OpenShift Virtualization

Red Hat Ceph Storage 5 外部ストレージを使用

大規模なチューニングとパフォーマンス

リファレンスアーキテクチャ

Boaz Ben Shabat (ボアズ・ベン・シャバト)

目次

目次 1

対象者 3

エグゼクティブサマリー 3

ソフトウェア・コンポーネント 4

物理コンポーネント 6

- RHCS クラスタ 6
- OpenShift クラスタ 7

アーキテクチャ 8

RHCS ネットワークチューニング 9

- アドレス解決プロトコル (ARP) のチューニング 9
  - ARP フラックス 9
  - ARP キャッシュ 10
- TCP/IP チューニング 10
  - TCP ウィンドウスケーリング 10
  - バッファサイズのチューニング 11
    - レイテンシー方式 11
    - パケットサイズ方式 12
- アダプターのチューニング 12
  - NIC バッファ 12
  - バックログキュー 13

RHCS のチューニング 14

- 配置グループ (PG) のチューニング 14
- Prometheus のチューニング 15

OpenShift Virtualization 16

- はじめに 16
- KubeletConfig 16
- テンプレート 18
  - Red Hat Linux 18
  - Fedora 19

Windows 20
Pod 22
VM のデプロイメント 22
VM のブートストーム 25
VM のレイテンシー 28
VM の移行 32
VM 移行の追加レイテンシー 37
大規模なクラスタアップグレード 39
まとめ 40
その他の資料 40

対象者

この資料の目的は、お客様、セールスエンジニア、フィールドコンサルタント、ソリューションアーキテクトなど、インフラストラクチャ・サービスの担当者を支援することです。
ここでは、OpenShift Virtualization の大規模デプロイメントの成功例をご紹介します。OpenShift Virtualization は Red Hat OpenShift® Container Platform の一機能であり、高可用性 (HA) 外部ネットワーク・ストレージ・ソリューションとして RHCS を使用します。

エグゼクティブサマリー

この資料では、外部の Red Hat® Ceph® Storage 5 (RHCS) クラスタ (47 ノード) と、合計 3,000 台の VM と 21,400 個の Pod を収容する OpenShift Virtualization 仮想マシン (VM) にストレージを提供する外部の Ceph クラスタを使用する Red Hat OpenShift Virtualization (100 ノード) の両方を組み込んだ大規模デプロイメントの成功から、Red Hat OpenShift Virtualization Performance および Scale チームが学んだことについて説明します。

このリファレンスアーキテクチャでは、RHCS と Red Hat OpenShift Virtualization をチューニングし、回復力のある 100 ノードの OpenShift クラスタを作成するために実際に行った手順を紹介します。

また、これらの手順の根拠となる理由を説明し、その推奨事項をあらゆるクラスタに適用できるようにするための情報も提供します。

以下の表は、プロダクション環境で発生する可能性のある最も重要なシナリオのパフォーマンス結果を示しています。

シナリオ	説明	結果
VM のデプロイメント	最大 800 台の VM の並列デプロイメント	テスト結果から、100 台の VM を一括してクローニングすると、最速のデプロイ時間が達成できることがわかりました。
VM のブートストーム	最大 1,000 台の VM の並列ブートストーム	起動時間はほぼ線形で、最短は 100 VM の場合の 01:42 (MM:SS)、最長は 1,000 VM の場合の 17:45 でした。

VM のレイテンシー	読み取りと書き込みの両方のワークロードレイテンシーと比較した持続的なアイドルレイテンシー	アイドル状態の VM のレイテンシーは、RHCS にアクセスする IO スレッドの数による影響を受けず、1,000,000 IOPS では、読み取りレイテンシーが最大 30% 削減され、書き込みレイテンシーは最大 88% 増加しました。
VM の移行	1,000 VM の移行	1,000 台の VM の移行 + 7,000 個の Pod のエビクションには、およそ 118 分 (HH:MM) かかりました。
VM 移行の追加レイテンシー	ワークロードを伴う 1,000 台の Red Hat Enterprise Linux® (RHEL) VM の移行	移行中の IO レイテンシーは読み取りで 9%、書き込みで 13% 増加し、移行時間は 3% 増加しましたが、実際の IOPS 速度への影響はありませんでした。
OpenShift クラスタのアップグレード	OpenShift クラスタのバージョンの更新	マイナーアップグレードには 35 分、メジャーアップグレードには 136 分かかりました。

