OpenShift Virtualization

Ajustes y rendimiento a gran escala

con el almacenamiento externo Red Hat Ceph Storage 5

Arquitectura de referencia

Boaz Ben Shabat

Índice

Público al que va dirigido

La finalidad de este documento es ayudar a las personas encargadas de los servicios de infraestructura, como los clientes, los ingenieros de ventas, los consultores de campo y los arquitectos de soluciones.
Se presenta un ejemplo de la implementación exitosa y a gran escala de la función OpenShift Virtualization, la cual forma parte de Red Hat OpenShift® Container Platform, con RHCS como la solución de almacenamiento de alta disponibilidad (HA) para la red externa.

Resumen ejecutivo

En el documento, se expone el conocimiento que adquirieron los equipos de rendimiento y adaptación de Red Hat OpenShift Virtualization luego de lograr la implementación exitosa a gran escala de un clúster externo de Red Hat® Ceph® Storage 5 (RHCS) (47 nodos) y Red Hat OpenShift Virtualization (100 nodos) con un clúster externo de Ceph que almacena 3000 máquinas virtuales (VM) junto con 21 400 pods.

En este documento sobre la arquitectura de referencia, detallaremos los pasos que seguimos para configurar RHCS y Red Hat OpenShift Virtualization y, así, crear un clúster resistente de OpenShift con 100 nodos.

También explicaremos el razonamiento detrás de cada paso y aportaremos la información necesaria para que se puedan aplicar las recomendaciones a cualquier clúster.

Consulte la tabla para conocer los resultados de rendimiento de los casos más importantes que se podrían presentar en el entorno de producción:

Caso	Descripción	Resultado
Implementaciones de VM	Implementación de hasta 800 VM en paralelo	Los resultados de las pruebas demuestran que se puede agilizar la implementación cuando se clonan las VM en grupos de 100.
Arranques simultáneos de las VM	Arranques simultáneos de hasta 1000 VM	Los tiempos de arranque casi lineales comenzaron al minuto 01:42 (MM:SS) para 100 VM y finalizaron al minuto 17:45 para 1000 VM.
Latencia de las VM	Latencia de inactividad constante en comparación con la latencia de las cargas de trabajo para las operaciones de lectura y de escritura	La cantidad de subprocesos de entrada y salida (I/O) que acceden a RHCS no influyen en la latencia de inactividad de las VM: 1 millón de operaciones de I/O por segundo (IOPS), reducción de hasta el 30 % de la latencia de lectura e incremento de hasta el 88 % de la latencia de escritura.
Migración de las VM	Migración de 1000 VM	La migración de 1000 VM y la expulsión de 7000 pods tomó alrededor de 118 minutos (HH:MM).
Aumento de la latencia en la migración de las VM	Migración de 1000 VM de Red Hat Enterprise Linux® (RHEL) con cargas de trabajo	La latencia de I/O durante la migración aumentó un 9 % para las operaciones de lectura y un 13 % para las de escritura. El tiempo que tomó la migración aumentó un 3 %, y la tasa de IOPS real no se vio afectada.
Actualización del clúster de OpenShift	Actualización de la versión del clúster de OpenShift	La actualización de la versión secundaria tomó 35 minutos, mientras que la de la versión principal llevó 136 minutos.

Elementos de software

Producto	Versión	Descripción
Red Hat OpenShift	4.9.15	Es la plataforma principal de Kubernetes para las empresas que permite diseñar, implementar y ejecutar las aplicaciones de manera uniforme en la nube, en las instalaciones o en el extremo de la red, con una experiencia similar a la nube.
Red Hat Ceph Storage	5	Solución de almacenamiento simplificada, abierta y con gran capacidad de ajuste para los canales modernos de datos. Está diseñada para el análisis de datos, la inteligencia artificial y el aprendizaje automático (IA/ML), y las cargas de trabajo más recientes. Ofrece almacenamiento definido por el software en el sistema de hardware estándar del sector que usted elija.
Red Hat OpenShift Data Foundation	4.9.2	Almacenamiento definido por el software para los contenedores. OpenShift Data Foundation es la plataforma de servicios de datos y almacenamiento para Red Hat OpenShift y ayuda a los equipos a desarrollar e implementar las aplicaciones en las nubes y en los hosts de servidores dedicados (bare metal) de manera eficiente y rápida.
Red Hat OpenShift Virtualization	4.9.2	La solución de Red Hat para ejecutar las VM en un clúster de Kubernetes. Tiene dos objetivos: el primero es lograr que todos los usuarios consoliden sus cargas de trabajo en una sola plataforma, para reducir la carga operativa de tener que gestionar una plataforma de virtualización adicional junto con la de contenedores, ya sea que utilicen las VM desde mucho tiempo o recién comiencen a trabajar con ellas. El segundo es que los usuarios puedan aprovechar el poder del motor y del ecosistema de Kubernetes para modernizar las funciones, la organización y la arquitectura de sus cargas de trabajo tradicionales.

Elementos físicos

Clúster de RHCS

10 * servidores en rack DELL PowerEdge R640:

Elemento	Especificaciones	Comentarios
CPU	40 núcleos	2* CPU Intel(R) Xeon(R) Gold 6230 de 2,10 GHz
Memoria	384 GB de RAM ECC	12 * SK Hynix 1x 32 GB DDR4-3200 RDIMM PC4-25600R Módulo Dual Rank x4
SSD (disco principal)	446,63 GB - 6 Gbps	SSD MICRON MTFDDAK480TDT
SSD (almacenamiento)	3574 GB - 12 Gbps	2 * SSD TOSHIBA KPM5XVUG1T92 de 1787,88 GB
NVME (almacenamiento)	2980,82 GB - 8 GT/s	NVME Samsung S5CXNA0N607551

37 * servidores en rack DELL PowerEdge R650:

Elemento	Especificaciones	Comentarios
CPU	56 núcleos	2 * CPU Intel(R) Xeon(R) Gold 6330 de 2,00 GHz
Memoria	384 GB de RAM ECC	12 * SK Hynix 1x 32 GB DDR4-3200 RDIMM PC4-25600R Módulo Dual Rank x4
SSD (disco principal)	446,63 GB - 6 Gbps	SSD MICRON MTFDDAK480TDT
SSD (almacenamiento)	3574 GB - 12 Gbps	2 * SSD TOSHIBA KPM5XVUG1T92 de 1787,88 GB
NVMe (almacenamiento)	2980,82 GB - 8 GT/s	NVME Samsung S5CXNA0N607551

Nota: El sistema de hardware que se utilizó para RHCS no era óptimo para la tarea porque el tamaño de los discos y las arquitecturas en su clúster no eran uniformes, lo cual afectó el rendimiento de Ceph. No obstante, esta es otra prueba fehaciente de la versatilidad que ofrece la plataforma.

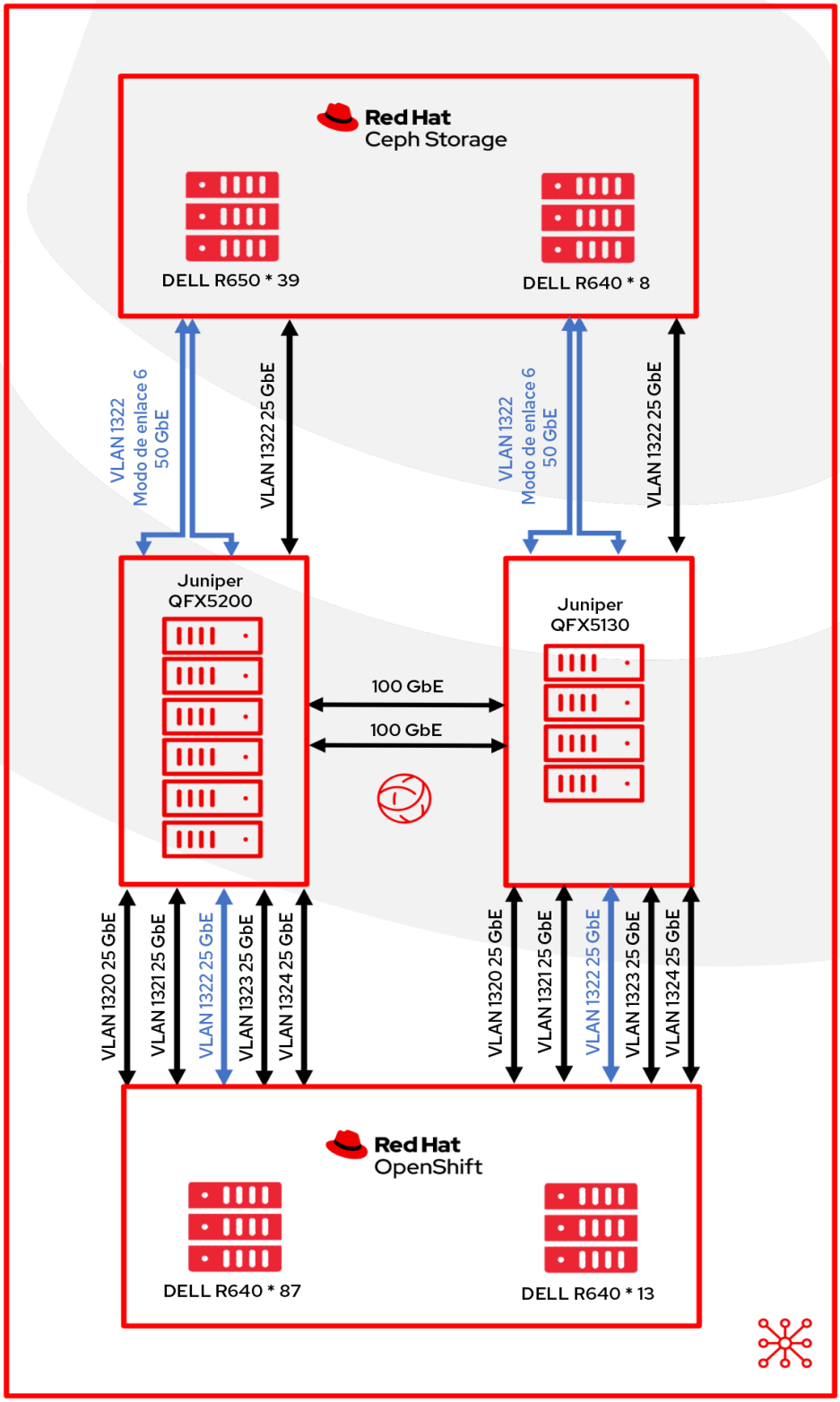
Clúster de OpenShift

100 * servidores en rack DELL PowerEdge R640:

Elemento	Especificaciones	Comentarios
CPU	40 núcleos	2 * CPU Intel(R) Xeon(R) Gold 6230 de 2,10 GHz 20 núcleos
Memoria	384 GB de RAM ECC	12 * SK Hynix 1x 32 GB DDR4-3200 RDIMM PC4-25600R Módulo Dual Rank x4
SSD (disco principal)	446,63 GB - 6 Gbps	SSD MICRON MTFDDAK480TDT

Arquitectura

En el siguiente diagrama, se muestra la arquitectura de red de los clústeres de OpenShift y Ceph. Tenga en cuenta que la ruta de datos entre el clúster de Ceph y el de Red Hat OpenShift Container Platform (OCP) en la red de área local virtual (VLAN) de un laboratorio privado utiliza el enlace de equilibrio de carga adaptable (balance-alb) con 2 * puertos de 25 GbE.



