OpenShift Virtualization

Con lo storage esterno Red Hat Ceph Storage 5

Ottimizzazione e prestazioni su larga scala

Architettura di riferimento

Boaz Ben Shabat

Contenuti

Destinatari

Scopo del presente documento è fornire assistenza ai responsabili dei servizi per l'infrastruttura, tra cui clienti, sales engineer, consulenti sul campo e solution architect.
Questo documento illustra un esempio di deployment su larga scala di OpenShift Virtualization, una funzionalità di Red Hat OpenShift® Container Platform, che utilizza RHCS come soluzione storage di rete esterna ad alta disponibilità (HA).

Riepilogo

Questo documento descrive le nozioni apprese dal team Red Hat OpenShift Virtualization Performance and Scale a seguito di un deployment su larga scala completato con successo che include sia un cluster esterno Red Hat® Ceph® Storage 5 (RHCS) (47 nodi) che Red Hat OpenShift Virtualization (100 nodi), dove il cluster Ceph esterno fornisce lo storage alle macchine virtuali (VM) di OpenShift Virtualization per un totale di 3000 VM e 21.400 pod.

Questa architettura di riferimento illustra i passaggi necessari per ottimizzare RHCS e Red Hat OpenShift Virtualization e creare un cluster OpenShift resiliente da 100 nodi.

Inoltre, verranno spiegate le motivazioni alla base della procedura e fornite informazioni che consentiranno di applicare queste indicazioni a qualsiasi cluster.

La seguente tabella mostra i risultati delle prestazioni per i principali scenari che potrebbero verificarsi in un ambiente di produzione:

Scenario	Descrizione	Risultato
Deployment delle VM	**Deployment parallelo, massimo 800 VM**	**I risultati dei test mostrano che è possibile ottenere i migliori tempi di deployment con la clonazione a blocchi di 100 macchine virtuali.**

Boot storm delle VM	Boot storm paralleli, massimo 1000 VM	I tempi di avvio quasi lineari sono iniziati alle 01:42 (MM:SS) per 100 VM e sono terminati alle 17:45 per 1000 VM.
Latenza delle VM	Latenza di inattività sostenuta rispetto alla latenza del carico di lavoro sia in lettura che in scrittura	La latenza di inattività delle VM non è influenzata dal numero di thread I/O che accedono a RHCS; a fronte di 1 milione di IOPS, la latenza in lettura si è ridotta con percentuali fino al 30%, mentre quella in scrittura è aumentata fino all'88%.
Migrazione delle VM	Migrazione di 1000 VM	La migrazione di 1000 macchine virtuali e la rimozione di 7000 pod hanno richiesto circa 118 minuti (HH:MM).
Latenza aggiuntiva della migrazione delle VM	Migrazione di 1000 macchine virtuali con Red Hat Enterprise Linux® (RHEL) con carico di lavoro	La latenza I/O durante la migrazione è aumentata del 9% in lettura e del 13% in scrittura, il tempo di migrazione è aumentato del 3% e il tasso IOPS effettivo non è stato influenzato.
Upgrade del cluster OpenShift	Aggiornamento della versione del cluster OpenShift	L'upgrade secondario ha richiesto 35 minuti, quello principale 136 minuti.

Componenti software

Prodotto	Versione	Descrizione
Red Hat OpenShift	4.9.15	Piattaforma Kubernetes enterprise leader di settore che offre un'esperienza simile al cloud in qualunque ambiente. Red Hat OpenShift offre la possibilità di creare, distribuire ed eseguire le applicazioni attraverso un'esperienza coerente nel cloud, negli ambienti on-premise o nell'edge della rete.

Red Hat Ceph Storage	5	Soluzione storage open source semplificata e altamente scalabile pensata per le attività di dati moderne. Progettata per l'analisi dei dati, l'intelligenza artificiale, l'apprendimento automatico e i carichi di lavoro emergenti, Red Hat Ceph Storage fornisce uno storage software-defined sull'hardware standard di settore che preferisci.
Red Hat OpenShift Data Foundation	4.9.2	Storage software-defined per container. Progettato come piattaforma dei servizi di storage e dati per Red Hat OpenShift, Red Hat OpenShift Data Foundation aiuta gli sviluppatori a realizzare e distribuire applicazioni sui cloud e sugli host bare metal in modo rapido ed efficace.
Red Hat OpenShift Virtualization	4.9.2	Soluzione Red Hat per l'esecuzione di VM su un cluster Kubernetes. OpenShift Virtualization si propone di raggiungere due obiettivi: il primo è quello di aiutare tutti gli utenti di VM, sia esperti che principianti, a consolidare i propri carichi di lavoro su un'unica piattaforma, riducendo così il dispendio operativo dovuto alla gestione di un'ulteriore piattaforma di virtualizzazione oltre a quella per container; il secondo è sfruttare la potenza del motore e dell'ecosistema Kubernetes per consentire agli utenti a modernizzare le funzionalità, gli agenti di orchestrazione e l'architettura dei carichi di lavoro tradizionali.

Componenti fisici

Cluster RHCS

10 server rack DELL PowerEdge R640:

Componente	Specifiche	Commenti
CPU	40 core	2 CPU Intel(R) Xeon(R) Gold 6230 da 2.10 GHz
Memoria	384 GB di RAM con ECC	12 SK Hynix 1x 32 GB DDR4-3200 RDIMM PC4-25600R Dual Rank x4 Module
SSD (disco di root)	446,63 GB - 6 Gbps	SSD MICRON MTFDDAK480TDT
SSD (storage)	3574 GB - 12 Gbps	2 SSD TOSHIBA KPM5XVUG1T92 1787,88 GB
NVME (storage)	2980,82 GB - 8 GT/s	NVME Samsung S5CXNA0N607551

37 server rack DELL PowerEdge R650:

Componente	Specifiche	Commenti
CPU	56 core	2 CPU Intel(R) Xeon(R) Gold 6330 da 2.00 GHz
Memoria	384 GB di RAM con ECC	12 SK Hynix 1x 32 GB DDR4-3200 RDIMM PC4-25600R Dual Rank x4 Module
SSD (disco di root)	446,63 GB - 6 Gbps	SSD MICRON MTFDDAK480TDT
SSD (storage)	3574 GB - 12 Gbps	2 SSD TOSHIBA KPM5XVUG1T92 1787,88 GB
NVME (storage)	2980,82 GB - 8 GT/s	NVME Samsung S5CXNA0N607551

Nota: l'hardware utilizzato per RHCS non è ottimale per l'attività in questione, in quanto dimensioni e architettura dei dischi non sono omogenee in tutto il cluster RHCS. Questo aspetto ha influito sulle prestazioni, ma è servito anche a dimostrare la versatilità di Ceph.

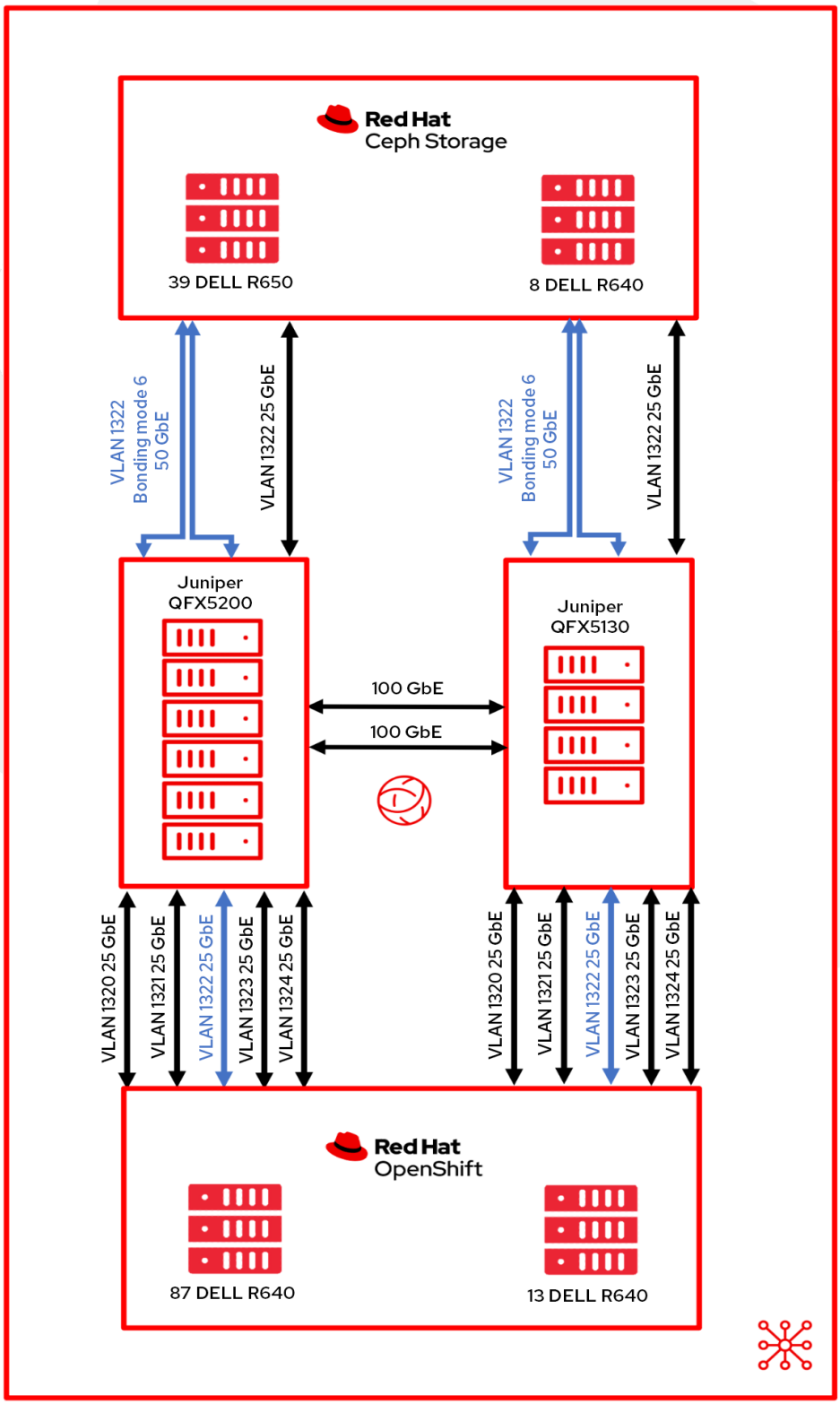
Cluster OpenShift

100 server rack DELL PowerEdge R640:

Componente	Specifiche	Commenti
CPU	40 core	2 CPU Intel(R) Xeon(R) Gold 6230 da 2.10 GHz e 20 core
Memoria	384 GB di RAM con ECC	12 SK Hynix 1x 32 GB DDR4-3200 RDIMM PC4-25600R Dual Rank x4 Module
SSD (disco di root)	446,63 GB - 6 Gbps	SSD MICRON MTFDDAK480TDT

Architettura

Questo diagramma mostra l'architettura di rete per i cluster OpenShift e Ceph. Il percorso dati tra il cluster Ceph e il cluster Red Hat OpenShift Container Platform (OCP) su una VLAN privata di laboratorio utilizza il bonding balance-alb con 2 porte da 25 GbE.