ソフトウェア・コンポーネント

製品	バージョン	説明
Red Hat OpenShift	4.9.15	エンタープライズ Kubernetes プラットフォームのリーダーであり、デプロイ先がどこであっても、クラウドのようなエクスペリエンスを実現します。クラウドでも、オンプレミスでも、ネットワークエッジでも、Red Hat OpenShift を使用すると、一貫したエクスペリエンスを通じてアプリケーションを構築、デプロイ、実行する場所を選択できます。
Red Hat Ceph Storage	5	先進的なデータパイプライン向けの、オープンでスケーラビリティに優れた、単純化されたストレージ・ソリューションです。データ分析、人工知能/機械学習 (AI/ML)、先進のワークロード向けに設計された Red Hat Ceph Storage により、お客様が選択した業界標準ハードウェア上でソフトウェア・デファインド・ストレージを利用できます。

Red Hat OpenShift Data Foundation	4.9.2	コンテナ向けのソフトウェア・デファインド・ストレージ。Red Hat OpenShift のデータおよびストレージサービス・プラットフォームとして設計された Red Hat OpenShift Data Foundation は、クラウドとベアメタルホスト全体でアプリケーションを迅速かつ効率的に開発し、デプロイするのに役立ちます。
Red Hat OpenShift Virtualization	4.9.2	Kubernetes クラスタ上で VM を実行するための Red Hat のソリューション。OpenShift Virtualization は 2 つの目標を達成するように設定されています。第 1 の目標は、すべてのユーザーがワークロードを 1 つのプラットフォームに統合できるようにすること、つまり、長期的な仮想マシンユーザーであっても初めて VM を使用するユーザーであっても、コンテナ・プラットフォームと並行して追加の仮想化プラットフォームを管理する運用オーバーヘッドを削減できるようにすることです。第 2 の目標は、Kubernetes エンジンとエコシステムの強みを利用して、ユーザーが従来のワークロード機能、オーケストレーション、アーキテクチャをモダナイズできるようにすることです。

物理コンポーネント

RHCS クラスタ

DELL PowerEdge R640 ラックサーバー (10 台):

コンポーネント	仕様	コメント
CPU	40 コア	Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz × 2
メモリ	384GB ECC RAM	SK Hynix 1x 32GB DDR4-3200 RDIMM PC4-25600R Dual Rank x4 Module × 12
SSD (root ディスク)	446.63 GB - 6 Gbps	MICRON SSD MTFDDAK480TDT
SSD (ストレージ)	3574 GB - 12 Gbps	TOSHIBA SSD KPM5XVUG1T92 1787.88 GB × 2
NVME (ストレージ)	2980.82 GB - 8 GT/s	Samsung NVME S5CXNA0N607551

DELL PowerEdge R650 ラックサーバー (37 台):

コンポーネント	仕様	コメント
CPU	56 コア	Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz × 2
メモリ	384GB ECC RAM	SK Hynix 1x 32GB DDR4-3200 RDIMM PC4-25600R Dual Rank x4 Module × 12
SSD (root ディスク)	446.63 GB - 6 Gbps	MICRON SSD MTFDDAK480TDT
SSD (ストレージ)	3574 GB - 12 Gbps	TOSHIBA SSD KPM5XVUG1T92 1787.88 GB × 2
NVME (ストレージ)	2980.82 GB - 8 GT/s	Samsung NVME S5CXNA0N607551

注：RHCS に使用されたハードウェアは、RHCS クラスタでのディスクサイズとアーキテクチャが不均一であったため、このタスクに最適なものではなく、Ceph のパフォーマンスに影響がありました。しかし、これは Ceph が提供できる多用途性を示す強力な証拠でもあります。

