diminuer le surplus de latence des écritures d'un tiers.


Latence supplémentaire des lectures
0,00 %
-10,00 %
-20,00 %
-30,00 %
-40,00 %
-30,89 %
-26,21 %
-20,74 %
-15,02 %
-13,11 %
-12,43 %
1 nœud de calcul
15 MV
15 000 IOPS
4 nœuds de calcul
60 MV
60 000 IOPS
8 nœuds de calcul
120 MV
120 000 IOPS
16 nœuds de calcul
240 MV
240 000 IOPS
32 nœuds de calcul
480 MV
480 000 IOPS
64 nœuds de calcul
960 MV
960 000 IOPS


Latence supplémentaire des écritures
100,00 %
75,00 %
50,00 %
25,00 %
0,00 %
-25,00 %
-2,78 %
18,22 %
28,66 %
31,14 %
39,53 %
88,36 %
1 nœud de calcul
15 MV
15 000 IOPS
4 nœuds de calcul
60 MV
60 000 IOPS
8 nœuds de calcul
120 MV
120 000 IOPS
16 nœuds de calcul
240 MV
240 000 IOPS
32 nœuds de calcul
480 MV
480 000 IOPS
64 nœuds de calcul
960 MV
960 000 IOPS

Migration des machines virtuelles

Dans le scénario suivant, nous avons testé la migration de 1 000 machines virtuelles. À des fins de réalisme, nous ne nous sommes pas contentés de migrer les machines virtuelles : nous avons également redémarré les nœuds de calcul sur lesquels elles résident. Pour ce faire, nous avons divisé les nœuds en trois zones, puis appliqué une configuration de machine vide à une zone spécifique, ce qui a déclenché le redémarrage de tous les nœuds associés à cette zone après l'éviction de toutes les machines virtuelles qui résidaient sur les nœuds de calcul.

Pour commencer, nous avons étiqueté chaque nœud de calcul dans une zone spécifique :

```
oc label node worker01 node-role.kubernetes.io/zone-0=""
oc label node worker02 node-role.kubernetes.io/zone-1=""
oc label node worker03 node-role.kubernetes.io/zone-2=""
```

Ensuite, nous avons créé un pool de configurations de machines pour chaque zone. La valeur `maxUnavailable: 10` définit le nombre de nœuds autorisés à cesser de fonctionner à tout moment du cycle de vie de la zone :

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: zone-0
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values:
[worker, zone-2]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/zone-0: ""
  paused: false
  maxUnavailable: 10
```

Nous avons également modifié l'opérateur hyperconverged-cluster :

```
oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

De plus, nous avons défini les paramètres de migration suivants pour augmenter le nombre de migrations simultanées :

```
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    parallelMigrationsPerCluster: 20 # default 5
    parallelOutboundMigrationsPerNode: 4 # default 2
    progressTimeout: 150
```

Nos tests ont révélé qu'il est fortement recommandé de porter le nombre de pods virt-api à un ratio de **1 pod kubevirt-api par groupe de 750 machines virtuelles**. Puisque notre configuration exécutait 3 000 machines virtuelles, nous avons mis à l'échelle le nombre de pods kubevirt-api en l'augmentant à 4.

Une fonction de mise à l'échelle automatique est déjà à l'étude pour ce scénario. Les progrès sont accessibles sur Github#7101. Actuellement, ce processus peut être effectué manuellement en corrigeant l'opérateur hyperconverged :

```
oc patch hco -n openshift-cnv kubevirt-hyperconverged --type=merge -p
'{"metadata":{"annotations":{"kubevirt.kubevirt.io/jsonpatch":"[{\"op\":
\"add\", \"path\": \"/spec/customizeComponents/patches\", \"value\":
[{\"resourceType\": \"Deployment\", \"resourceName\": \"virt-api\",
\"type\": \"json\", \"patch\": \"[{\\\"op\\\": \\\"replace\\\",
\\\"path\\\": \\\"/spec/replicas\\\", \\\"value\\\": 4}]\"}]}]"}}}'
```