Red Hat

Ottimizzazione della rete per RHCS

Questa sezione descrive l'ottimizzazione della rete Linux eseguita sui nodi Ceph per soddisfare le esigenze degli ambienti di grandi dimensioni.

Ottimizzazione dell'Address Resolution Protocol (ARP)

Flusso ARP

Qualsiasi host Linux che disponga di più interfacce di rete sulla stessa subnet potrebbe essere interessato dal problema noto come "flusso ARP". Tale problema può verificarsi quando un host invia una risposta a una richiesta ARP per le interfacce sulla stessa subnet. Questo comportamento non è necessariamente preoccupante; tuttavia, in alcuni casi il flusso ARP potrebbe causare il malfunzionamento di alcune applicazioni a causa di una mappatura errata tra gli indirizzi IPv4 e gli indirizzi MAC.

Negli host basati su RHEL, è possibile correggere il problema modificando /etc/sysctl.d/99.8-arp.conf su tutti gli host RHCS e aggiungendo le seguenti righe:

```
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**: consente di avere più interfacce di rete sulla stessa subnet e di inviare una risposta agli ARP di ciascuna di esse a seconda che avvenga o meno il routing del pacchetto dall'IP interessato dall'ARP all'interfaccia da parte del kernel (è quindi necessario utilizzare il routing source-based per il corretto funzionamento). In altre parole, permette di controllare quali schede (di solito 1) invieranno le risposte alle richieste ARP.
- **ignore=1**: risponde solo se l'indirizzo IP di destinazione è un indirizzo locale configurato sull'interfaccia di entrata.
- **arp_announce=1**: consigliamo di evitare indirizzi locali non presenti nella subnet di destinazione per questa interfaccia. Questa modalità è utile quando gli host di destinazione raggiungibili tramite l'interfaccia richiedono che l'indirizzo IP sorgente nelle richieste ARP faccia parte della loro rete logica configurata sull'interfaccia ricevente.

Assicurati di caricare le nuove impostazioni di rete con il comando:

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

Cache ARP

La cache ARP contiene l'elenco delle corrispondenze generate tra indirizzi IP e indirizzi MAC. Affinché vi sia spazio sufficiente per tutte le voci in caso di scenari su larga scala, è necessario aumentare le dimensioni della cache ARP modificando **/etc/sysctl.d/99.7-arpcachesize.conf** e aggiungendo le seguenti righe:

```
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
```

Il valore numerico imposta la soglia a partire dalla quale si avvia il processo di garbage collection per le voci della cache nell'indirizzo IPv4 di destinazione. In presenza di un valore doppio, il sistema rifiuterà nuove allocazioni.

Ottimizzazione dei protocolli TCP/IP

Window scaling del TCP

Le impostazioni di rete predefinite di RHEL potrebbero non garantire prestazioni ottimali in termini di throughput/latenza per processi paralleli di grandi dimensioni solitamente presenti nelle configurazioni su larga scala. Ecco come ottimizzare la rete Linux e alcuni dispositivi di rete per migliorare le prestazioni dei processi paralleli:

Per sfruttare al meglio le reti a elevata larghezza di banda, è necessario aumentare il parametro di window size del TCP.
A tal fine, dobbiamo assicurarci di avere abilitato il window scaling del TCP.
È possibile effettuare la verifica tramite il comando
cat /proc/sys/net/ipv4/tcp_window_scaling

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

Per rendere permanente la modifica anche in caso di riavvio, utilizza:

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

Ottimizzazione delle dimensioni del buffer

Il passo successivo consiste nel calcolare la "dimensione del buffer di trasmissione" e la "dimensione del buffer di ricezione" del socket.
In generale, il buffer di lettura/scrittura di ogni socket può contenere un minimo di 2 pacchetti, un valore predefinito di 4 pacchetti o un massimo di 10 pacchetti. Se il buffer del socket di rete è di dimensioni troppo ridotte, potrebbe esaurirsi e ridurre il throughput effettivo, con conseguente impatto sulle prestazioni. Se invece è sufficientemente grande, può migliorare le prestazioni in una certa misura.

Ecco alcuni termini utili da conoscere:

- rmem_max: dimensione massima del buffer di ricezione.
- wmem_max: dimensione massima del buffer di trasmissione.
- wmem_default: dimensione predefinita del buffer di trasmissione.
- max_backlog: dimensione massima della coda di ricezione.
- Netdev_budget: numero massimo di pacchetti acquisiti da tutte le interfacce in un ciclo di polling.

Per calcolare la dimensione ottimale del buffer abbiamo utilizzato il metodo della dimensione del pacchetto; tuttavia, per le configurazioni che presentano una latenza elevata, si consiglia di utilizzare il metodo della latenza.

Metodo della latenza

È possibile eseguire l'ottimizzazione calcolando il throughput massimo di una singola connessione TCP utilizzando la latenza.

Dimensioni ottimali = (ritardo bidirezionale in microsecondi) x (dimensione del collegamento in Mb/s) x 1024^2

Per esempio, il nostro bonding ha una velocità di 50000 Mb/s. La latenza dal nodo Ceph al cluster OpenShift è 0,208; dividendola per 1000, otteniamo i microsecondi. Moltiplicandola per 50000, otteniamo 10,4, che equivale a 10905190 byte.

In alternativa:

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

Ora non ci resta che impostare wmem_default, che contiene 4 pacchetti, cioè ¼ di 10905190. Ecco come dovrebbe apparire l'impostazione:

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

Metodo delle dimensioni del pacchetto

Un altro metodo per l'ottimizzazione è quello delle dimensioni del pacchetto, che consiste nell'ipotizzare una dimensione media dei pacchetti per ciascun file di 512 KB; la dimensione ottimale di ogni socket si ottiene con la formula: dimensione massima = (dimensione del pacchetto in 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
```

Ricordiamo che l'ottimizzazione in base alla latenza è il metodo preferito per le reti che riscontrano spesso una latenza elevata.

Ottimizzazione della scheda

Buffer della NIC

Nelle configurazioni su larga scala che contengono più host, la velocità del traffico in ingresso potrebbe potenzialmente superare la capacità del kernel di svuotare i buffer abbastanza rapidamente. In casi simili, i buffer della NIC si sovraccaricano e il traffico viene perso e considerato come "softirq miss". Per evitare questo scenario, è possibile incrementare il tempo di CPU per softirq (netdev_budget) e aumentare il parametro budget secondo necessità. Nella nostra configurazione, abbiamo aumentato il parametro budget a 1000, il che significa che softirq svuoterà 1000 messaggi sulla NIC prima di rilasciare la CPU.

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

Se la terza colonna in **/proc/net/softnet_stat** aumenta gradualmente nel tempo:

```
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
```

Significa che softirq non ha ottenuto sufficiente tempo di CPU. In tal caso, è possibile aumentare il parametro budget, preferibilmente con incrementi graduali.

Coda del backlog

Il kernel Linux include una coda dove viene memorizzato il traffico dopo essere stato ricevuto dalla NIC, ma prima che venga elaborato da uno degli stack di protocollo (TCP/IP/ISCSI). Ogni core della CPU presenta una coda del backlog in cui viene memorizzato il traffico; se la coda è già al massimo della sua capacità, qualsiasi pacchetto aggiuntivo viene eliminato.

Se la seconda colonna in **/proc/net/softnet_stat** aumenta gradualmente nel tempo:

```
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
```

Significa che la coda del backlog di netdev è sovraccarica e che è necessario aumentare il parametro netdev_max_backlog, preferibilmente con incrementi graduali.

In questa configurazione di scala, abbiamo impostato il valore a 5000.

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

Ottimizzazione di RHCS

Questa sezione descrive l'ottimizzazione specifica per Ceph eseguita sui nodi Ceph per soddisfare le esigenze degli ambienti di grandi dimensioni.

Ottimizzazione dei gruppi di posizionamento (PG)

I PG sono una raccolta di oggetti replicati da un dispositivo di storage degli oggetti (OSD); ogni oggetto è un container per la memorizzazione di dati e metadati.

Possiamo ottenere il numero ottimale di PG per pool impostando il nostro target a 100 PG per OSD (secondo le procedure consigliate per rbd e librados), quindi moltiplicare per la capacità massima utilizzata del pool (il valore predefinito è 85%), dividere per il numero di repliche e arrotondare alla potenza di 2 - 2^(round(log2(x))) più vicina:

```
(PG target per OSD) x (OSD) x (%dati (capacità massima utilizzata del
pool))
----------------------------------------------------------------------
                              (3 repliche)
```

Oppure, nella nostra configurazione: (100 * 141*0,85) / 3 =3995, che arrotondato a una potenza di 2 ammonta a 4096 PG totali.

Ecco lo script ottenuto con basic calculator (bc):

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

È possibile aumentare ulteriormente il numero di PG per OSD, il che può potenzialmente ridurre la varianza del carico per OSD nel cluster; tuttavia, ciascun PG richiede un po' più di CPU e memoria sugli OSD in cui è memorizzato. Pertanto, il numero di OSD deve essere testato e ottimizzato per ciascun ambiente.

Il passaggio successivo consiste nell'applicare queste impostazioni al cluster:

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

Ottimizzazione di Prometheus

Per monitorare il cluster Ceph, è possibile utilizzare il dashboard Ceph per visualizzare le statistiche del pool Ceph. Eseguiamo il comando:

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

Per ridurre il carico sul sistema in caso di cluster di grandi dimensioni, è possibile limitare la raccolta delle statistiche dei pool con:

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

Si consiglia inoltre di abbassare la frequenza di polling di Prometheus, per evitare che Ceph Manager subisca rallentamenti e che non ci sia il tempo per l'esecuzione degli altri plug-in ceph-mgr.

In questo caso, il comando imposta l'intervallo di scraping a 60 secondi:

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

Red Hat

OpenShift Virtualization

Introduzione

Per dimostrare le funzionalità e la stabilità di OpenShift Virtualization su larga scala, verranno illustrati i seguenti flussi di lavoro:

- Deployment delle VM.
- Boot storm delle VM.
- Latenza aggiuntiva delle VM con e senza carico di lavoro.
- Migrazione delle VM con e senza carico di lavoro.