Red Hat

Ajuste de la red de RHCS

En esta sección, se explica la manera en que se ajustó la red de Linux en los nodos de Ceph para que se adaptara al entorno de grandes dimensiones.

Ajuste del protocolo de resolución de direcciones (ARP)

Flujo de ARP

El problema de flujo de ARP puede afectar cualquier host de Linux que tenga varias interfaces de red en la misma subred, y se puede producir cuando el host responde a una solicitud de ARP para esas interfaces. Si bien esto no siempre representa un problema, en algunos casos, puede generar un comportamiento incorrecto en las aplicaciones debido a la asignación incorrecta entre las direcciones IPv4 y MAC.

En los hosts que se basan en RHEL, podemos corregir el comportamiento editando /etc/sysctl.d/99.8-arp.conf en todos los hosts de RHCS y agregando estas líneas de código:

```
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**: le permite tener varias interfaces de red en la misma subred, y la respuesta a las solicitudes de ARP para cada una dependerá de la manera en que actuaría el kernel: si enrutaría un paquete desde la IP que recibe la solicitud hacia la interfaz (por eso se debe utilizar el enrutamiento basado en el origen para que funcione). Es decir, permite controlar las tarjetas (por lo general, la número 1) que responderán la solicitud ARP.
- **ignore=1**: sirve para responder solo si la dirección IP de destino es local y está configurada en la interfaz entrante.
- **announce=1**: intenta evitar las direcciones locales que no se encuentran en la subred de destino para esta interfaz. Resulta útil cuando los hosts de destino a los que se puede llegar a través de esta interfaz necesitan que la dirección IP de origen en las solicitudes de ARP sea parte de su red lógica configurada en la interfaz receptora.

Ejecute este código para asegurarse de cargar los ajustes de red nuevos:

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

Memoria caché de ARP

La memoria caché de ARP conserva una lista de entradas ARP que se generan cuando se establece la correspondencia entre la dirección IP y la dirección MAC. Para evitar los casos a gran escala en los que la caché de ARP no tiene la capacidad para todas las entradas, será necesario aumentar su tamaño editando **/etc/sysctl.d/99.7-arpcachesize.conf** e incorporando estas líneas de código:

```
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
```

El valor numérico establece el límite a partir del cual recolectaremos los elementos no utilizados para las entradas de caché de destino IPv4. Cuando se duplique este valor, el sistema rechazará las asignaciones nuevas.

Ajuste de TCP/IP

Ajuste de la ventana de TCP

Es posible que los ajustes predeterminados de red de RHEL no generen la latencia ni el rendimiento óptimos para los trabajos grandes en paralelo que suelen formar parte de las configuraciones a gran escala. Para mejorar el rendimiento de los trabajos simultáneos, ajuste la red de Linux y ciertos dispositivos de esta forma:

Las redes con gran ancho de banda se pueden aprovechar mejor con una ventana de TCP más grande.
Por eso nos aseguramos de que la ventana de TCP se pueda ampliar.
Puede escribir **cat /proc/sys/net/ipv4/tcp_window_scaling** para verificarlo.

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

Escriba el siguiente código para que no se borre la configuración durante los reinicios:

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

Ajuste del tamaño del búfer

El siguiente paso será calcular el tamaño del búfer de envío y de recepción del socket. Por lo general, el búfer de lectura y escritura de cada socket puede contener un mínimo de dos paquetes, un valor predeterminado de cuatro o un máximo de diez. Si el búfer del socket de red es demasiado pequeño, podría llenarse y disminuir el rendimiento efectivo, lo cual afectaría el funcionamiento, pero si es lo suficientemente grande, puede mejorar el desempeño en cierta medida.

Antes de continuar, definiremos algunos términos:

- rmem_max: el tamaño máximo del búfer de recepción.
- wmem_max el tamaño máximo del búfer de envío.
- wmem_default: el tamaño predeterminado del búfer de envío.
- max_backlog: el tamaño máximo de la cola de recepción.
- Netdev_budget: la cantidad máxima de paquetes que se toman de todas las interfaces en un ciclo de sondeo.

Calculamos el tamaño ideal del búfer en función del tamaño del paquete, pero en los entornos con latencia alta, recomendamos calcularlo según la latencia.

Método de cálculo según la latencia

Optimice el búfer usando la latencia para calcular la productividad máxima de una sola conexión TCP.

Tamaño óptimo = (tiempo de ida y vuelta en microsegundos) x (tamaño del enlace en Mb/s) x 1024^2

Por ejemplo, nuestro enlace se ejecuta a 50 000 Mb/s. La latencia desde el nodo de Ceph hasta el clúster de OpenShift es de 0,208 dividido en 1000 para hacer la conversión a microsegundos. Luego, si se multiplica por 50 000, da como resultado 10,4, que convertido a bytes es 10 905 190.

O bien:

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

Ahora solo tenemos que configurar wmem_default, que contiene cuatro paquetes, es decir, ¼ de 10 905 190. La configuración debería verse así:

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

Método de cálculo según el tamaño del paquete

Optimización según el tamaño del paquete: otra opción es asumir que el tamaño promedio del paquete por archivo es de 512 KB; el tamaño ideal de cada socket sería: tamaño máximo = (tamaño del paquete en MB/s) x 1024^2.

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

Tenga en cuenta que para optimizar las redes que suelen experimentar latencia alta se prefiere utilizar el método de cálculo según la latencia.

Ajuste del adaptador

Búferes de tarjeta de interfaz de red (NIC)

En las configuraciones de gran tamaño con muchos hosts, la tasa de tráfico entrante podría superar la capacidad del kernel para purgar los búferes con la rapidez suficiente. Si esto sucede, se desbordarán los búferes de la tarjeta de interfaz de red (NIC), se perderá el tráfico y se lo contabilizará como pérdida de softirq. Podemos evitarlo si aumentamos el tiempo de CPU para

softirq, lo cual se conoce como netdev_budget, y así incrementaremos la capacidad tanto como sea necesario. En nuestra configuración, la aumentamos a 1000: softirq purgará 1000 mensajes en la NIC antes de salir de la CPU.

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

Puede suceder que la tercera columna en **/proc/net/softnet_stat** aumente de forma gradual con el tiempo:

```
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
```

Esto indica que softirq no obtuvo el tiempo suficiente de CPU y, en este caso, se puede aumentar la capacidad de a poco.

Cola de trabajos pendientes

En todos los kernels de Linux, el tráfico se almacena en una cola luego de que lo reciba la NIC y antes de que lo procese una stack de protocolo (TCP, IP o iSCSI).
Cada núcleo de CPU tiene una cola de trabajos pendientes en la cual se almacena el tráfico, y si alcanza su capacidad máxima, no podrá guardar ningún otro paquete.

Puede suceder que la segunda columna **/proc/net/softnet_stat** aumente de forma gradual con el paso del tiempo:

```
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
```

Esto indica que la cola de trabajos pendientes de netdev está desbordada y que se debe incrementar netdev_max_backlog de a poco.

En este ajuste, establecimos el valor en 5000.

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

Red Hat

Ajuste de RHCS

En esta sección, se explica el ajuste específico que se realizó en los nodos de Ceph para este entorno a gran escala.

Ajuste de los grupos de ubicación (PG)

Los PG son grupos de objetos que sirven para almacenar datos y metadatos y se replican a través de un dispositivo de almacenamiento de objetos (OSD).

Podemos conseguir la cantidad ideal de PG por grupo si configuramos 100 de ellos por OSD como objetivo (según las prácticas recomendadas para rbd y librados), luego lo multiplicamos por la capacidad máxima de uso del grupo (el valor predeterminado es 85 %), lo dividimos por la cantidad de réplicas y lo redondeamos a la potencia más cercana a 2 - 2^(round(log2(x))):

```
( Target PGs per OSD ) x ( OSDs ) x ( %Data (pool max used capacity))
-----------------------------------------------------------------------
                              ( 3 replicas )
```

En nuestros ajustes (100 * 141*0,85) / 3 es igual a 3995, que redondeado a una potencia de 2 da como total 4096 PG.

Lo programamos con la calculadora básica (bc):

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

Tenga en cuenta que puede aumentar aún más la cantidad de PG por OSD, lo cual podría disminuir la variación en la carga por OSD en su clúster, pero cada PG requiere un poco más de CPU y memoria en el OSD que lo almacena. Por eso es necesario probar y ajustar la cantidad de OSD para cada entorno.

El siguiente paso será aplicar los ajustes en el clúster:

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

Red Hat

Ajuste de Prometheus

El panel de Ceph muestra las estadísticas de su grupo, y puede utilizarse para supervisar los clústeres de la plataforma. Podemos ejecutar:

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

Para reducir la carga del sistema en los clústeres grandes, podemos limitar la recopilación de estadísticas del grupo con este código:

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

También es buena idea reducir la tasa de sondeo de Prometheus, para evitar que se genere un bloqueo en el gestor de Ceph y que otros complementos ceph-mgr no tengan tiempo de ejecutarse.

En este caso, el comando configura el intervalo de extracción en 60 segundos:

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

Red Hat

OpenShift Virtualization

Introducción

Para demostrar las funciones y la estabilidad de OpenShift Virtualization a gran escala, probaremos estos flujos de trabajo:

- Implementaciones de VM
- Arranques simultáneos de VM
- Aumento de la latencia en las VM con y sin cargas de trabajo
- Migración de las VM con y sin cargas de trabajo

Establecimos el objetivo de densidad para esta configuración en 3000 VM y 21 400 pods en todo el clúster. Utilizamos la siguiente configuración para lograrlo:

- 1500 VM de RHEL 8.5 con almacenamiento permanente
- 500 VM de Windows 10 con almacenamiento permanente
- 1000 VM de Fedora con almacenamiento efímero
- 21 400 pods inactivos

En pocas palabras, es una densidad de 30 VM y 214 pods por nodo.

KubeletConfig

Para lograr las dimensiones que mencionamos al principio, tuvimos que ignorar el límite predeterminado de pods por nodo con esta configuración 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
```

Además de aumentar `maxPods` a 500 (el valor predeterminado es 250), también incrementamos el valor predeterminado de `kubeAPIBurst` a 200 (suele ser 100) y de `kubeAPIQPS` a 100 (suele ser 50), para obtener la mayor capacidad posible. A modo de comparación general, la cantidad máxima de pods que utilizamos de forma predeterminada para Kubernetes estándar es de 110 pods por nodo.