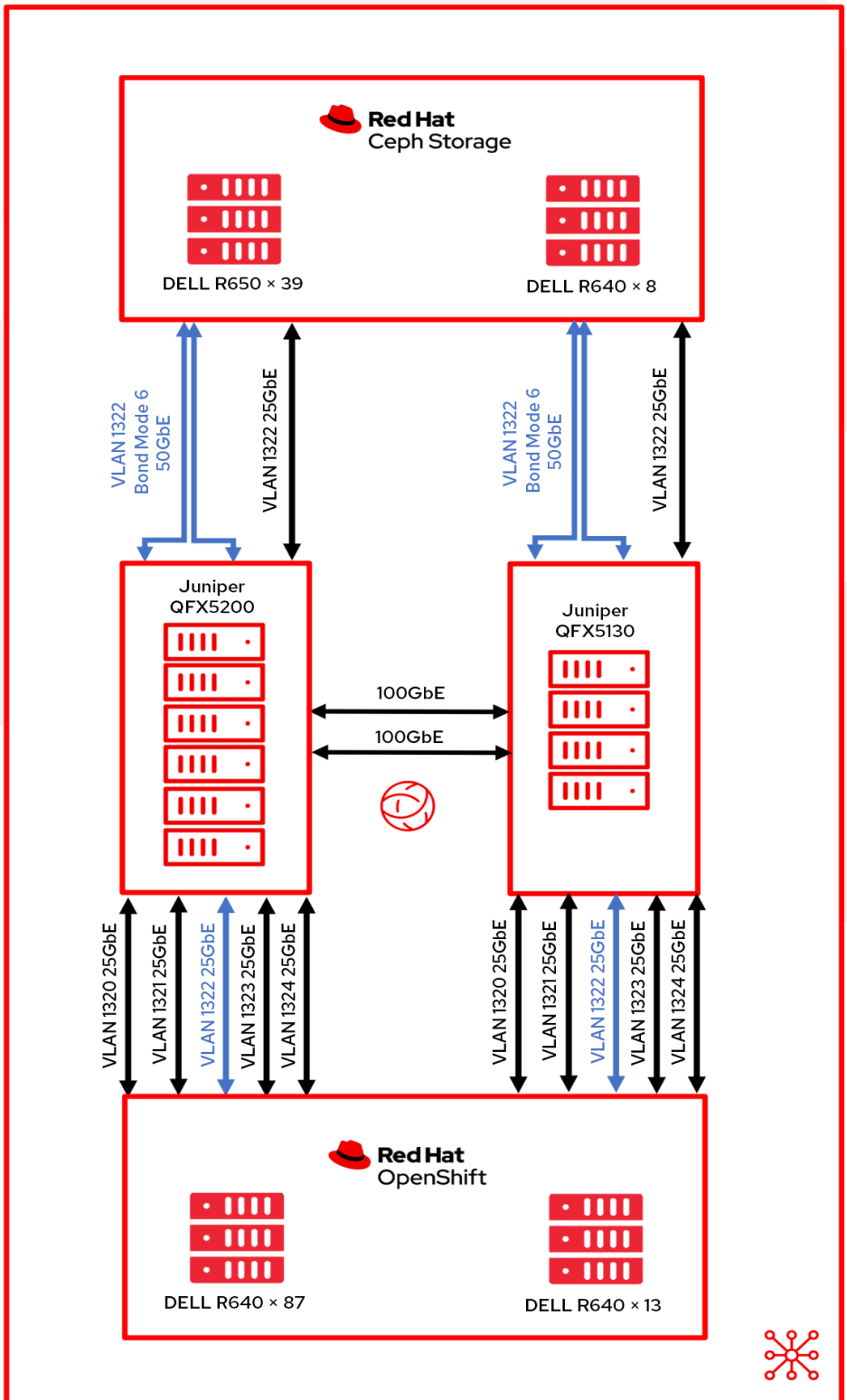
OpenShift クラスタ

DELL PowerEdge R640 ラックサーバー (100 台)：

コンポーネント	仕様	コメント
CPU	40 コア	Intel(R) Xeon(R) Gold 6230 CPU @ 2.10GHz 20 コア × 2
メモリ	384GB ECC RAM	SK Hynix 1x 32GB DDR4-3200 RDIMM PC4-25600R Dual Rank x4 Module × 12
SSD (root ディスク)	446.63 GB - 6 Gbps	MICRON SSD MTFDDAK480TDT