L'obiettivo di densità per questa configurazione era di 3000 VM e 21.400 pod nel cluster, ed è stato raggiunto in questo modo:

- 1500 VM di storage permanente con RHEL 8.5.
- 500 VM di storage permanente con Windows10.
- 1000 VM di storage temporaneo con Fedora.
- 21.400 pod inattivi.

O più semplicemente, una densità di 30 VM e 214 pod per nodo.

KubeletConfig

Per raggiungere le dimensioni menzionate nell'introduzione, abbiamo dovuto aggirare il limite predefinito di pod per nodo, applicando questa 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
```

Oltre ad aumentare il parametro `maxPods` a 500 (dal valore predefinito di 250), abbiamo incrementato il valore predefinito di `kubeAPIBurst` a 200 (in precedenza 100) e quello di `kubeAPIQPS` a 100 (da 50) per gestire il maggiore potenziale di bursting. Come termine di paragone, il numero massimo predefinito di pod per Kubernetes standard è 110 pod per nodo.

Nessuna delle modifiche di cui sopra è obbligatoria. Per il momento si sconsiglia di superare i 250 pod per nodo, poiché non è stato effettuato alcun test a lungo termine. Tuttavia, nel corso delle nostre prove, non abbiamo riscontrato alcun problema legato alla densità.

Abbiamo applicato un'ulteriore KubeletConfig relativa a BZ#1984442, che ci consente di ottenere una distribuzione uniforme dei pod delle VM su tutti i nodi:

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

È possibile abilitare entrambe le KubeletConfig personalizzate per i nodi di lavoro utilizzando un'etichetta:

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

Le modifiche alle KubeletConfig causeranno il riavvio dei nodi associati.

Modelli

Tutti i modelli di SO utilizzati sono quelli predefiniti disponibili nella procedura guidata di OpenShift Virtualization, con alcune modifiche per la nostra rete personalizzata.

Red Hat Linux

Il modello in uso può essere recuperato tramite il comando:

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

Di seguito una copia per completezza, comprensiva delle nostre modifiche:

```
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

Il modello in uso può essere recuperato tramite il comando:

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

Di seguito una copia per completezza, comprensiva delle nostre modifiche:

```
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

Il modello in uso può essere recuperato tramite il comando:

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

Di seguito una copia per completezza, comprensiva delle nostre modifiche:

```
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
```

Deployment delle VM

Il deployment delle VM è alla base di qualsiasi ambiente virtuale. Su ampia scala, la velocità di distribuzione di un gran numero di VM influisce direttamente sull'efficienza della produzione.

In questa sezione, illustreremo il tipo di prestazioni attese quando si clonano più immagini di VM da una sorgente golden image utilizzando il metodo di clonazione Ceph CSI.

Se si clonano più di 100 VM utilizzando la strategia di clonazione CSI-clone, è possibile che Ceph CSI non rimuova i cloni e anche l'eliminazione manuale potrebbe non riuscire (BZ#2055595). Pertanto, in questo caso è consigliabile evitare la clonazione da snapshot e utilizzare invece il comando `cloneStrategy: copy`.

Per abilitare la clonazione da snapshot di CSI, è necessario modificare il profilo di storage di OpenShift Virtualization:

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

Dobbiamo inoltre aggiungere questa specifica:

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

Iniziamo importando un'immagine RHEL QCOW da uno dei nostri host in un volume di dati (DV):

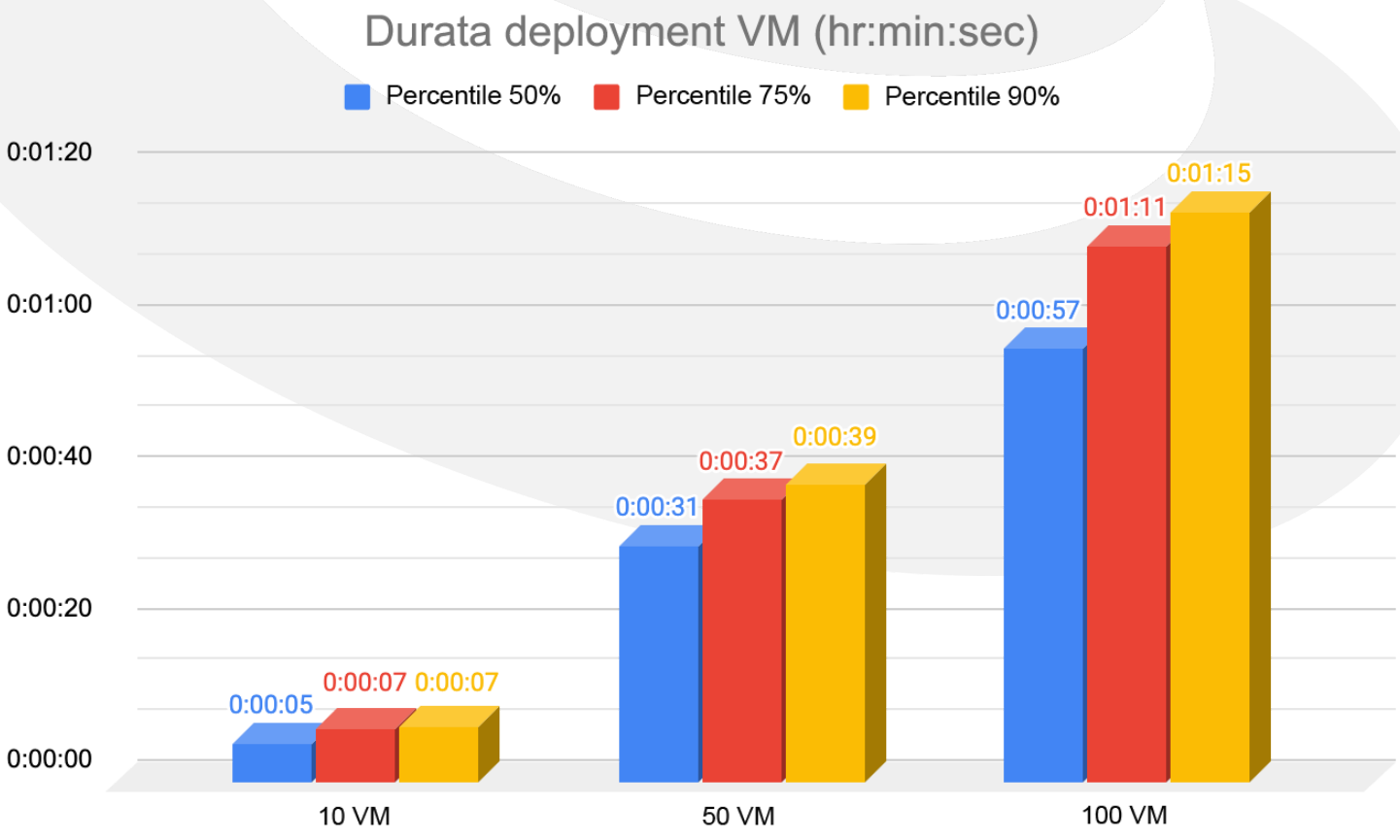
```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: rhel-clone-dv
spec:
  source:
      http:
         url: http://internel.server.com/ISO/rhel8.qcow2
  pvc:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 40Gi
    volumeMode: Block
    storageClassName: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

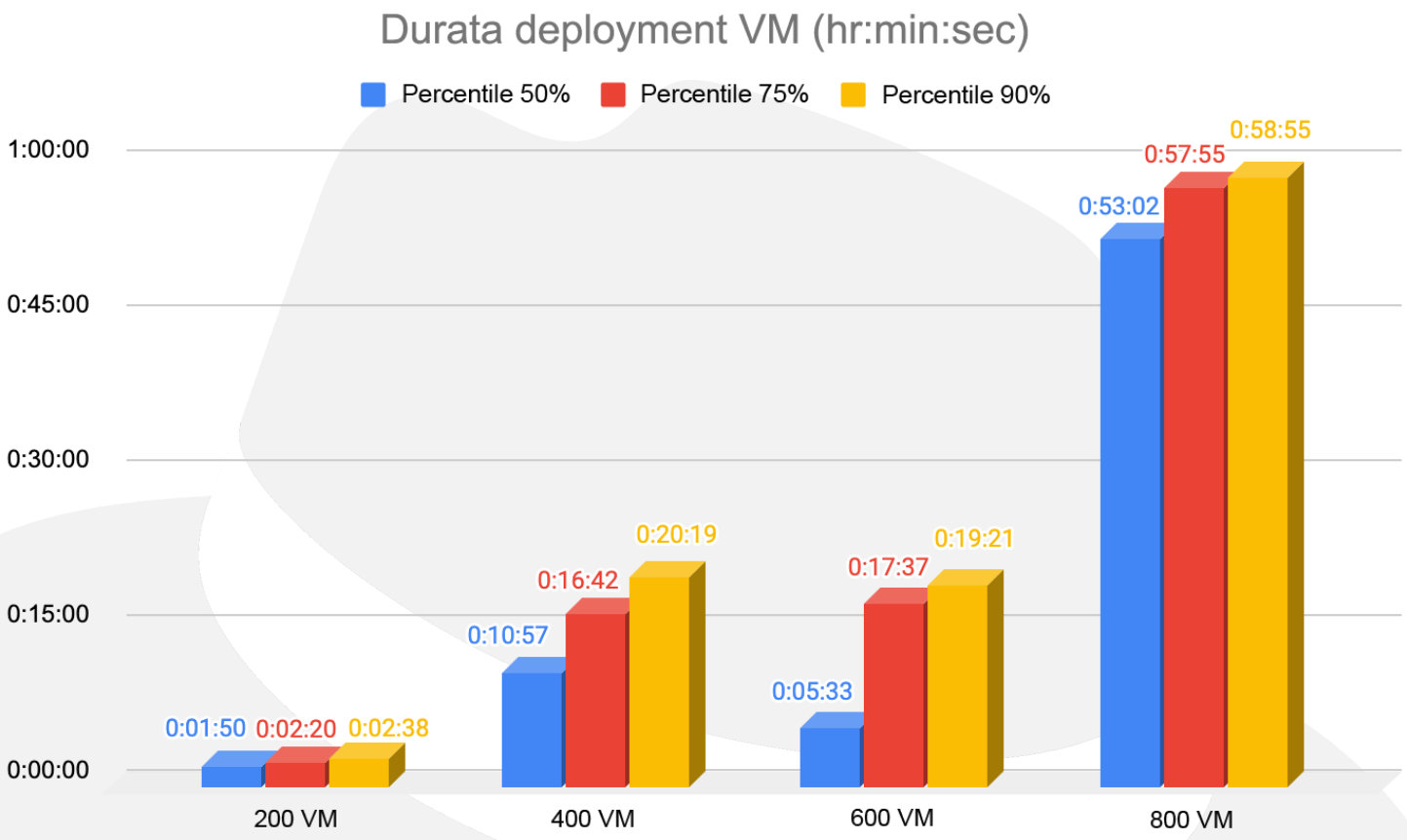
Al termine dell'importazione, distribuiamo il numero desiderato di VM in parallelo e calcoliamo il tempo necessario per completare la clonazione di ciascuna macchina. Per farlo, eseguiamo delle query a ogni VM a intervalli di 2 secondi fino al completamento della clonazione.