Tenga en cuenta que no es necesario que implemente ninguna de las modificaciones anteriores. En la actualidad no es recomendable superar los 250 pods por nodo porque no se ha probado su funcionamiento a largo plazo. No obstante, en nuestra prueba no experimentamos ningún problema relacionado con la densidad.

Aplicamos una configuración KubeletConfig adicional relacionada con BZ#1984442, lo cual permite distribuir los pods de las VM de forma uniforme en todos los nodos:

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

Ambas configuraciones personalizadas KubeletConfig se pueden habilitar para los nodos de trabajo utilizando la etiqueta:

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

Tenga en cuenta que las modificaciones en las configuraciones KubeletConfig reiniciarán los nodos asociados.

Plantillas

Todas las plantillas del SO que utilizamos son las que están disponibles en el asistente de OpenShift Virtualization como plantillas predeterminadas, con algunos cambios en nuestra red personalizada.

Red Hat Linux

Puede usar este código para recuperar la plantilla utilizada:

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

Esta es la plantilla completa con las modificaciones que realizamos:

```
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

Puede usar este código para recuperar la plantilla utilizada:

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

Esta es la plantilla completa con las modificaciones que realizamos:

```
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

Puede usar este código para recuperar la plantilla utilizada:

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

Esta es la plantilla completa con las modificaciones que realizamos:

```
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
```

Implementación de VM

La implementación de las VM es la base de todo entorno virtual. Si el objetivo es tener un entorno con muchas VM, la velocidad con la que se implemente una mayor cantidad de máquinas influirá directamente en la eficiencia de la producción.

En esta sección, mostraremos el tipo de rendimiento esperado al clonar varias imágenes de VM a partir de una fuente de imágenes de referencia con el método de interfaz de almacenamiento de contenedores (CSI) de Ceph.

Tenga en cuenta que, si utiliza esta estrategia para clonar más de 100 VM, es posible que no se puedan eliminar los clones a través de la CSI de Ceph ni de forma manual (BZ#2055595). Por lo tanto, en este caso se recomienda evitar la clonación de las instantáneas y, en su lugar, utilizar `cloneStrategy: copy`.

Para habilitar la función de clonación de instantáneas de CSI, se debe editar el perfil de almacenamiento de OpenShift Virtualization:

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

También se debe agregar esta especificación:

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

Primero tenemos que importar una imagen QCOW de RHEL de uno de nuestros hosts a un volumen de datos (DV):