Nous avons ensuite lancé la migration en créant cette configuration de machine :

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: zone_target #target zone name
  name: job_name #must be unique every time.
spec:
  config:
    ignition:
```

```
    config: {}
    security:
      tls: {}
    timeouts: {}
    version: 3.1.0
  networkd: {}
  passwd: {}
  storage:
    files:
    - contents:
        source: data:text/plain;charset=utf-8;base64,Zm9vCg==
        verification: {}
      filesystem: root
      mode: 420
      path: /var/tmp/tmp_dir
osImageURL: ""
```

Répartition des 1 000 machines virtuelles migrées :

- 400 machines virtuelles RHEL
- 400 machines virtuelles Fedora
- 200 machines virtuelles Windows

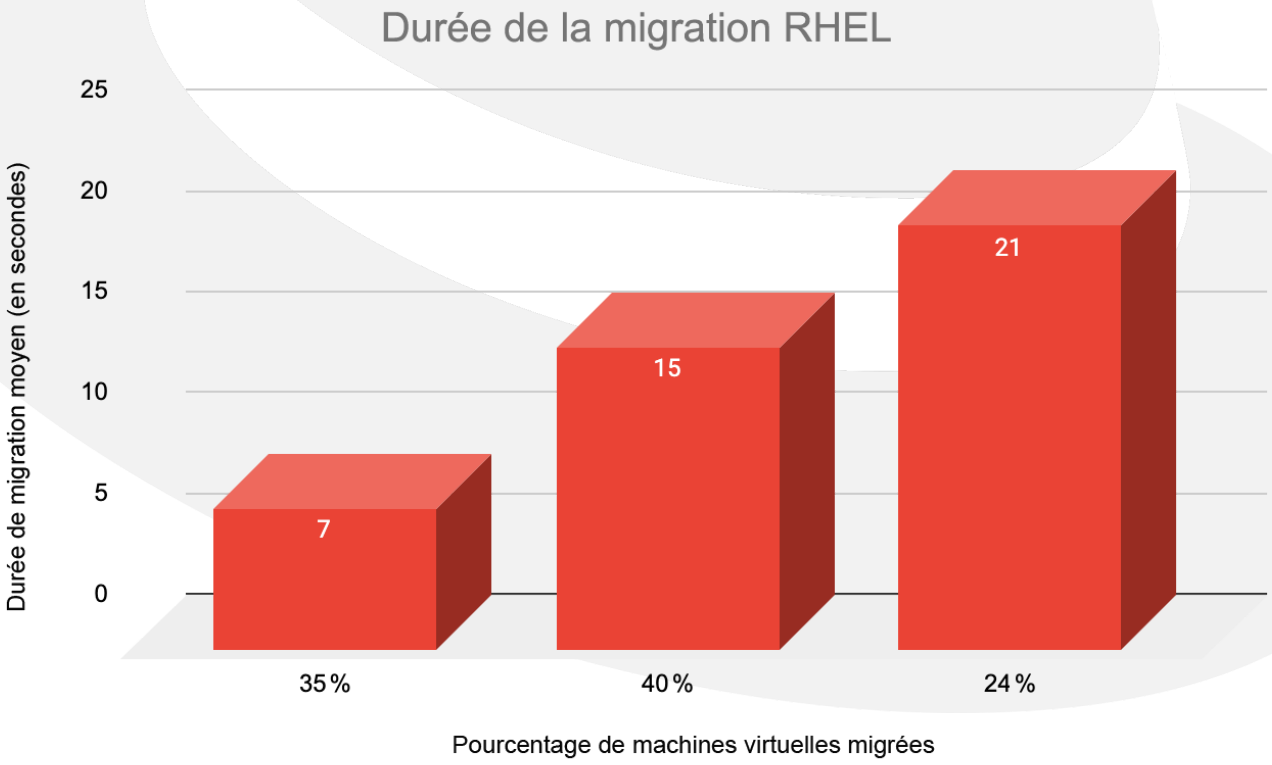
En outre, les 7 000 pods en colocation avec les machines virtuelles ont été redémarrés sur d'autres nœuds de calcul.

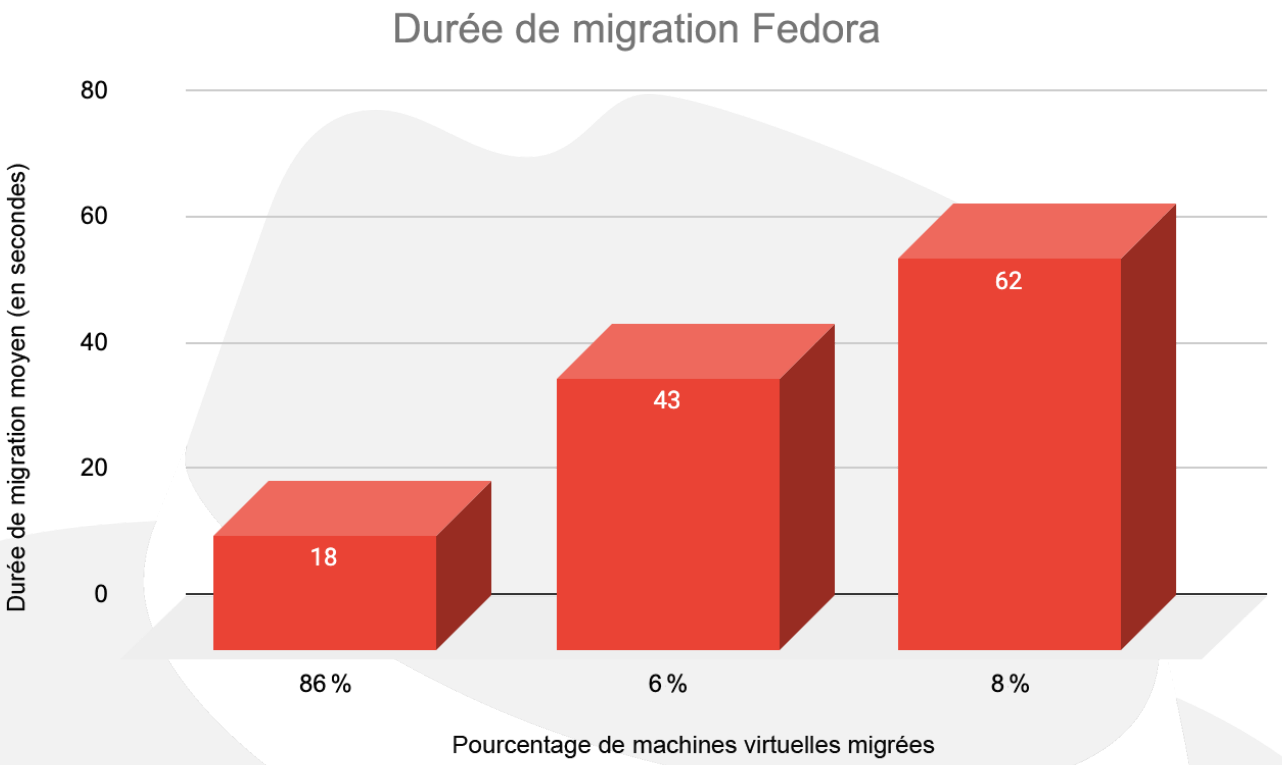