Il metodo più efficace per il deployment delle VM è a gruppi di 100, ovvero distribuire 100 macchine virtuali, attendere il completamento della clonazione e distribuire le 100 macchine successive.

In generale, il deployment parallelo di più di 10 VM causa una penalità dovuta all'acquisizione di un blocco sul vol-id dell'immagine principale a livello di Ceph CSI; pertanto, la clonazione da tale immagine non è più parallela ma seriale, e il provisioner esterno può inviare solo 10 chiamate parallele gRPC al driver CSI in qualsiasi momento.

Inoltre, una volta superata la soglia dei 250 cloni, le immagini rbd inizieranno ad appiattirsi, aumentando ulteriormente la penalità della clonazione. Il tempo necessario per appiattire un clone aumenta con le dimensioni dello snapshot.





Boot storm delle VM

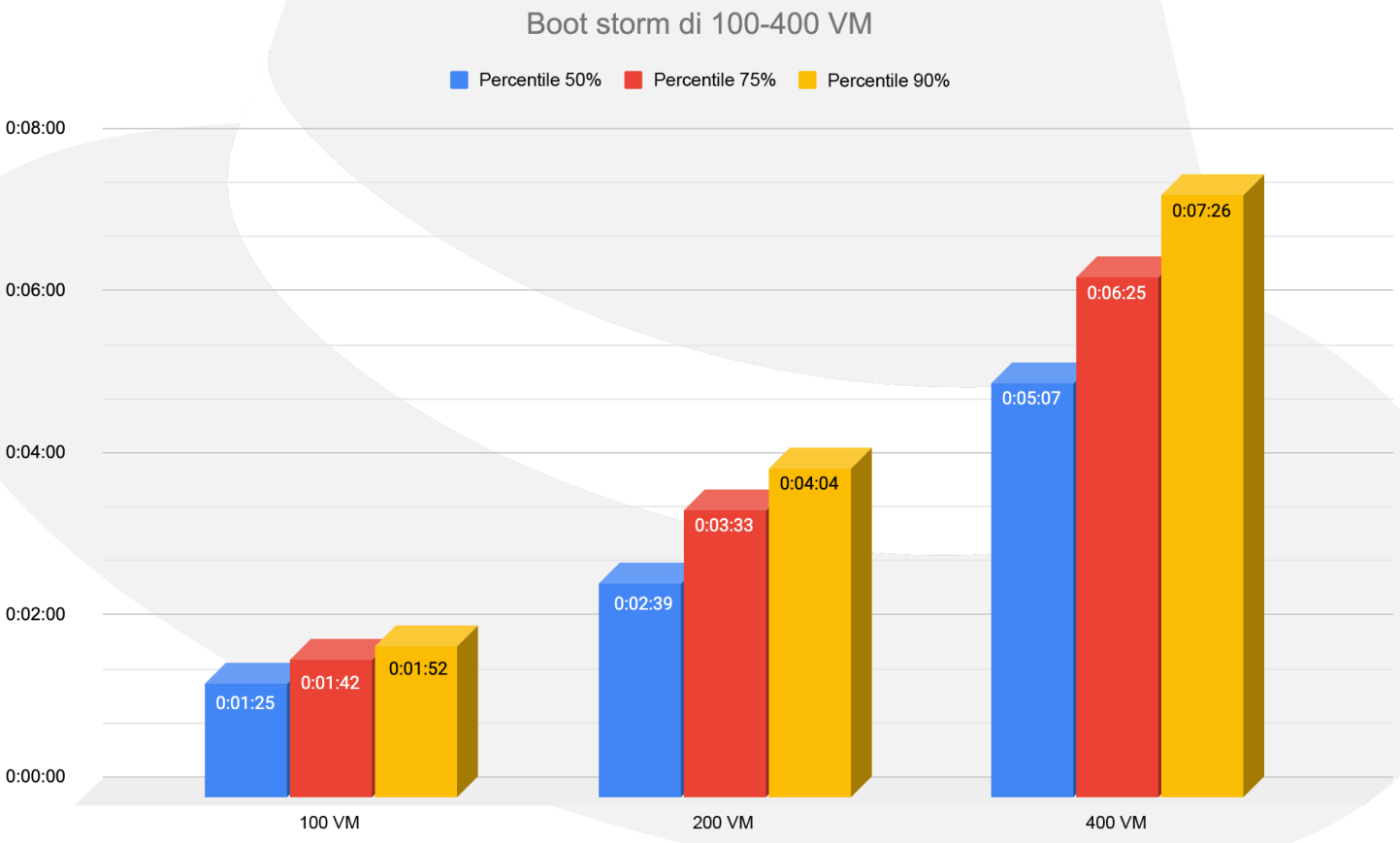
In questo scenario, abbiamo testato il tempo necessario per l'avvio di un gran numero di VM, al fine di dimostrare la resilienza di OpenShift Virtualization e del piano di controllo. È una situazione che si verifica spesso durante il ripristino di emergenza di un ambiente, ad esempio a seguito di un'interruzione di corrente.

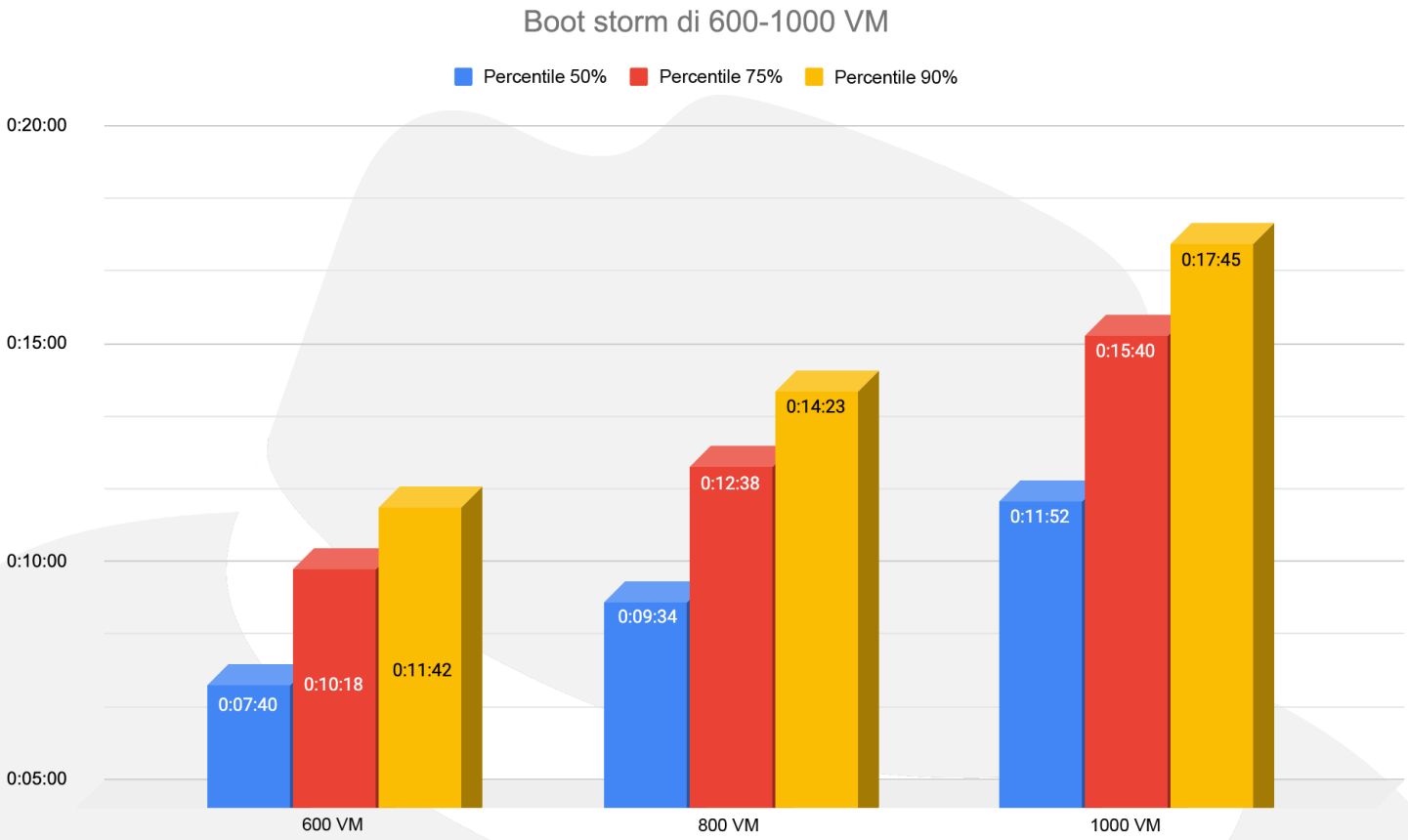
La misurazione viene effettuata per ogni VM a partire dal momento della richiesta e si interrompe una volta che la macchina è in esecuzione e accessibile tramite accesso SSH (cioè l'host si è avviato correttamente e il daemon SSH è attivo). Abbiamo misurato i tempi

eseguendo una query a ciascuna VM. Quando lo stato diventa "In esecuzione", tenta di accedere alla VM tramite SSH ogni 2 secondi, finché la connessione SSH non riesce.

Come nello scenario di deployment, tutte le richieste di avvio delle VM vengono eseguite in parallelo per mettere sotto stress tutti i componenti del cluster.

Come si può notare dai grafici sottostanti, con il nostro cluster OCP omogeneo i tempi di avvio sono quasi lineari fino a 1000 VM, ma le code più lunghe potrebbero comportare tempi di avvio più lenti.




Boot storm di 600-1000 VM
Percentile 50%
Percentile 75%
Percentile 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

Latenza delle VM

Ciascuna VM possiede il proprio thread I/O per l'elaborazione di I/O, a meno che non vi siano richieste multiple di volumi permanenti (Persistent Volume Claim, PVC) e che `dedicatedIOThread:` sia impostato su true. In tal caso, ogni PVC avrà il suo thread I/O.