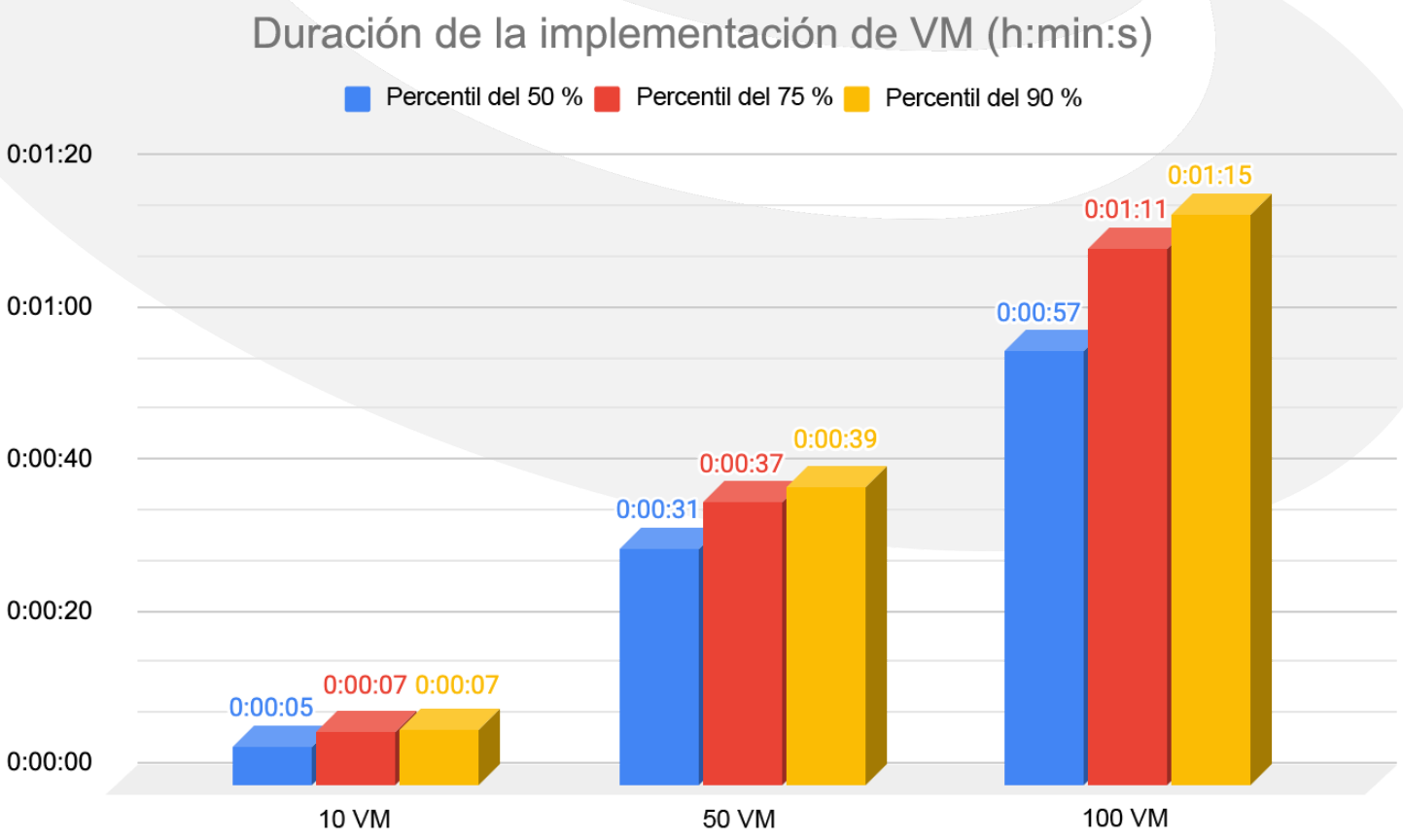
```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: rhel-clone-dv
spec:
  source:
      http:
         url: http://internel.server.com/ISO/rhel8.qcow2
  pvc:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 40Gi
    volumeMode: Block
    storageClassName: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

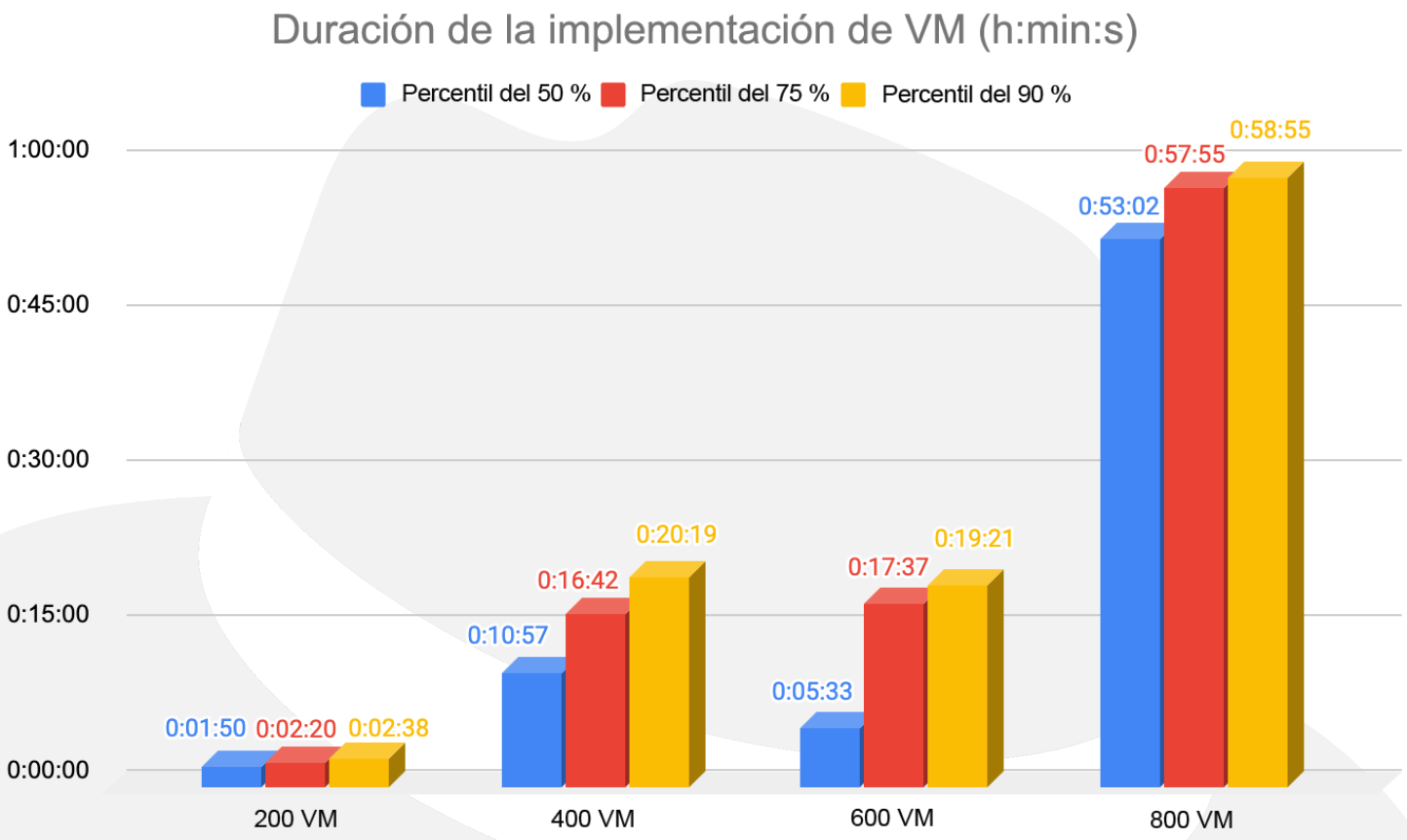
Luego de finalizar la importación, implementamos la cantidad deseada de VM en paralelo y calculamos el tiempo que demoran en completar la clonación realizándoles consultas con intervalos de 2 segundos hasta que termine el proceso.

Lo más efectivo será implementar grupos de 100 VM; es decir, se implementan 100 VM, se espera a que finalice el proceso de clonación y se implementan las siguientes 100.

En general, cuando se implementen más de 10 VM en paralelo, se generará una penalización, ya que se produce un bloqueo en el ID de volumen de la imagen principal de la CSI de Ceph. Por lo tanto, el proceso de clonación a partir de la misma imagen principal se realiza en serie y no en paralelo, y el proveedor externo solo puede enviar 10 llamadas gRPC en simultáneo al controlador de la CSI en cualquier momento.

Además, luego de superar los 250 clones, las imágenes RBD comenzarán a nivelarse, y esto aumentará aún más la penalización del proceso de clonación. El tiempo que lleva la nivelación de un clon aumenta con el tamaño de la instantánea.





Arranques simultáneos de las VM

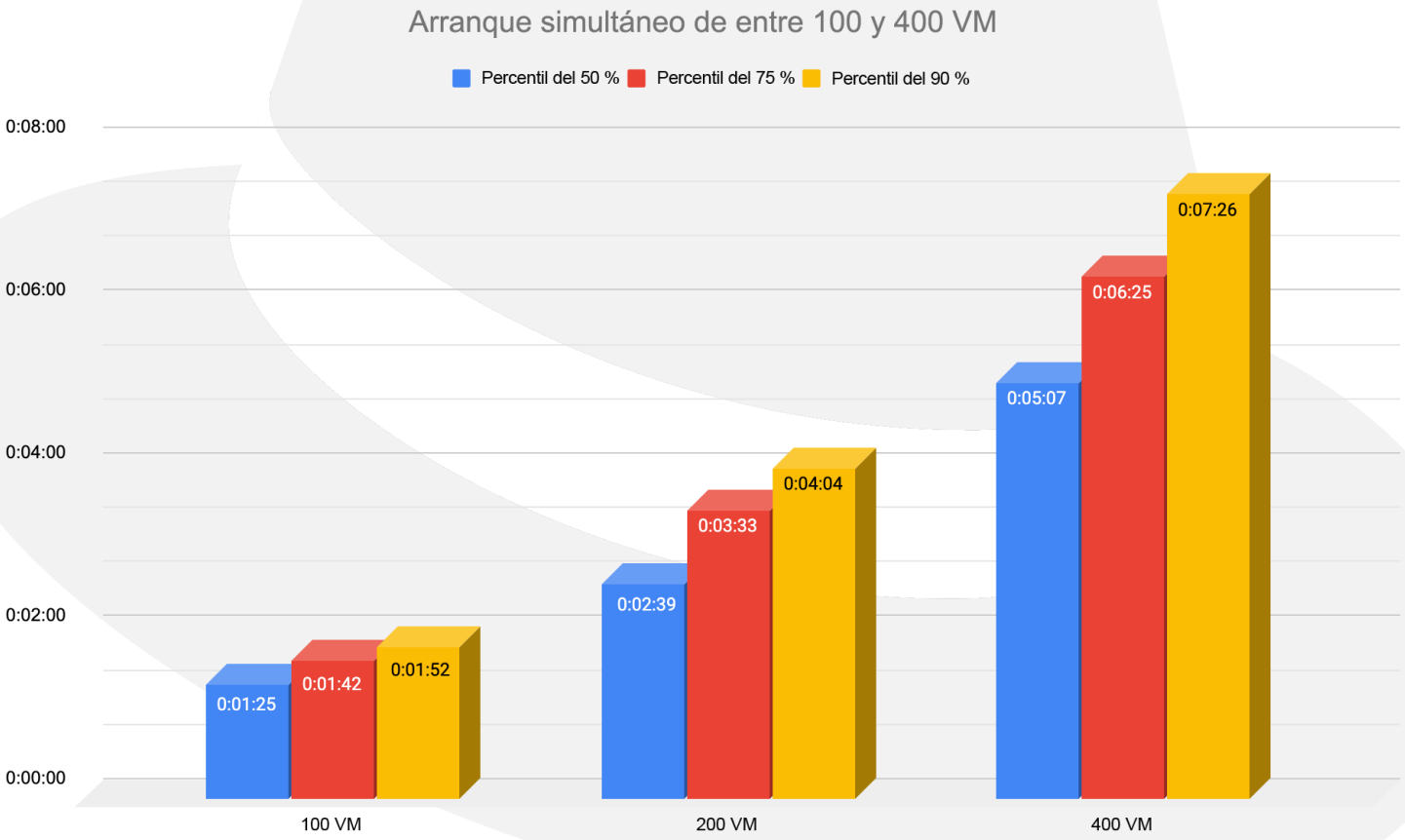
En este caso, probamos cuánto tiempo demora el arranque de varias VM, lo cual pone de manifiesto la resistencia de OpenShift Virtualization y del plano de control. Estos casos suelen ocurrir durante la recuperación del entorno ante un desastre, como el corte de la energía eléctrica.

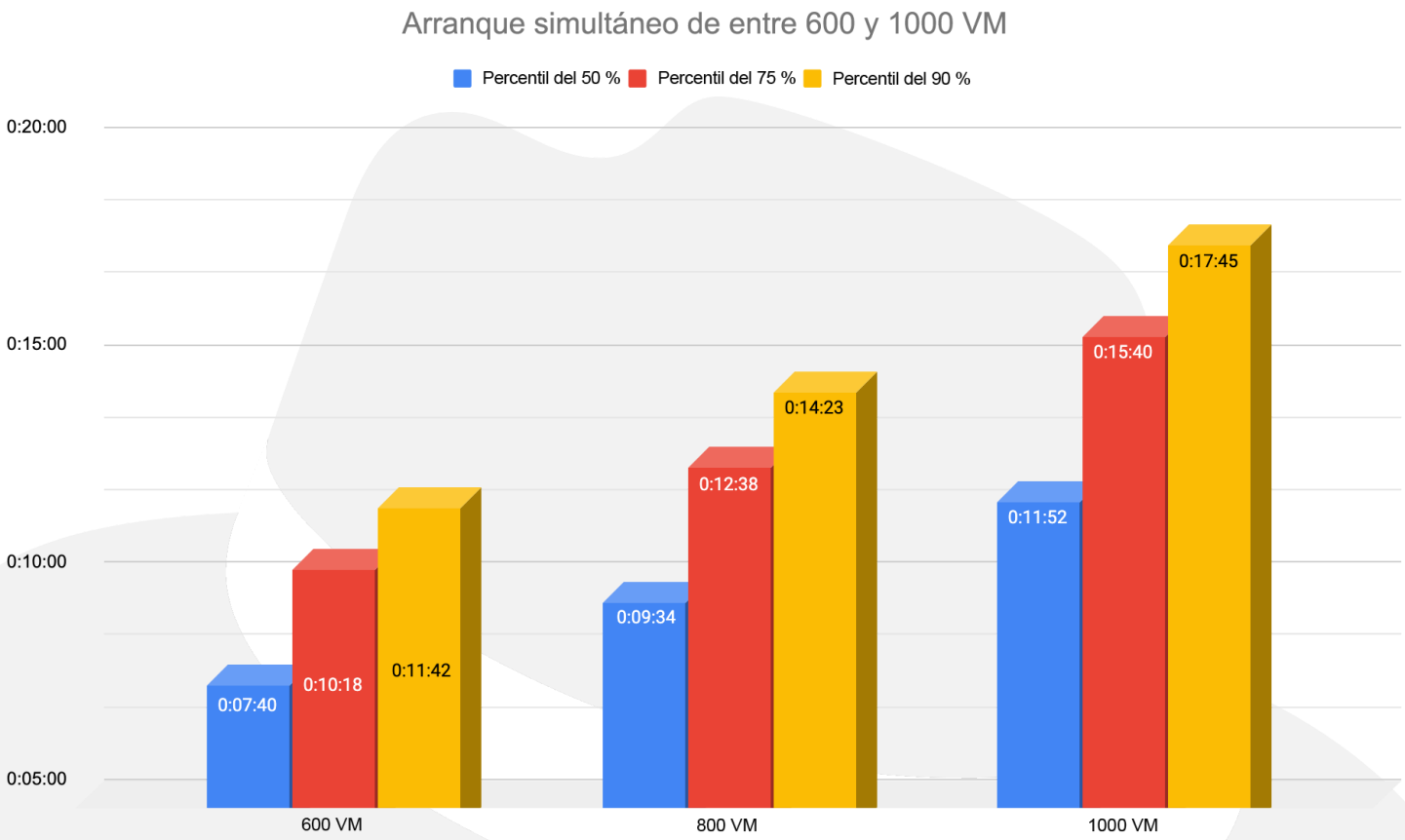
La medición de cada una de las VM empieza en el momento de la solicitud y finaliza cuando la VM comienza a ejecutarse y se puede acceder a ella mediante el protocolo SSH, lo cual significa que el host se inició correctamente y que el daemon de SSH está activo. Para calcular los

tiempos, ejecutamos una consulta en cada VM. Cuando el estado cambia a "En ejecución", intenta conectar el SSH a la VM cada dos segundos, hasta que finalmente pueda hacerlo.

Tal como sucede en la implementación, todas las solicitudes de inicio de las VM se ejecutan en paralelo para exigir a todas las partes del clúster.

Como se muestra en el siguiente gráfico, con nuestro clúster uniforme de OCP, los tiempos de inicio son casi lineales hasta las 1000 VM, pero las colas largas pueden ralentizarlos.