Lors de la migration, les journaux d'instance de machine virtuelle (VMI) indiquent le temps nécessaire à la migration complète de chaque machine virtuelle :

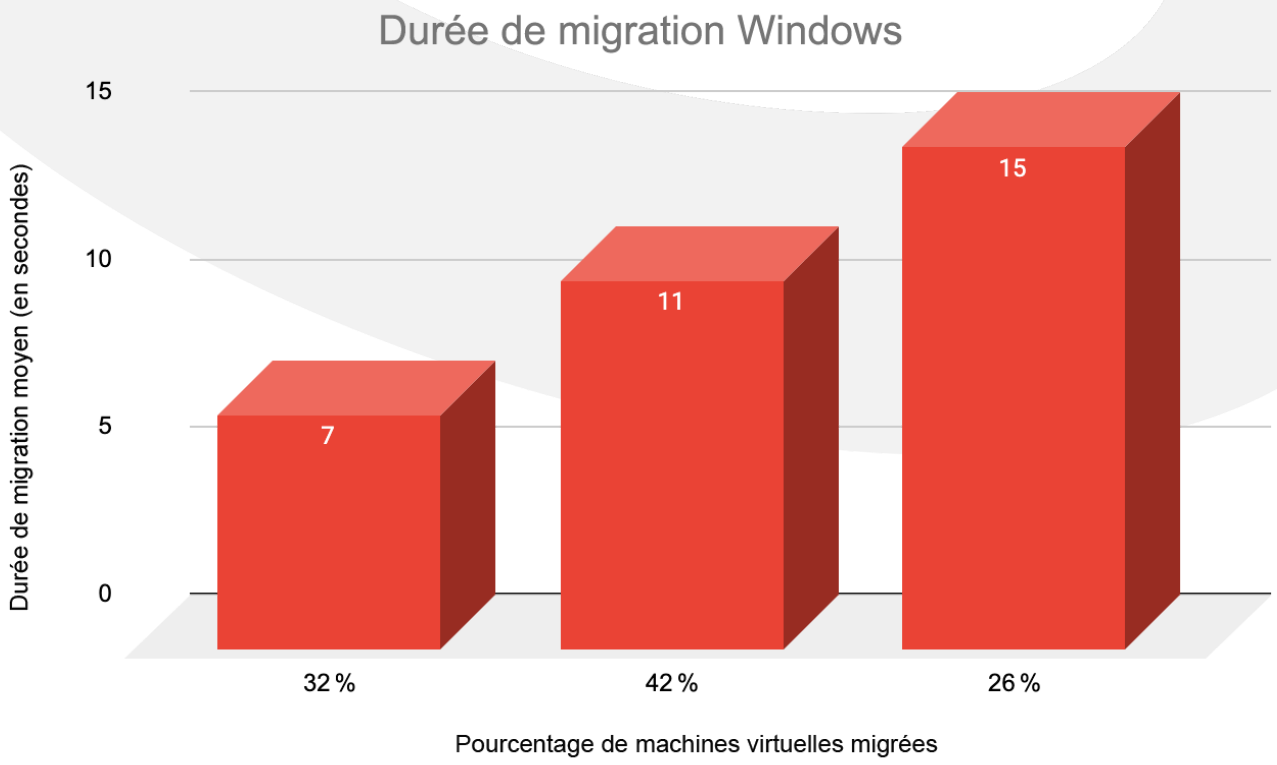
```
Phase Transition Timestamps:
  Phase:                        Scheduling
  Phase Transition Timestamp:   2022-04-10T07:15:13Z
  Phase:                        Scheduled
  Phase Transition Timestamp:   2022-04-10T07:15:23Z
  Phase:                        Running
  Phase Transition Timestamp:   2022-04-10T07:15:25Z
```

Les graphiques ci-dessous montrent les durées de migration de chaque système d'exploitation, distribués par pourcentage. Par exemple, la migration des machines virtuelles RHEL a duré 7 secondes en moyenne pour 35 % d'entre elles.