Nel seguente scenario, abbiamo utilizzato 15 VM per nodo di lavoro e abbiamo testato fino a 64 nodi di lavoro, o 960 VM, per dimostrare che la presenza di più thread che accedono contemporaneamente al cluster RHCS non causa di per sé alcuna penalità della latenza.

A tal fine, abbiamo scelto blocchi da 4 KB sia per le letture che per le scritture casuali e abbiamo eseguito questi test:

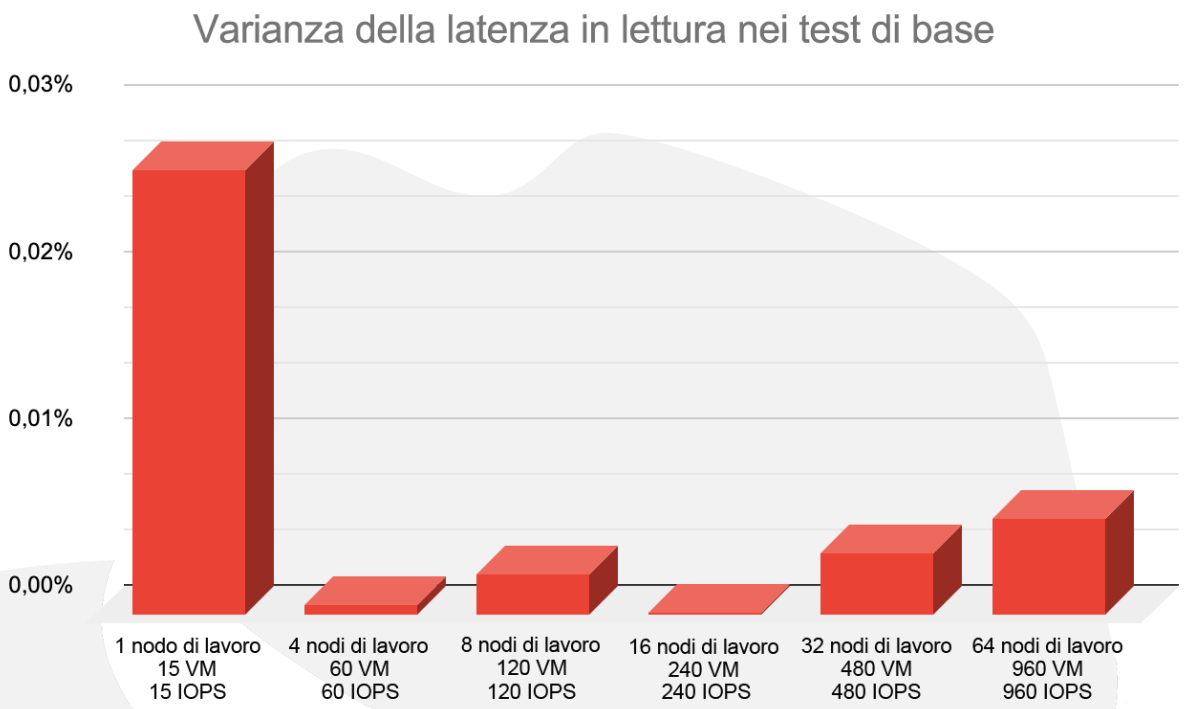
- Base: abbiamo utilizzato 15 VM da ciascun nodo di lavoro; ogni macchina genera un singolo IOPS a partire da 15 VM (15 IOPS, nodo singolo) e fino a 64 nodi per un totale di 960 VM (960 IOPS).
- Carico di lavoro: abbiamo utilizzato esattamente 15 VM da ciascun nodo di lavoro; ogni macchina genera 1000 IOPS a partire da 15 VM (15.000 IOPS, nodo singolo) e fino a 64 nodi per un totale di 960 VM (960.000 IOPS).

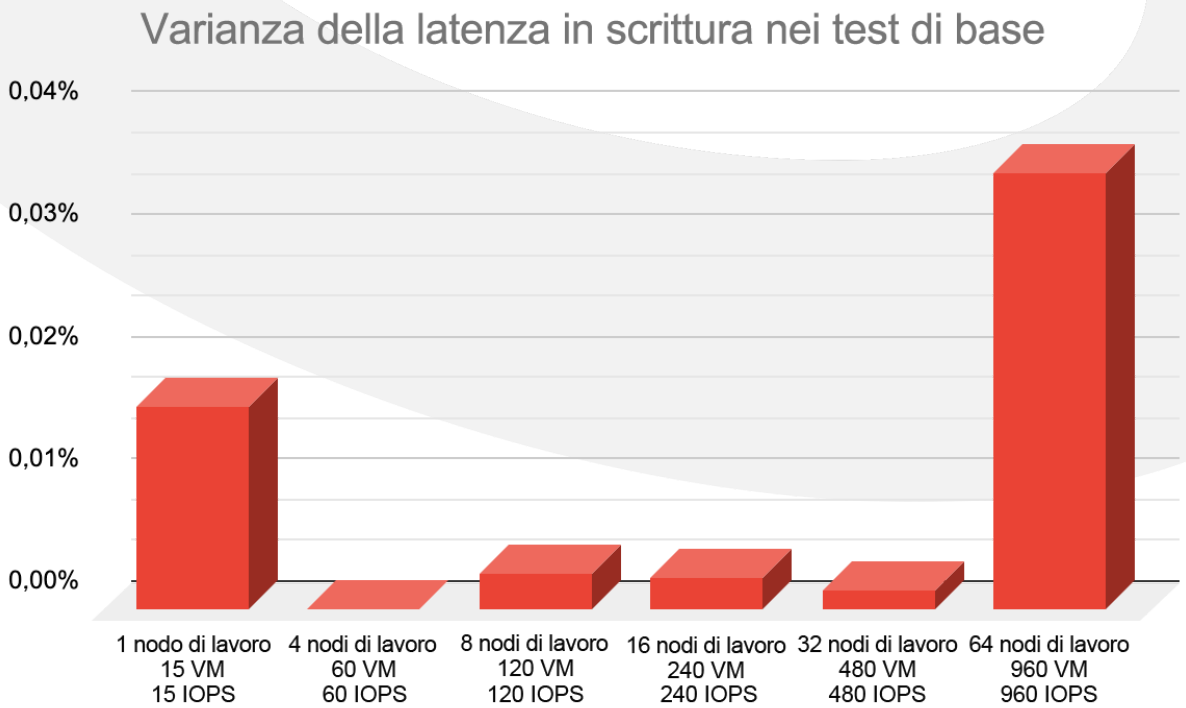
Il set di dati del file system di ogni VM è composto da 300 directory, ognuna di esse contenente 8 file da 20 MiB ciascuno; più semplicemente, ogni VM ha un set di dati di 4,8 GiB.

Abbiamo scelto un blocco di piccole dimensioni per evitare, per quanto possibile, qualsiasi incoerenza che potrebbe verificarsi a causa del networking e dei dischi Ceph non omogenei sul cluster RHCS.

Sebbene nei nostri test abbiamo utilizzato un massimo di 960 VM, in totale erano in esecuzione sul cluster 3000 VM insieme a 21.400 pod.

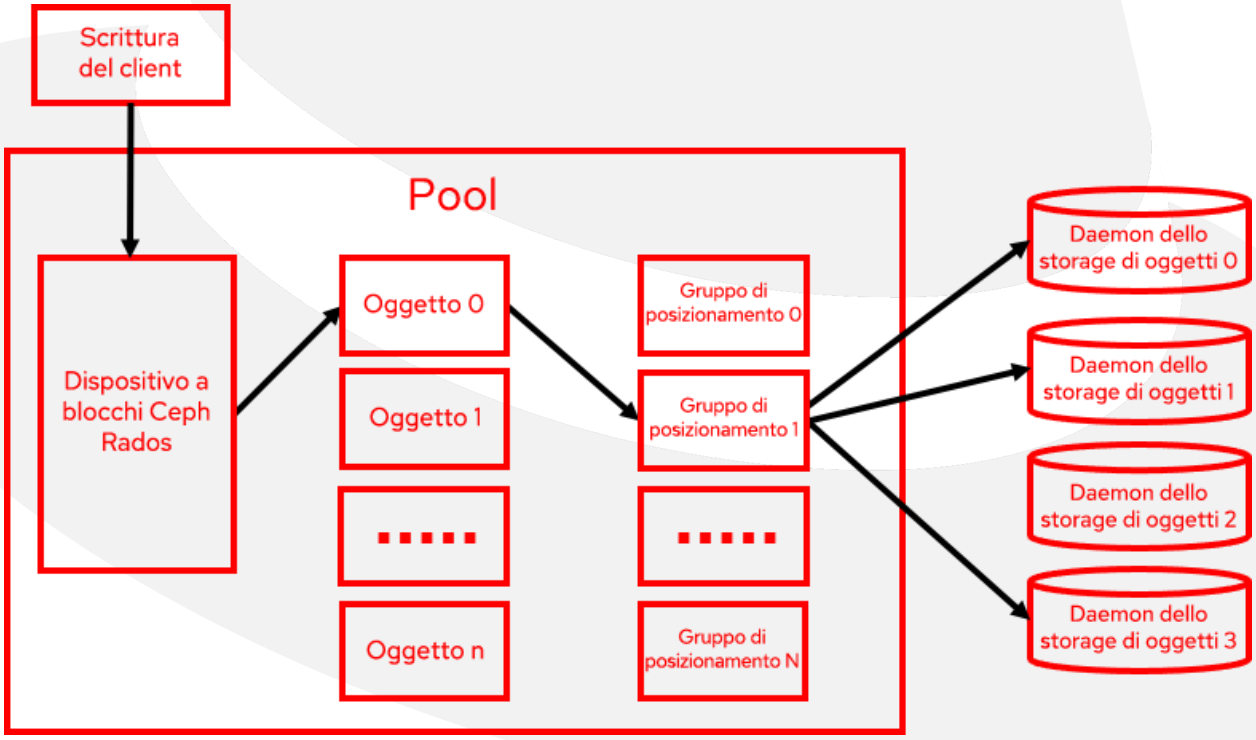
Come illustrato in entrambi i grafici sottostanti, durante l'esecuzione dei test di base la varianza della latenza sia per le letture che per le scritture casuali era inferiore allo 0,04%.