Arranque simultáneo de entre 600 y 1000 VM
Percentil del 50 %
Percentil del 75 %
Percentil del 90 %
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 VM
800 VM
1000 VM

Latencia de las VM

Cada VM tiene su propio subproceso de entrada y salida (I/O), a menos que haya varias reclamaciones de volumen permanente (PVC) y que `dedicatedIOThread:` esté configurado como "true". En este caso, cada PVC tendrá su propio subproceso de I/O.

En el siguiente caso, usamos 15 VM por nodo de trabajo y probamos hasta 64 de ellos; es decir, 960 VM, para demostrar que a pesar de que varios subprocesos acceden al clúster de RHCS a la misma vez, por sí mismos no generarán penalidades de latencia.

Para esta configuración elegimos un bloque de 4 KB para las operaciones de lectura y escritura aleatorias, y ejecutamos estas pruebas:

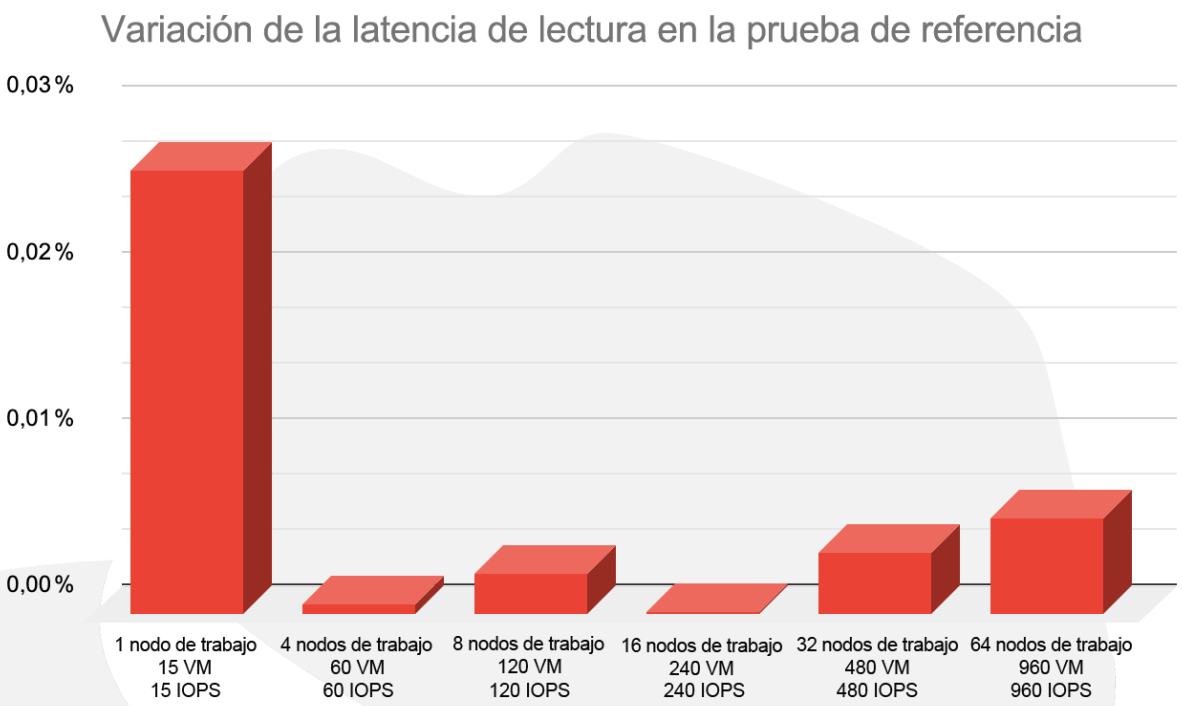
- Prueba de referencia: utilizamos 15 VM de cada nodo de trabajo, y cada una de ellas generó una sola IOPS. Comenzamos con 15 VM (15 IOPS, un solo nodo) y finalizamos con 64 nodos, con un total de 960 VM (960 IOPS).
- Prueba con carga de trabajo: utilizamos 15 VM de cada nodo de trabajo, y cada una de ellas generó 1000 IOPS. Comenzamos con 15 VM (15 000 IOPS, un solo nodo) y finalizamos con 64 nodos, con un total de 960 VM (960 000 IOPS).

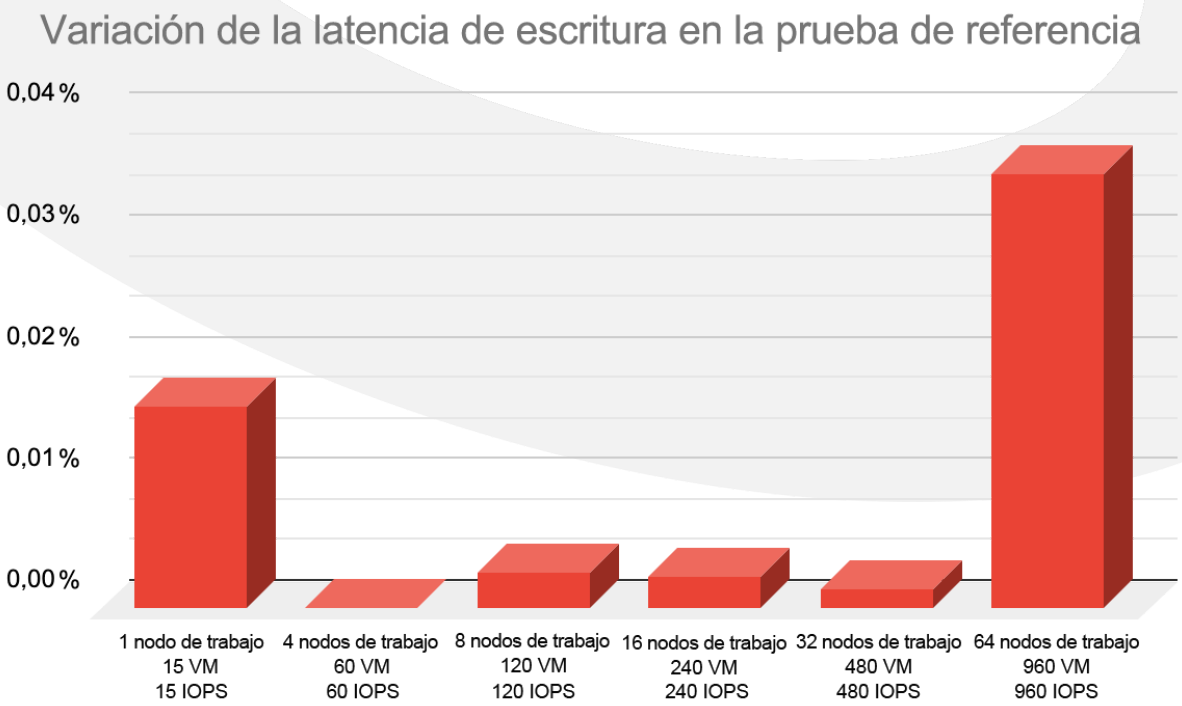
Cada conjunto de datos del sistema de archivos de las VM consta de 300 directorios con 8 archivos de 20 MiB cada uno. Dicho de manera más sencilla, cada VM tiene un conjunto de datos de 4,8 GiB.

Seleccionamos un bloque pequeño para evitar, en la medida de lo posible, que se produjeran discrepancias a causa de las conexiones de red y la falta de uniformidad en los discos de Ceph en el clúster de RHCS.

Tenga presente que, aunque utilizamos un máximo de 960 VM en la prueba, en realidad había 3000 VM y 21 400 pods en ejecución en el clúster.

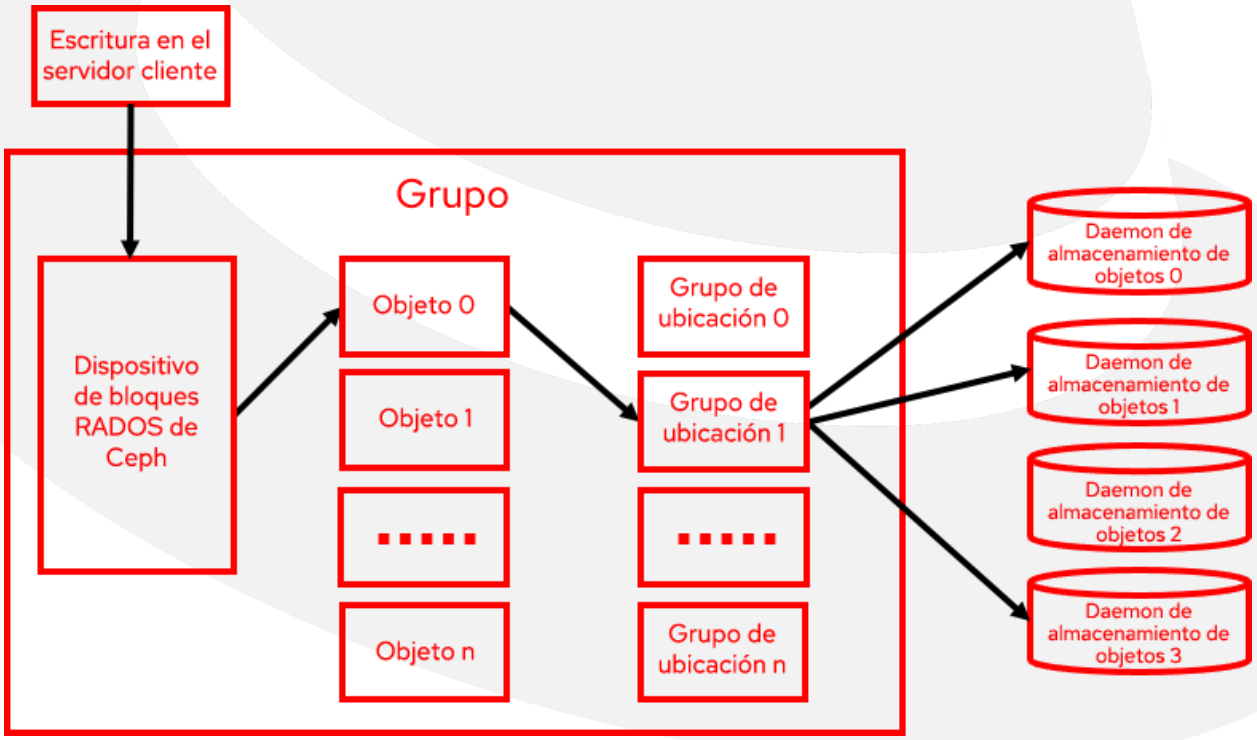
Como se puede observar en los siguientes gráficos, durante la ejecución de las pruebas de referencia, la latencia de las operaciones de lectura y escritura aleatorias varió en menos de un 0,04 %.


Variación de la latencia de lectura en la prueba de referencia
0,03 %
0,02 %
0,01 %
0,00 %
1 nodo de trabajo
15 VM
15 IOPS
4 nodos de trabajo
60 VM
60 IOPS
8 nodos de trabajo
120 VM
120 IOPS
16 nodos de trabajo
240 VM
240 IOPS
32 nodos de trabajo
480 VM
480 IOPS
64 nodos de trabajo
960 VM
960 IOPS

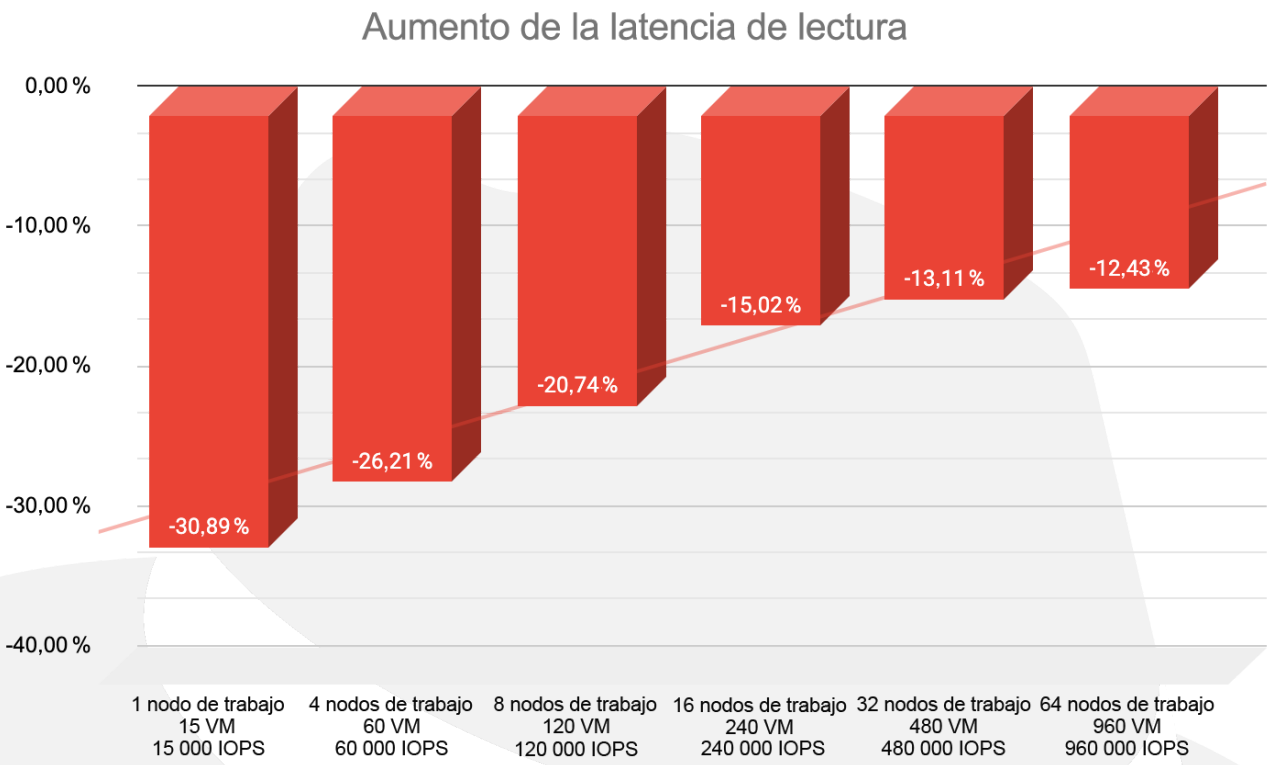

Variación de la latencia de escritura en la prueba de referencia
0,04 %
0,03 %