Remarque : la durée de migration n'a pas forcément de lien avec le système d'exploitation, mais elle en a un avec la charge de l'invité à ce moment, celle de l'hôte et celle du réseau, la technologie de stockage, la politique de migration, la taille des images, etc. Les résultats peuvent donc varier.




Durée de migration Fedora
Durée de migration moyen (en secondes)
80
60
40
20
0
18
43
62
86 %
6 %
8 %
Pourcentage de machines virtuelles migrées


Durée de migration Windows
Durée de migration moyen (en secondes)
15
10
5
0
7
11
15
32 %
42 %
26 %
Pourcentage de machines virtuelles migrées

Durée de migration moyen par système d'exploitation :

Système d'exploitation	Durée de migration moyen (en secondes)	Remarques
RHEL	14	Revendication de volume persistant de 40 Gio
Fedora	23	Disque de conteneur evictionStrategy: Restart
Windows	12	Revendication de volume persistant de 40 Gio

En résumé, la migration totale a pris 118 minutes, en plus des 35 minutes qu'a duré le passage des nœuds à l'état « Ready » en raison du paramètre `maxUnavailable: 10`, mentionné plus haut.

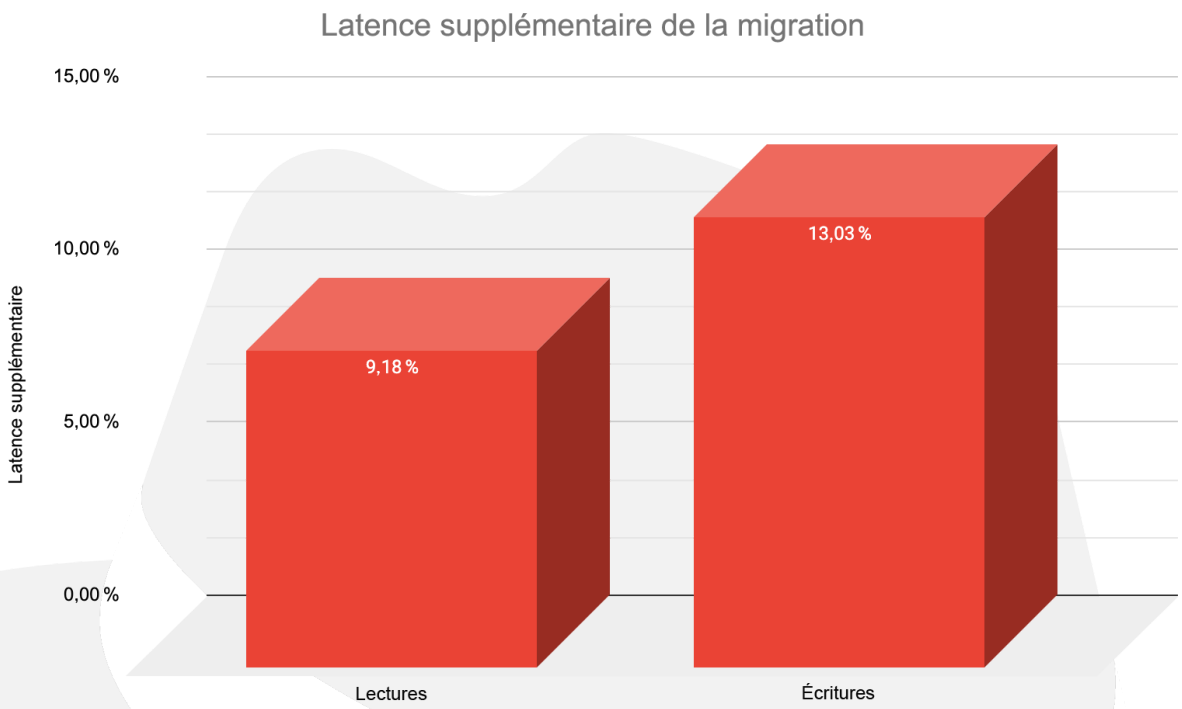
Augmentation de la latence due à la migration des machines virtuelles

Dans le scénario suivant, nous avons testé la migration de 1 000 machines virtuelles, mais en n'utilisant cette fois que des machines virtuelles RHEL. Par ce choix, nous voulions éviter les écarts dus aux différents traitements que réservent les systèmes d'exploitation aux E/S.

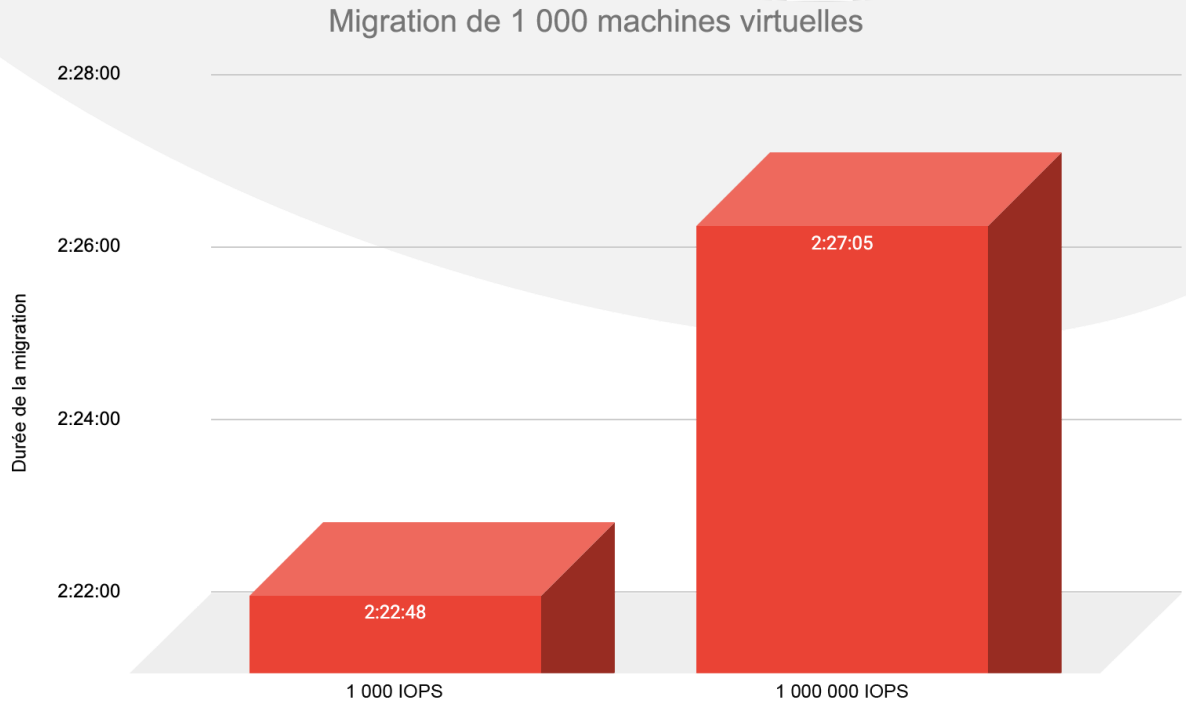
Comme précédemment, nous avons utilisé des blocs de 4 Ko pour les lectures et les écritures aléatoires. Voici les tests que nous avons réalisés :

- Base de référence : chacune des 1 000 machines virtuelles génère une seule IOPS, pour un total de 1 000 IOPS.
- Charge de travail : chacune des 1 000 machines virtuelles génère 1 000 IOPS, pour un total de 1 million d'IOPS.

Le graphique ci-dessous montre la pénalité de latence moyenne au cours de la migration pour les lectures et les écritures, par rapport à la base de référence (uniquement des machines virtuelles RHEL) :



De plus, comme le montre cet autre graphique, la durée de la migration a augmenté de 5 %. Notez que le bogue BZ#2069098 peut faire varier la durée de la migration :



Red Hat

Mise à niveau des clusters à grande échelle

Dans le scénario suivant, nous avons testé des mises à niveau mineures et majeures.

Pour commencer, nous avons mis à niveau le cluster de la version 4.9.15 à 4.9.23, tout en simulant une véritable mise à niveau de la production. À cette fin, 3 000 machines virtuelles et 21 400 pods s'exécutaient sur le cluster. Par ailleurs, une charge de travail légère de 4 Ko a été générée sur 1 500 machines virtuelles, à une fréquence de 100 IOPS par machine virtuelle.

Nous avons lancé la mise à niveau en exécutant la commande suivante :