Varianza della latenza in lettura nei test di base
0,03%
0,02%
0,01%
0,00%
1 nodo di lavoro 15 VM 15 IOPS
4 nodi di lavoro 60 VM 60 IOPS
8 nodi di lavoro 120 VM 120 IOPS
16 nodi di lavoro 240 VM 240 IOPS
32 nodi di lavoro 480 VM 480 IOPS
64 nodi di lavoro 960 VM 960 IOPS

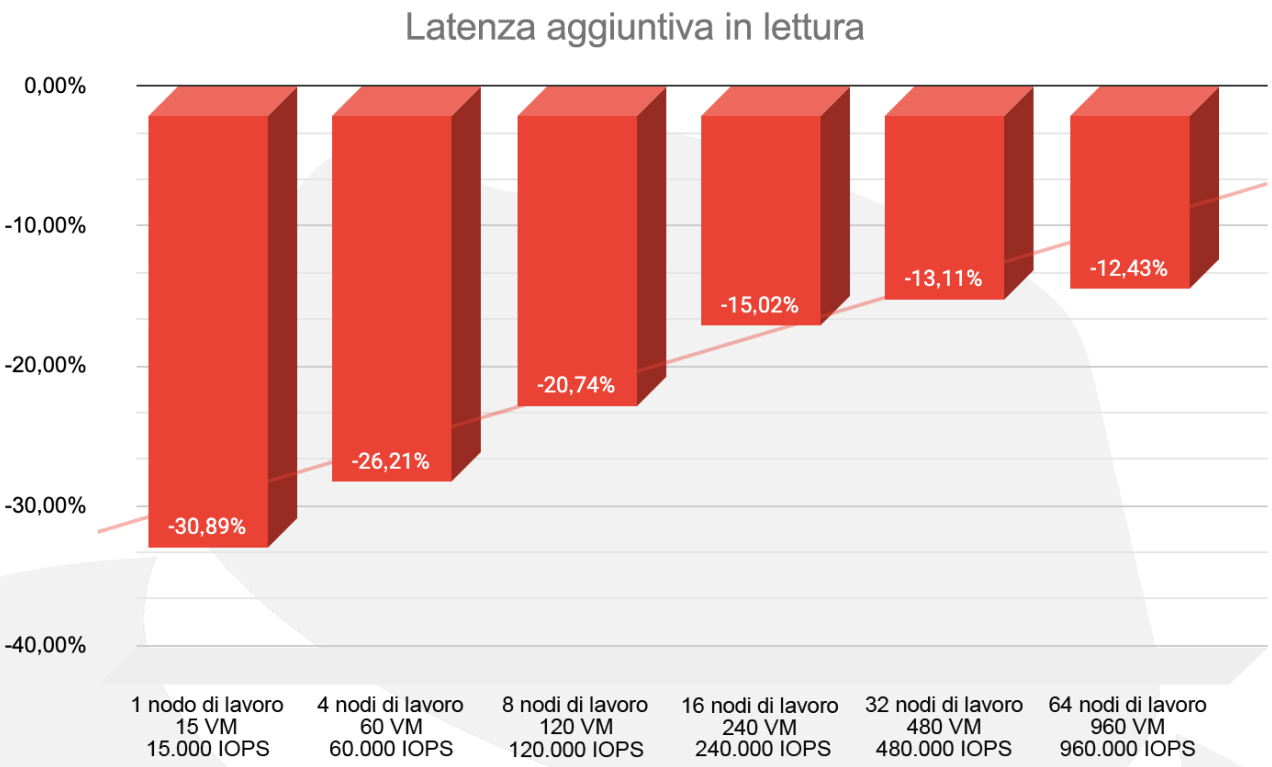

Varianza della latenza in scrittura nei test di base
0,04%
0,03%
0,02%
0,01%
0,00%
1 nodo di lavoro 15 VM 15 IOPS
4 nodi di lavoro 60 VM 60 IOPS
8 nodi di lavoro 120 VM 120 IOPS
16 nodi di lavoro 240 VM 240 IOPS
32 nodi di lavoro 480 VM 480 IOPS
64 nodi di lavoro 960 VM 960 IOPS

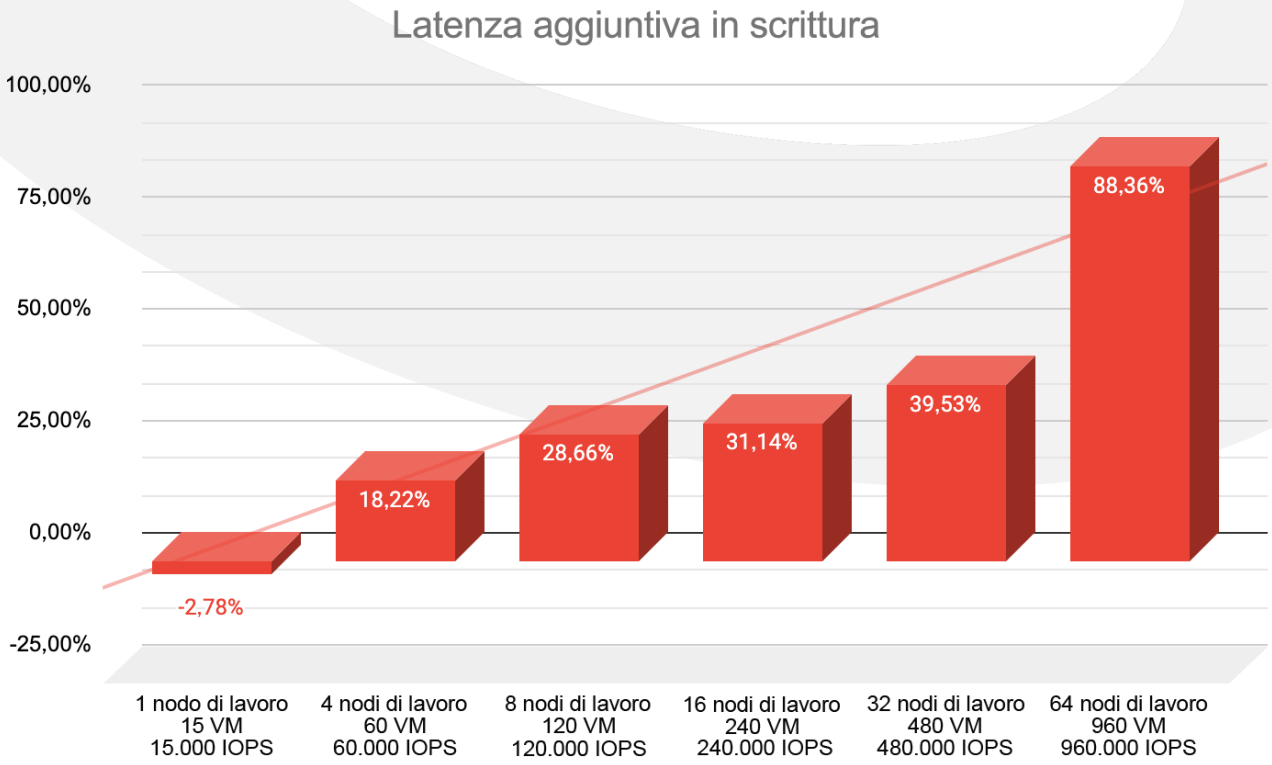
Il grafico seguente mostra l'andamento della latenza nello scenario del carico di lavoro rispetto ai risultati di base (minore significa migliore). Per quanto riguarda le prestazioni di lettura, un tasso IOPS più elevato produce una latenza più bassa in una certa misura, a causa dell'allocazione delle risorse che si verifica a valori di bursting più elevati rispetto al carico di lavoro IOPS inattivo/basso sulla VM.

Tuttavia, le scritture presentano un flusso di dati diverso, necessario per mantenere l'alta disponibilità. Come mostrato nel diagramma, per ogni scrittura generata da Ceph 3 copie dei dati attraversano la rete per raggiungere i rispettivi OSD. Pertanto, la latenza in scrittura ne risente.



Nei casi di applicazioni sensibili alla latenza, il sovraccarico della latenza in scrittura può essere ridotto di un terzo per ogni replica di dati ridotta.


Latenza aggiuntiva in lettura
0,00%
-10,00%
-20,00%
-30,00%
-40,00%
-30,89%
-26,21%
-20,74%
-15,02%
-13,11%
-12,43%
1 nodo di lavoro
15 VM
15.000 IOPS
4 nodi di lavoro
60 VM
60.000 IOPS
8 nodi di lavoro
120 VM
120.000 IOPS
16 nodi di lavoro
240 VM
240.000 IOPS
32 nodi di lavoro
480 VM
480.000 IOPS
64 nodi di lavoro
960 VM
960.000 IOPS


Latenza aggiuntiva in scrittura
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 di lavoro
15 VM
15.000 IOPS
4 nodi di lavoro
60 VM
60.000 IOPS
8 nodi di lavoro
120 VM
120.000 IOPS
16 nodi di lavoro
240 VM
240.000 IOPS
32 nodi di lavoro
480 VM
480.000 IOPS
64 nodi di lavoro
960 VM
960.000 IOPS

Migrazione delle VM

Nel seguente scenario, abbiamo testato una migrazione di 1000 VM. Per simulare una migrazione realistica, non ci siamo limitati a migrare le macchine, ma abbiamo anche riavviato i nodi di lavoro su cui risiedono le VM. Per farlo, abbiamo suddiviso i nodi in 3 zone e applicato una machine config vuota a una zona specifica, il che ha comportato il riavvio di tutti i nodi ad essa associati una volta espulse le macchine virtuali che risiedevano sui nodi di lavoro.

Per prima cosa, abbiamo applicato un'etichetta da ogni nodo di lavoro a una zona specifica:

```
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=""
```

Quindi, abbiamo creato un pool di machine config per ogni zona. Il parametro `maxUnavailable: 10` imposta il numero di nodi che possono essere disattivati in qualsiasi momento del ciclo di vita della 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
```

Abbiamo anche modificato l'operatore hyperconverged-cluster:

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

E inserito le seguenti impostazioni di migrazione per aumentare il numero di migrazioni parallele:

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

Dai nostri test, abbiamo scoperto che è consigliabile aumentare il numero di pod virt-api fino a un rapporto di **1 pod kubevirt-api ogni 750 VM**. Dal momento che in questa configurazione erano in esecuzione 3000 macchine virtuali, abbiamo portato il numero di pod API kubevirt a 4.

La funzionalità di scalabilità automatica per questo scenario è già in lavorazione e può essere monitorata su Github#7101. Al momento è possibile procedere manualmente applicando una patch all'operatore 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}]\"}]}]"}}}'
```

Per attivare la migrazione, abbiamo creato questa machine config:

```
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: ""
```

La migrazione di 1000 VM comprendeva:

- 400 VM con RHEL.
- 400 VM con Fedora.
- 200 VM con Windows.