0,02 %
0,01 %
0,00 %
1 nodo de trabajo
15 VM
15 IOPS
4 nodos de trabajo
60 VM
60 IOPS
8 nodos de trabajo
120 VM
120 IOPS
16 nodos de trabajo
240 VM
240 IOPS
32 nodos de trabajo
480 VM
480 IOPS
64 nodos de trabajo
960 VM
960 IOPS

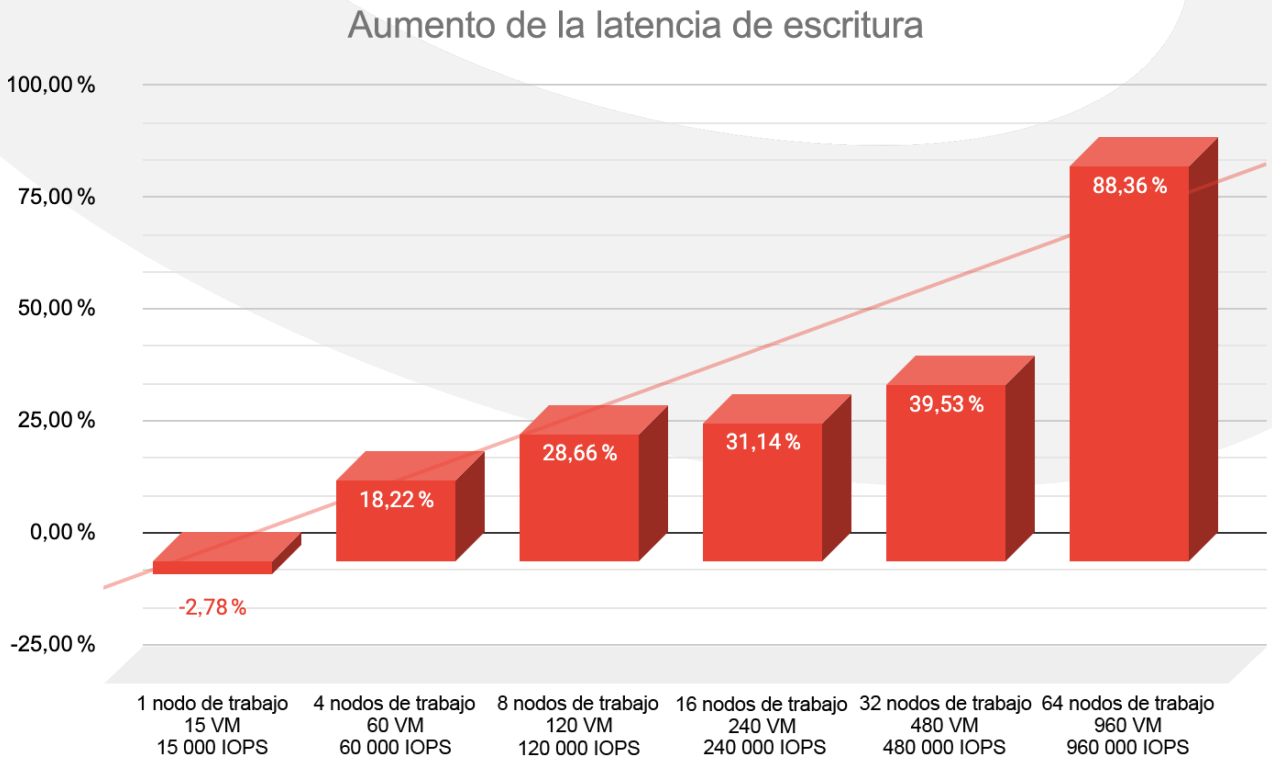
En el siguiente gráfico se comparan las variaciones de la latencia en la prueba con carga de trabajo en comparación con las obtenidas en la de referencia (los valores más bajos indican los mejores resultados). En cuanto al rendimiento de lectura, un índice de IOPS más alto generará menor latencia hasta cierto punto, dada la asignación de recursos que tiene lugar durante los picos de actividad más elevados en comparación con la carga de trabajo de IOPS inactiva o baja en las VM.

Las operaciones de escritura, por el contrario, tienen un flujo de datos distinto, que es necesario para mantener la alta disponibilidad. Como se muestra en el diagrama, por cada operación de escritura de Ceph, pasan por la red tres copias de los datos hacia cada uno de los OSD respectivos. Por lo tanto, la latencia de escritura se verá afectada.



Tenga en cuenta que en el caso de las aplicaciones que deben responder con rapidez ante eventos específicos, es posible disminuir un tercio de la sobrecarga de latencia de la escritura por cada réplica de datos que se haya reducido.


Aumento de la latencia de lectura
0,00 %
-10,00 %
-20,00 %
-30,00 %
-40,00 %
-30,89 %
-26,21 %
-20,74 %
-15,02 %
-13,11 %
-12,43 %
1 nodo de trabajo
15 VM
15 000 IOPS
4 nodos de trabajo
60 VM
60 000 IOPS
8 nodos de trabajo
120 VM
120 000 IOPS
16 nodos de trabajo
240 VM
240 000 IOPS
32 nodos de trabajo
480 VM
480 000 IOPS
64 nodos de trabajo
960 VM
960 000 IOPS


Aumento de la latencia de escritura
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 nodo de trabajo
15 VM
15 000 IOPS
4 nodos de trabajo
60 VM
60 000 IOPS
8 nodos de trabajo
120 VM
120 000 IOPS
16 nodos de trabajo
240 VM
240 000 IOPS
32 nodos de trabajo
480 VM
480 000 IOPS
64 nodos de trabajo
960 VM
960 000 IOPS

Red Hat

Migración de las VM

En el siguiente caso, realizamos una prueba de migración de 1000 VM. Para que la simulación fuera lo más real posible, no solo migramos las VM, sino que también reiniciamos los nodos de trabajo en los que se alojan. Pudimos lograrlo dividiendo los nodos en tres zonas y luego aplicando la configuración de una máquina vacía en una zona específica. Esto generó que todos los nodos asociados a ella se reiniciaran luego de quitar las VM que se alojaban allí.

Primero, asignamos cada nodo de trabajo a una zona determinada mediante una etiqueta:

```
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=""
```

Luego, creamos un grupo de configuración de máquinas por cada zona. Recuerde que `maxUnavailable: 10` configura la cantidad de nodos que pueden fallar en cualquier momento del ciclo de vida de la zona:

```
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
```

También editamos el operador del clúster hiperconvergente:

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

Luego, establecimos los siguientes ajustes de migración para aumentar la cantidad que pueden ocurrir en paralelo:

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

En la prueba, concluimos que es muy recomendable aumentar la cantidad de pods virt-api a razón de **1 pod kubevirt-api por cada 750 VM**. Como en esta configuración ejecutamos 3000 VM, ampliamos a 4 la cantidad de pods de la API kubevirt.

En este caso, la función de ajuste automático ya está habilitada; puede supervisarla con Github#7101. En este momento, puede hacerlo de forma manual ejecutando un parche en el operador hiperconvergente:

```
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}]\"}]}]"}}}'
```

Luego, utilizamos la siguiente configuración de máquina para activar la migración:

```
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: ""
```

Migramos 1000 máquinas virtuales de estos proveedores:

- 400 VM de RHEL
- 400 VM de Fedora
- 200 VM de Windows

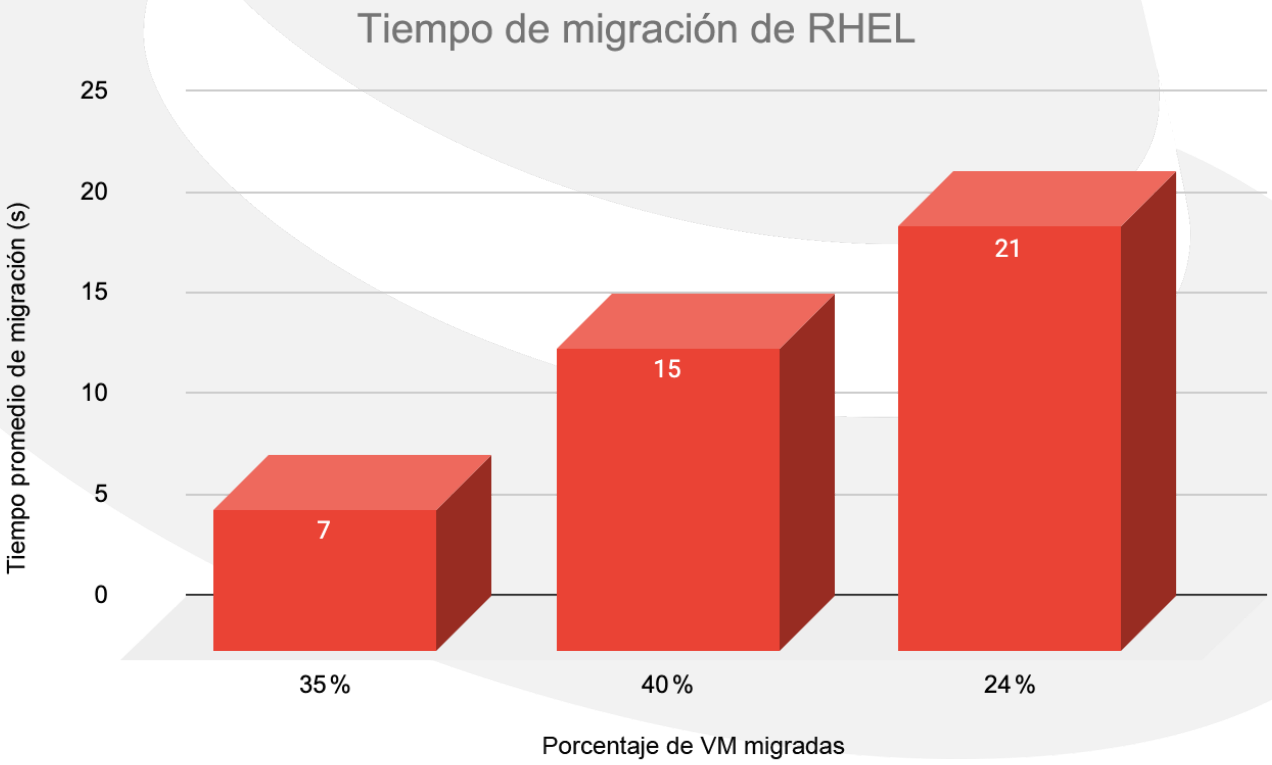
También reiniciamos en diferentes nodos de trabajo los 7000 pods en la misma ubicación que las VM.

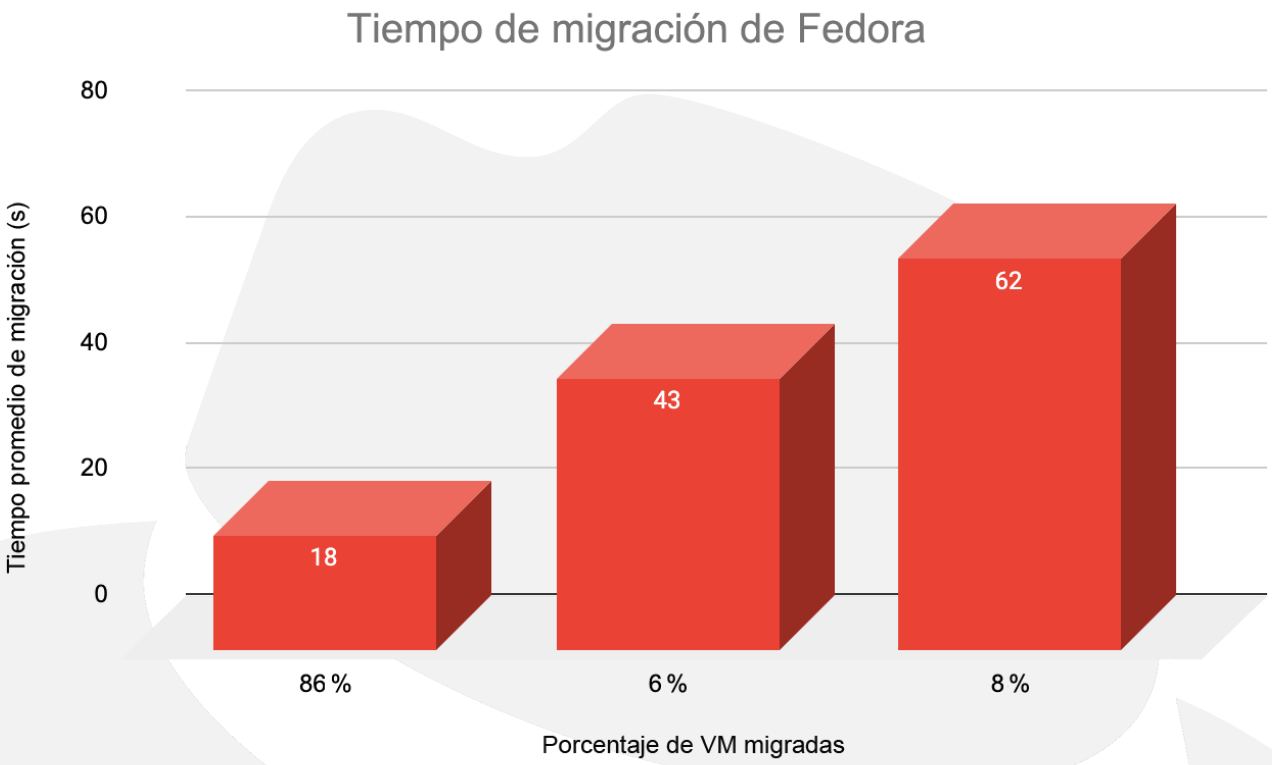
Durante la migración, podemos ver en los registros de VMI de cada VM el tiempo que demoró en completar el proceso:

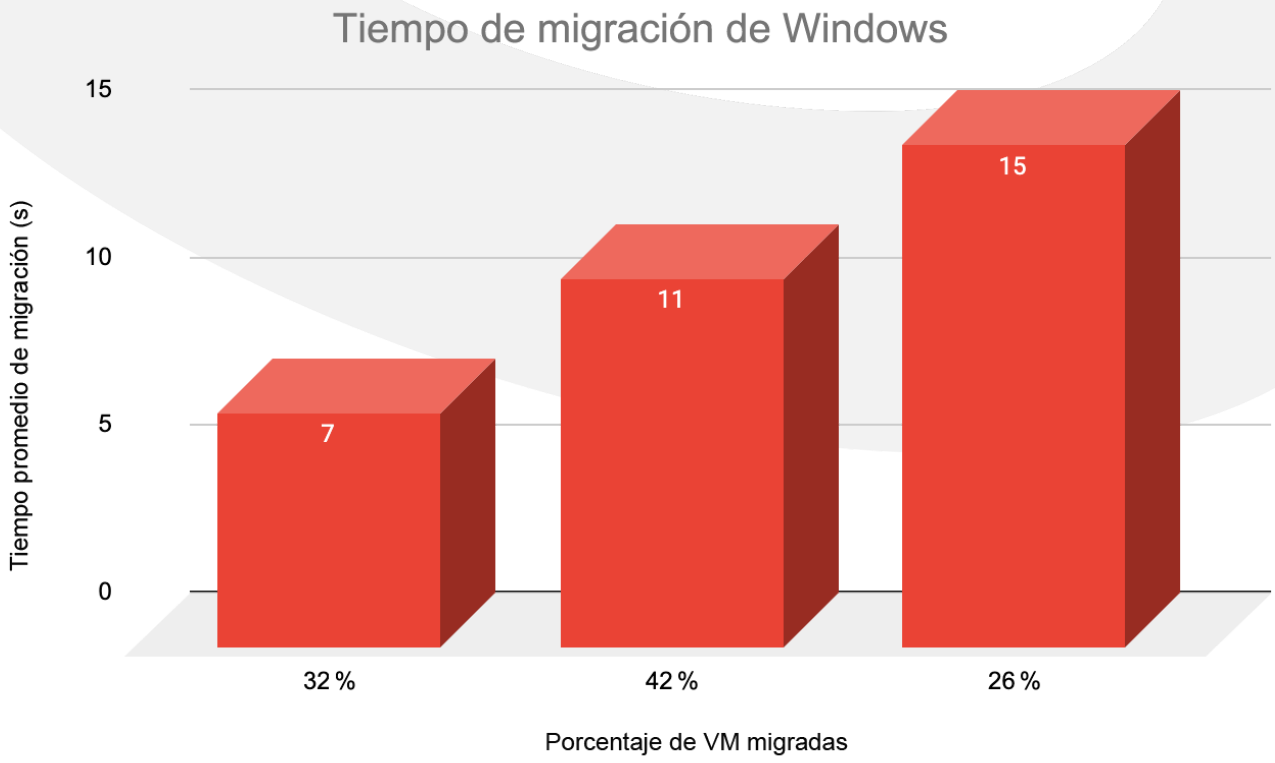
```
Phase Transition Timestamps:
  Phase:                        Scheduling
  Phase Transition Timestamp:   2022-04-10T07:15:13Z
  Phase:                        Scheduled
  Phase Transition Timestamp:   2022-04-10T07:15:23Z
  Phase:                        Running
  Phase Transition Timestamp:   2022-04-10T07:15:25Z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

En el siguiente gráfico, se muestran los tiempos de migración de cada SO en porcentajes. Por ejemplo, el tiempo promedio de migración para el 35 % de las VM de RHEL fue de 7 segundos.

Considere que el tiempo que tarda la migración no necesariamente está relacionado con el SO, sino con la carga del guest en el momento, la carga del host, la carga de la red, la tecnología de almacenamiento, la política de migración y el tamaño de la imagen, entre otros aspectos. Por eso los resultados podrían variar.