```
$ oc adm upgrade --to 4.9.23
```

Pour suivre l'avancée du processus, nous avons saisi :

```
$ oc get clusterversion
```

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.9.15    True        True          25m     Working towards
4.9.23: 569 of 738 done (77% complete)
```

Au total, le processus de mise à niveau mineure a pris **35 minutes**.

La prochaine étape consistait à tester une mise à niveau majeure en passant de la version 4.9.23 du cluster à la version 4.10.9, toujours dans les conditions de mise à niveau précédentes. Nous avons lancé la mise à niveau en exécutant la commande suivante :

```
oc adm upgrade channel candidate-4.10 --allow-explicit-channel
oc adm upgrade --to 4.10.9 --allow-explicit-upgrade
```

Encore une fois, pour suivre l'avancée du processus, nous avons saisi :

```
NAME      VERSION   AVAILABLE   PROGRESSING   SINCE   STATUS
version   4.9.23    True        True          40m     Working towards
4.10.9: 95 of 771 done (12% complete)
```

Au total, le processus de mise à niveau majeure a pris **136 minutes**.

Remarque : pour certaines mises à niveau, des modifications peuvent nécessiter une réinitialisation partielle de tous les nœuds, ce qui ralentit la migration et allonge considérablement le délai de mise à niveau.

Conclusion

Dans cette architecture de référence, nous avons montré les capacités et la résilience d'OpenShift Virtualization à grande échelle. Associée à Red Hat Ceph Storage et/ou à Red Hat OpenShift Data Foundation, la fonction OpenShift Virtualization de Red Hat OpenShift Container Platform offre une solution de production complète qui intègre des conteneurs, des machines virtuelles et un stockage à haute disponibilité. Elle peut également être déployée sur tout type d'hôte qui respecte la configuration matérielle minimale requise.

Cette architecture de référence a été conçue pour atteindre l'objectif que vous avions fixé en amont, mais pour obtenir un niveau idéal de résilience, d'évolutivité et de simplicité d'exploitation quotidienne dans votre environnement, avec ses conditions et exigences particulières, il est conseillé d'envisager également d'autres architectures. Par exemple, au-delà de certaines limites, une approche à clusters multiples sera plus appropriée si le nombre de nœuds ou la charge de travail deviennent trop importants, ou si le cluster subit trop d'agitation.

Ressources supplémentaires

Configuration minimale requise pour le système et l'environnement :
https://docs.openshift.com/container-platform/3.11/install/prerequisites.html

Modèles OpenShift :
https://docs.openshift.com/container-platform/4.9/openshift_images/using-templates.html

Opérateur de configuration de machine :
https://docs.openshift.com/container-platform/4.9/post_installation_configuration/machine-configuration-tasks.html

Migration dynamique et délais d'attente :
https://docs.openshift.com/container-platform/4.9/virt/live_migration/virt-live-migration-limits.html

Mise à jour d'un cluster avec l'interface en ligne de commande :
https://docs.openshift.com/container-platform/4.10/updating/updating-cluster-cli.html

À propos de Red Hat

Premier éditeur mondial de solutions Open Source, Red Hat s'appuie sur une approche communautaire pour fournir des technologies Linux, de cloud hybride, de conteneurs et Kubernetes fiables et performantes. Red Hat aide ses clients à développer des applications cloud-native, à intégrer des applications nouvelles et existantes ainsi qu'à gérer et à automatiser des environnements complexes. Conseiller de confiance auprès des entreprises du Fortune 500, Red Hat propose des services d'assistance, de formation et de consulting reconnus qui apportent à tout secteur les avantages de l'innovation ouverte. Situé au cœur d'un réseau mondial d'entreprises, de partenaires et de communautés, Red Hat participe à la croissance et à la transformation des entreprises et les aide à se préparer à un avenir toujours plus numérique.