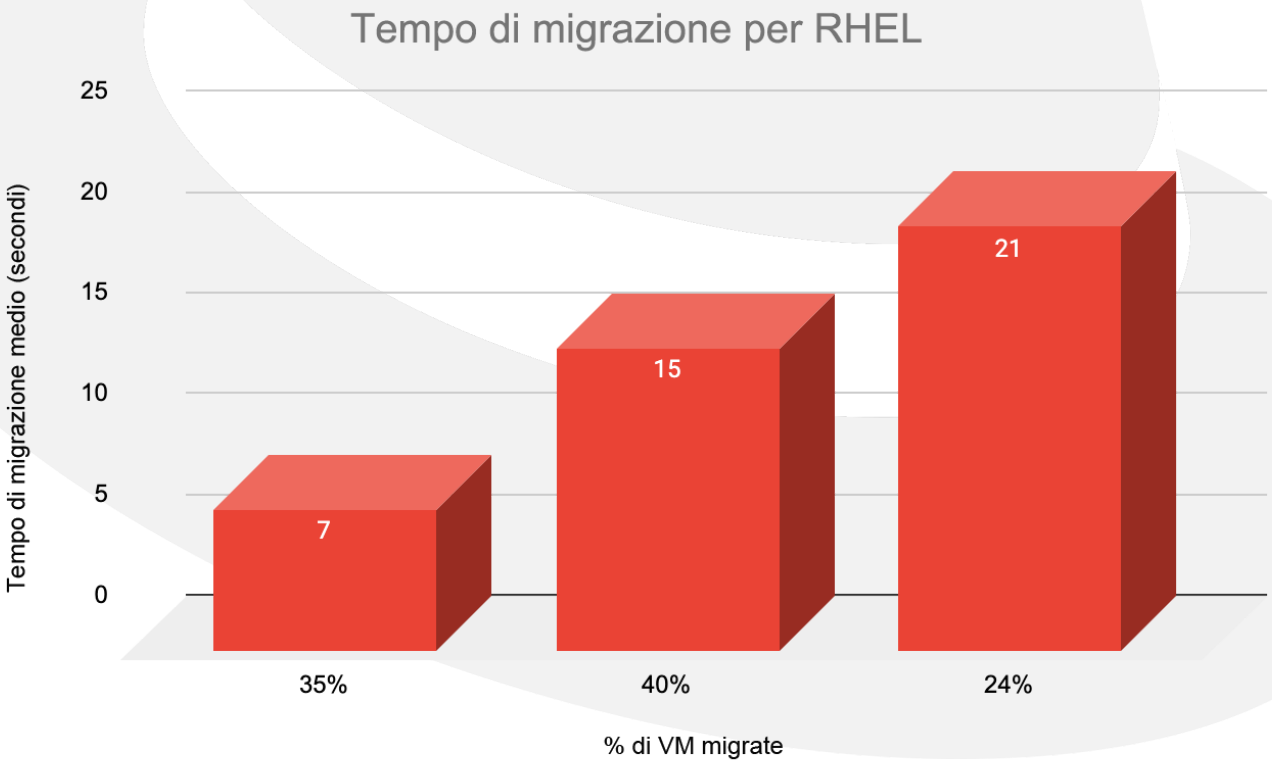
Anche i 7000 pod presenti nella stessa posizione delle macchine virtuali sono stati riavviati su nodi di lavoro diversi.

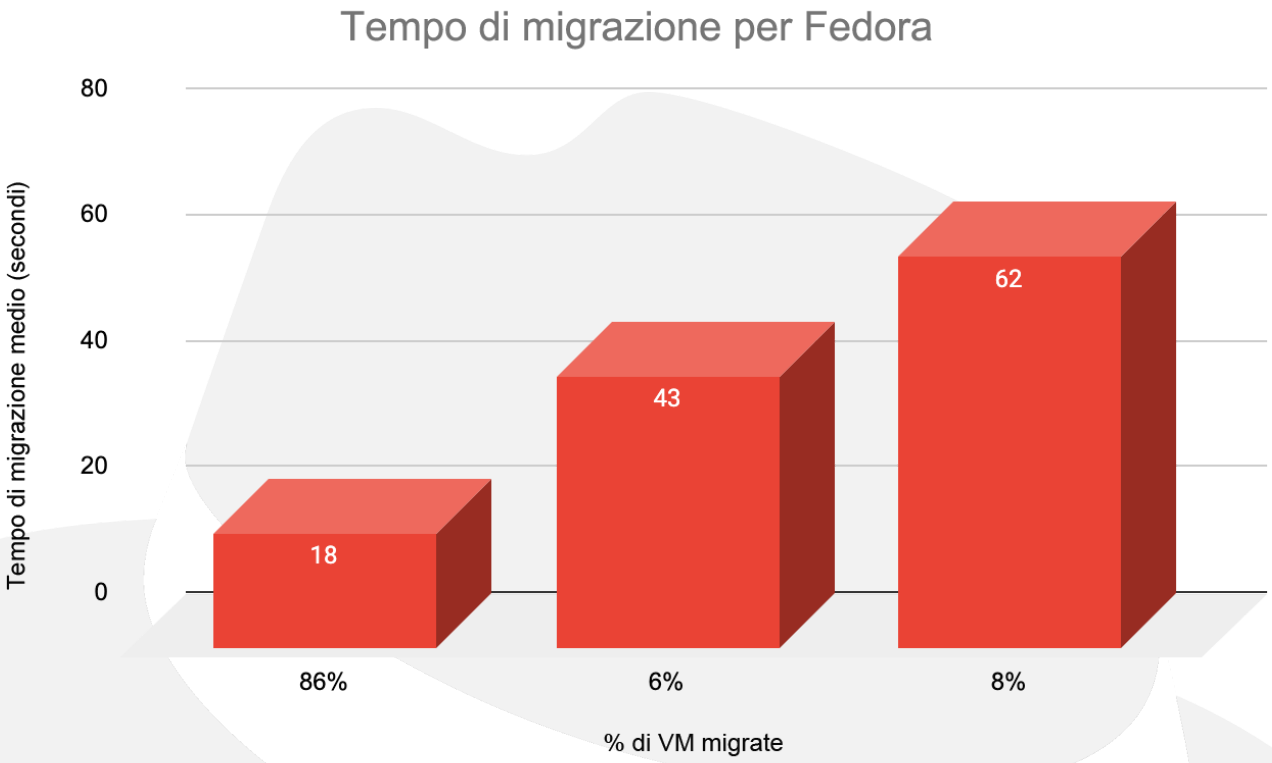
È possibile visualizzare il tempo necessario al completamento della migrazione di ciascuna VM nei log VMI:

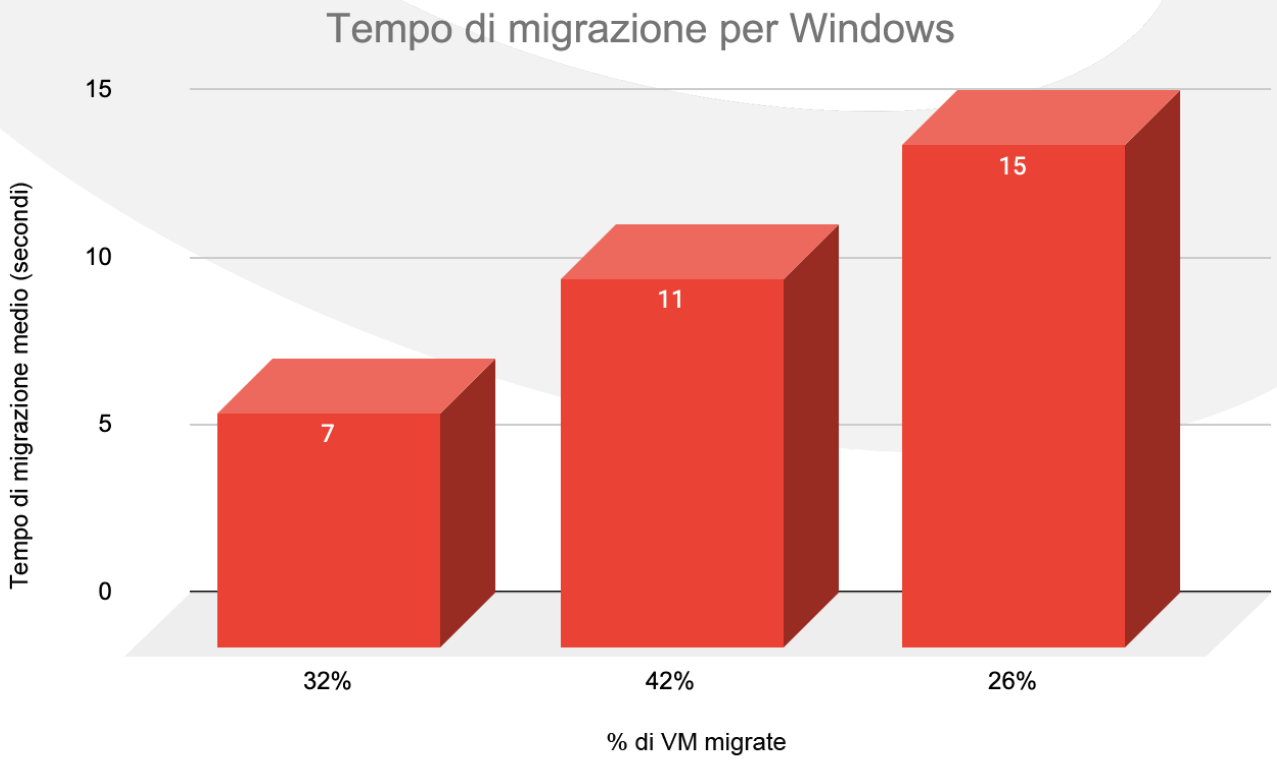
```
Phase Transition Timestamps:
  Phase:                        Scheduling
  Phase Transition Timestamp:   2022-04-10T07:15:13Z
  Phase:                        Scheduled
  Phase Transition Timestamp:   2022-04-10T07:15:23Z
  Phase:                        Running
  Phase Transition Timestamp:   2022-04-10T07:15:25Z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

I grafici seguenti mostrano i tempi di migrazione di ciascun SO distribuiti in base alla percentuale. Ad esempio, per quanto riguarda la migrazione delle VM con RHEL, il 35% delle macchine ha completato la migrazione in un tempo medio di 7 secondi.

Il tempo di migrazione non è necessariamente legato al SO, ma piuttosto al carico guest attuale, al carico host, al carico della rete, alla tecnologia di storage, alla policy di migrazione, alle dimensioni dell'immagine, ecc. Pertanto, i risultati possono variare.




Tempo di migrazione per Fedora
Tempo di migrazione medio (secondi)
80
60
40
20
0
18
43
62
86%
6%
8%
% di VM migrate


Tempo di migrazione per Windows
Tempo di migrazione medio (secondi)
15
10
5
0
7
11
15
32%
42%
26%
% di VM migrate

Tempo di migrazione medio per SO:

SO	Tempo di migrazione medio (sec.)	Commenti
RHEL	14	PVC 40 GiB
Fedora	23	Disco di container evictionStrategy: Restart
Windows	12	PVC 40 GiB

In conclusione, il completamento della migrazione ha richiesto un tempo totale di 118 minuti più ulteriori 35 minuti di attesa affinché i nodi raggiungessero lo stato "Pronto" per via del parametro `maxUnavailable: 10` menzionato in precedenza.

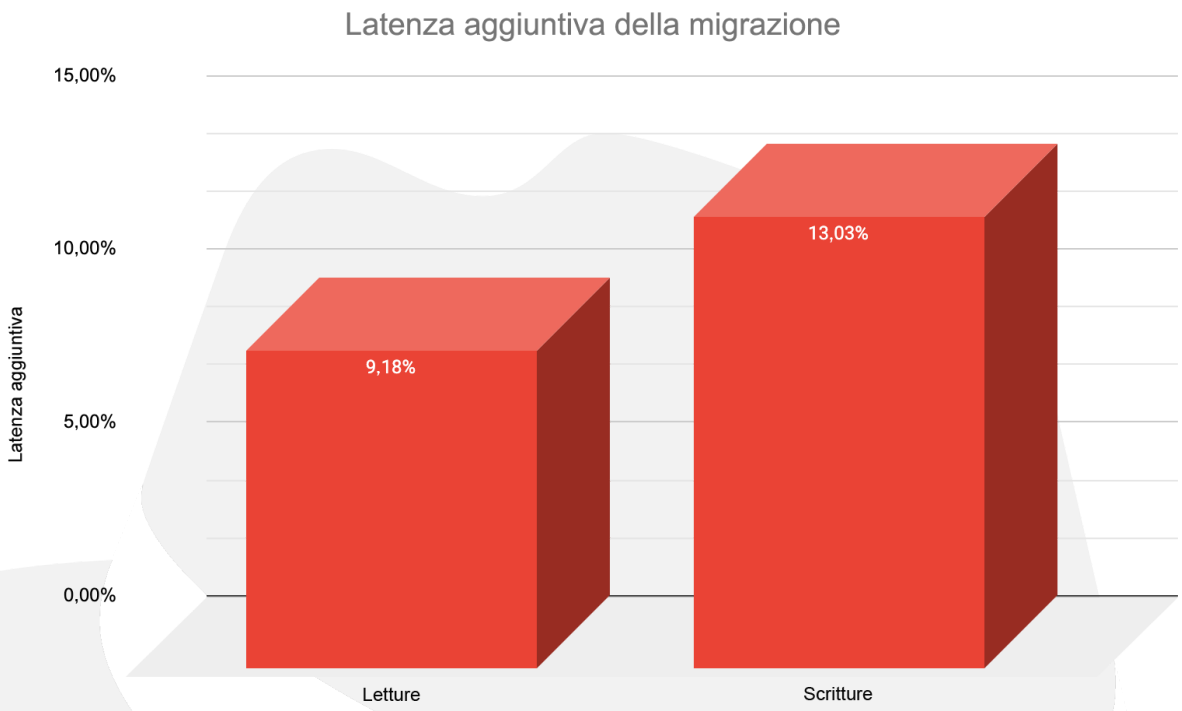
Latenza aggiuntiva della migrazione delle VM

Nel seguente scenario, abbiamo testato una migrazione di 1000 VM, ma in questo caso abbiamo utilizzato soltanto macchine con RHEL. Abbiamo optato per questa soluzione per evitare qualsiasi discrepanza che potrebbe verificarsi a causa del modo in cui i diversi sistemi operativi gestiscono I/O.

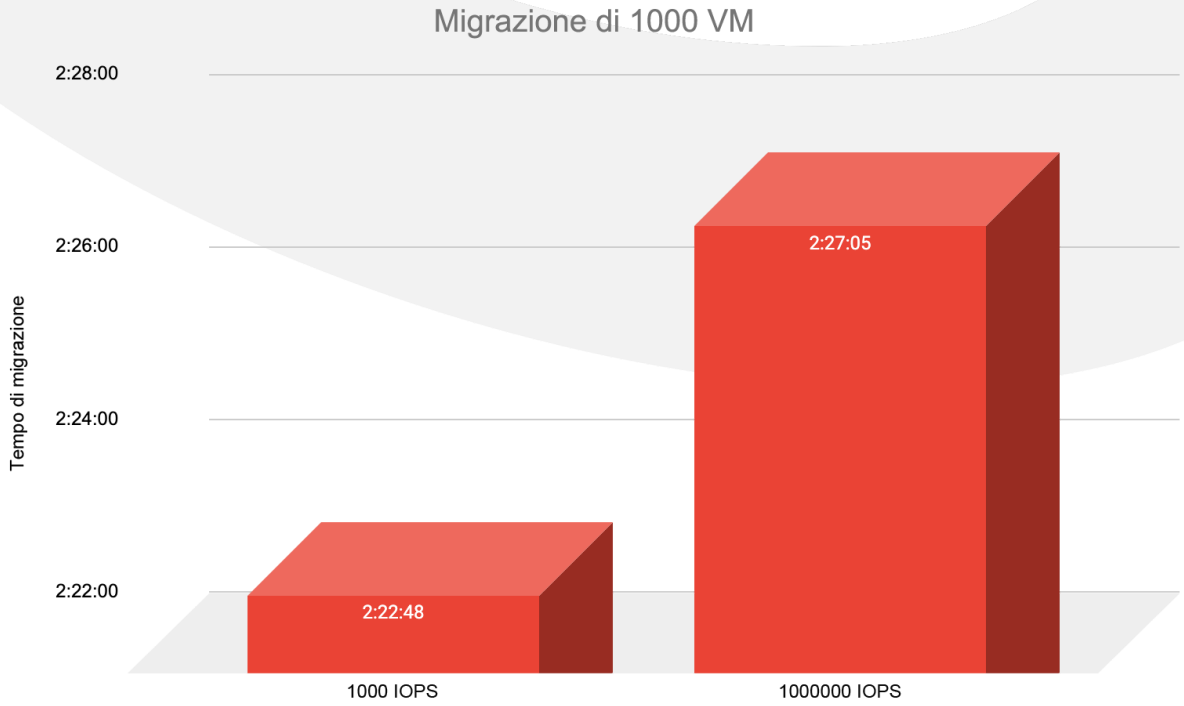
Come in precedenza, abbiamo scelto blocchi di 4 KB sia per le letture che per le scritture casuali e abbiamo eseguito questi test:

- Base: ciascuna delle 1000 VM genera un singolo IOPS per un totale di 1000 IOPS.
- Carico di lavoro: ciascuna delle 1000 macchine virtuali genera 1000 IOPS al secondo per un totale di un milione di IOPS.

Il grafico seguente mostra la penalità media di latenza durante la migrazione sia in lettura che in scrittura rispetto al valore di base (solo VM con RHEL):



Inoltre, come si può vedere nel grafico sottostante, il tempo di migrazione ha subito una variazione del 5%. I risultati relativi al tempo di migrazione possono variare a causa di [BZ#2069098]:



Red Hat

Upgrade del cluster su larga scala

Nel seguente scenario, abbiamo testato sia upgrade secondari sia principali.

Siamo partiti con l'upgrade del cluster dalla versione 4.9.15 alla 4.9.23 e simulando un vero upgrade di produzione; ciò significa che tutte le 3000 VM e i 21.400 pod erano in esecuzione sul cluster. Inoltre, abbiamo generato un carico di lavoro leggero di 4 KB su 1500 VM a una velocità di 100 IOPS per VM.

Abbiamo avviato l'upgrade eseguendo:

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

È possibile monitorare l'avanzamento dell'upgrade con:

```
$ 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)
```

La durata totale del processo di upgrade secondario è stata di **35 minuti**.

Il passaggio successivo è stato testare un upgrade principale aggiornando il cluster dalla versione 4.9.23 alla 4.10.9 ed eseguendo nuovamente l'upgrade alle stesse condizioni. Abbiamo avviato l'upgrade eseguendo:

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

Anche in questo caso, è possibile monitorare l'avanzamento dell'upgrade con:

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

La durata totale del processo di upgrade principale è stata di **136 minuti**.

Alcuni upgrade potrebbero includere modifiche che richiedono il soft reset di tutti i nodi, con conseguente aumento significativo della durata dell'upgrade a causa del tempo di migrazione aggiuntivo.

Conclusioni

In questa architettura di riferimento abbiamo dimostrato le funzionalità e la resilienza OpenShift Virtualization su larga scala. La funzionalità OpenShift Virtualization di Red Hat OpenShift Container Platform, insieme a Red Hat Ceph Storage e/o Red Hat OpenShift Data Foundation, offre una soluzione di produzione completa che integra container, macchine virtuali e storage ad alta disponibilità e che può essere distribuita su qualsiasi host che soddisfi i requisiti hardware minimi.

Sebbene questa architettura di riferimento illustri come raggiungere l'obiettivo prefissato, è importante valutare anche altre architetture per raggiungere la resilienza, la scalabilità e l'uniformità delle operazioni quotidiane in base a determinate condizioni e ai requisiti ambientali. Ad esempio, dopo un certo limite, si dovrebbe prendere in considerazione un approccio multicluster quando il numero di nodi o il carico di lavoro diventa troppo elevato o si verifica un'eccessiva perdita sul cluster.

Ulteriori risorse

Requisiti di ambiente e sistema:
https://docs.openshift.com/container-platform/3.11/install/prerequisites.html

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

Machine Config Operator:
https://docs.openshift.com/container-platform/4.9/post_installation_configuration/machine-configuration-tasks.html

Migrazione in tempo reale e timeout:
https://docs.openshift.com/container-platform/4.9/virt/live_migration/virt-live-migration-limits.html

Aggiornamento del cluster con la CLI:
https://docs.openshift.com/container-platform/4.10/updating/updating-cluster-cli.html

Informazioni su Red Hat

Red Hat è leader mondiale nella fornitura di soluzioni software enterprise open source. Con un approccio basato sul concetto di community, distribuisce tecnologie come Kubernetes, container, Linux e cloud ibrido caratterizzate da affidabilità e prestazioni elevate. Red Hat consente di sviluppare applicazioni cloud native, integrare applicazioni IT nuove ed esistenti, e automatizzare e gestire ambienti complessi. Considerata un partner affidabile dalle aziende della classifica Fortune 500, Red Hat fornisce pluripremiati servizi di consulenza, formazione e assistenza, che portano i vantaggi dell'innovazione open source in qualsiasi settore. Red Hat è l'elemento catalizzatore in una rete globale di aziende, partner e community, e permette alle organizzazioni di crescere, evolversi e prepararsi a un futuro digitale.