Tiempo de migración de Fedora
Tiempo promedio de migración (s)
80
60
40
20
0
18
43
62
86 %
6 %
8 %
Porcentaje de VM migradas


Tiempo de migración de Windows
Tiempo promedio de migración (s)
15
10
5
0
7
11
15
32 %
42 %
26 %
Porcentaje de VM migradas

Tiempo promedio de migración por SO:

SO	Tiempo promedio de migración (s)	Comentarios
RHEL	14	40 GiB PVC
Fedora	23	Expulsión del disco de contenedor. Estrategia: reinicio
Windows	12	40 GiB PVC

En resumen, todo el proceso tomó 118 minutos, y tuvimos 35 minutos de espera hasta que los nodos cambiaran al estado "Listo" debido a la configuración `maxUnavailable: 10` que mencionamos anteriormente.

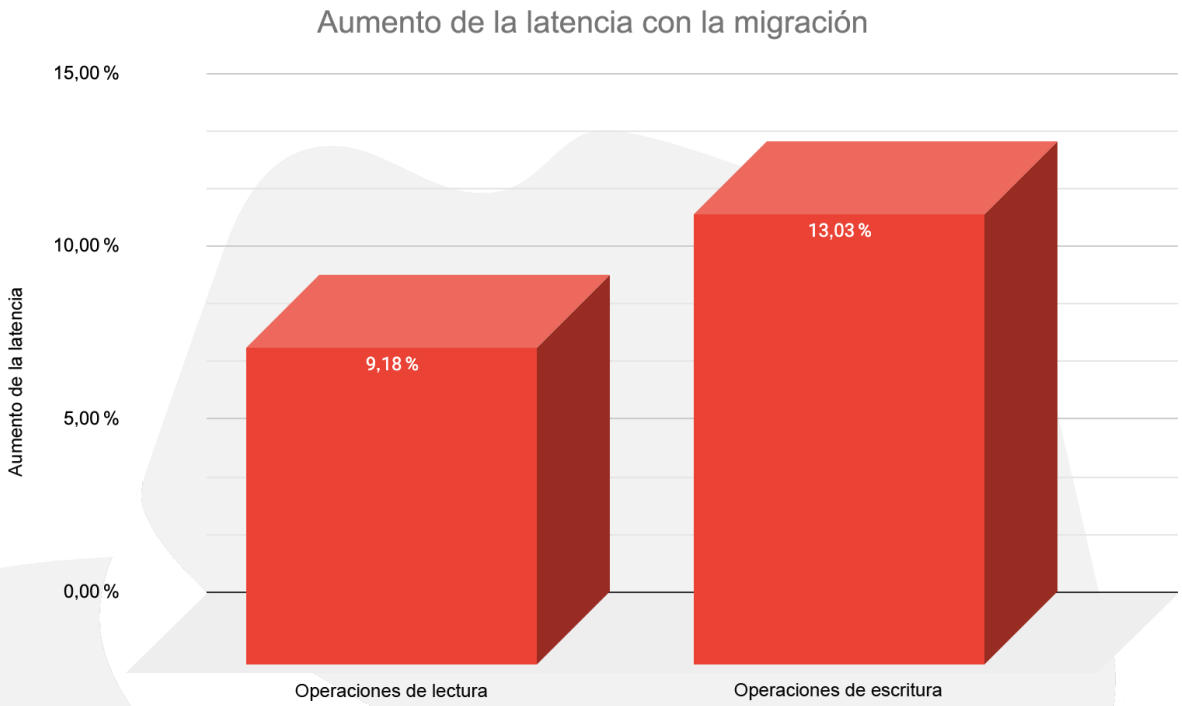
Aumento de la latencia con la migración de las VM

En el siguiente caso, probamos la migración de 1000 VM, pero esta vez solo utilizamos las VM de RHEL para evitar las posibles diferencias en la forma que tiene cada sistema operativo de gestionar la I/O.

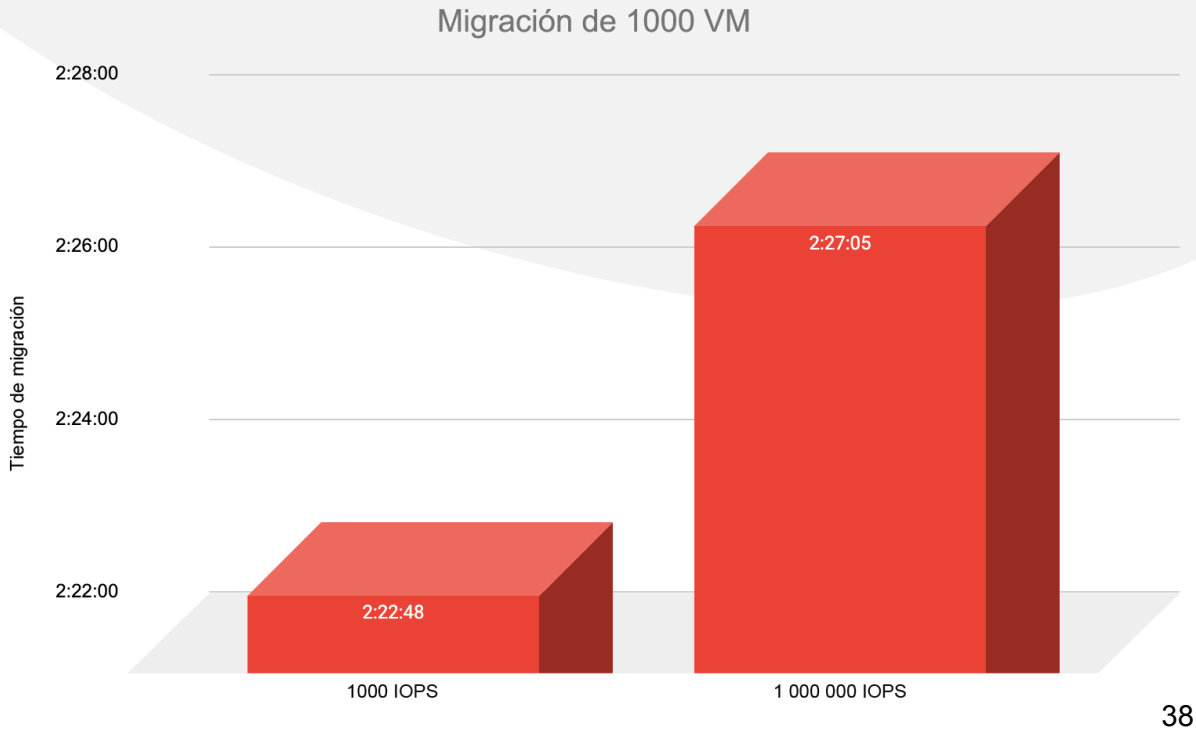
Al igual que en el caso anterior, elegimos un bloque de 4 KB para las operaciones de lectura y escritura aleatorias, y ejecutamos estas pruebas:

- Prueba de referencia: cada una de las 1000 VM genera una sola IOPS hasta un total de 1000 IOPS.
- Prueba con carga de trabajo: cada una de las 1000 VM genera 1000 IOPS por segundo hasta un total de 1 000 000 de IOPS.

En el siguiente gráfico se muestra la penalidad de latencia promedio durante la migración para las operaciones de lectura y escritura en comparación con la referencia (solo para las VM de RHEL):



Además, como podemos observar en el gráfico a continuación, solo hubo un impacto del 5 % en el tiempo de migración. Considere que el resultado del tiempo de migración puede variar debido a [BZ#2069098]():



Actualización del clúster según sea necesario

En el siguiente caso, probamos la actualización de una versión principal y una secundaria.

Primero actualizamos el clúster de la versión 4.9.15 a la 4.9.23 mientras simulamos la situación real en producción; esto significa que la totalidad de las 3000 VM y los 21 400 pods estaban en ejecución en el clúster. Además, generamos una carga de trabajo ligera de 4 KB en 1500 VM a razón de 100 IOPS por VM.

Para iniciar la actualización, ejecutamos:

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

Puede supervisar el proceso con estas líneas de código:

```
$ 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)
```

El proceso de migración de la versión secundaria tomó un total de **35 minutos**.

Luego, en la prueba actualizamos la versión principal del clúster de la 4.9.23 a la 4.10.9, con un entorno en las mismas condiciones que la vez anterior. Para iniciar la actualización, ejecutamos:

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

Al igual que antes, puede supervisar el proceso con estas líneas de código:

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

El proceso de migración de toda la versión principal tomó un total de **136 minutos**.

Tenga en cuenta que, en algunos casos, las actualizaciones incluyen modificaciones en las que es necesario restablecer todos los nodos de forma parcial, lo cual aumentará considerablemente el tiempo de actualización debido al tiempo de migración adicional.

Conclusión

En este documento sobre la arquitectura de referencia, demostramos las funciones y la resistencia de OpenShift Virtualization a gran escala. Es una función de Red Hat OpenShift Container Platform que, junto con Red Hat Ceph Storage y Red Hat OpenShift Data Foundation, puede ofrecer una solución de producción completa que incluye los contenedores, las máquinas virtuales y el almacenamiento de alta disponibilidad, y que puede implementarse en todo host que cumpla con los requisitos mínimos de hardware.

Si bien en este ejemplo se indica cómo lograr el objetivo establecido, es importante considerar otras arquitecturas para lograr la resistencia, la capacidad de adaptación y el buen funcionamiento de las operaciones diarias según los requisitos y las condiciones del entorno. Por ejemplo, cuando se superan ciertos límites, si la cantidad de nodos o la carga de trabajo aumentan de forma considerable o si se realizan demasiadas modificaciones de código en el clúster, debe considerarse el enfoque con varios clústeres.

Recursos adicionales

Requisitos del sistema y el entorno:
https://docs.openshift.com/container-platform/3.11/install/prerequisites.html

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

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

Migración activa y tiempos de espera:
https://docs.openshift.com/container-platform/4.9/virt/live_migration/virt-live-migration-limits.html

Actualización de un clúster con la CLI:
https://docs.openshift.com/container-platform/4.10/updating/updating-cluster-cli.html

Acerca de Red Hat

Red Hat es el proveedor líder mundial de soluciones de software open source para empresas, que ha adoptado un enfoque impulsado por la comunidad para ofrecer tecnologías confiables y de alto rendimiento de Linux, nube híbrida, contenedores y Kubernetes. Red Hat ayuda a que los clientes desarrollen aplicaciones en la nube, integren las aplicaciones de TI nuevas y actuales, y automaticen y gestionen los entornos complejos. Red Hat es un asesor de confianza de las empresas de la lista Fortune 500 y presta servicios galardonados de soporte, capacitación y consultoría para ofrecer los beneficios de la innovación abierta a todos los sectores. Red Hat es un centro de conexión en una red internacional de empresas, partners y comunidades, a los que ayuda a crecer, transformarse y prepararse para el futuro digital.