アーキテクチャ

以下の図は、OpenShift クラスタと Ceph クラスタのネットワーキング・アーキテクチャを示しています。Ceph とプライベートラボ VLAN 上の Red Hat OpenShift Container Platform (OCP) クラスタ間のデータパスは、2 つの 25 GbE ポートを備えた balance-alb ボンディングを使用することに注意してください。



RHCS ネットワークチューニング

このセクションでは、この大規模環境に対応するために Ceph ノードで行った Linux ネットワークチューニングについて説明します。

アドレス解決プロトコル (ARP) のチューニング

ARP フラックス

同じサブネット上に複数のネットワーク・インタフェースを持つ Linux ホストは、ARP フラックスの問題による影響を受ける可能性があります。ARP フラックスの問題は、ホストが同じサブネット上のインタフェースに対する ARP 要求に応答するときに発生することがあります。この動作が必ずしも問題になるわけではありません。しかし、場合によっては ARP フラックスにより、IPv4 アドレスと MAC アドレス間のマッピングが正しくないために一部のアプリケーションが誤動作する可能性があります。

RHEL ベースのホストでは、すべての RHCS ホストの /etc/sysctl.d/99.8-arp.conf を編集し、以下の行を追加することでこの動作を修正できます。

```
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** - 同じサブネット上で複数のネットワーク・インタフェースを持つことを可能にします。また、カーネルがインタフェースから ARP 要求 の IP パケットをルーティングするかどうかに基づいて、各インタフェースの ARP が応答できるようにします (したがって、これを機能させるためにはソースベースのルーティングを使用する必要があります)。つまり、ARP 要求に応答するカード (通常は 1) の制御が可能となります。
- **ignore=1** - 出力先 IP アドレスが受信インタフェース上で設定されたローカルアドレスである場合にのみ応答します。
- **arp_announce=1** - このインタフェースでは、出力先のサブネットにないローカルアドレスは使用しないようにします。このインタフェースを経由してアクセス可能な出力先ホストが、ARP 要求のソース IP アドレスが受信側インタフェース上に設定されるロジカルなネットワークの一環となるよう要求した場合、このモードは役立ちます。

以下のコマンドを実行して、新しいネットワーク設定をロードします。

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

ARP キャッシュ

ARP キャッシュは、IP アドレスが MAC アドレスに解決されるときに生成される ARP エントリーのリストを保持します。ARP キャッシュにすべてのエントリーを保持できなくなるような大規模なケースを回避するには、**/etc/sysctl.d/99.7-arpcachesize.conf** を編集して以下に示す行を追加し、ARP キャッシュのサイズを増やす必要があります。

```
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
```

この数値は、IPv4 宛先キャッシュエントリーのガベージコレクションを開始するしきい値を設定します。この値が 2 倍になると、システムは新しい割り当てを拒否します。

TCP/IP チューニング

TCP ウィンドウスケーリング

RHEL のデフォルトのネットワーク設定では、大規模セットアップでよく見られる大規模な並列ジョブに対して最適なスループット/レイテンシーのパフォーマンスが得られない可能性があります。Linux ネットワークと一部のネットワークデバイスをチューニングして並列ジョブのパフォーマンスを向上させる方法を以下に示します。

高帯域幅ネットワークを有効に利用するには、より大きな TCP ウィンドウサイズを使用する必要があります。
したがって、TCP ウィンドウスケーリングが有効になっていることを確認しました。
これは **cat /proc/sys/net/ipv4/tcp_window_scaling** で確認できます。

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

次のコマンドを使用して再起動し、永続化させます。

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

バッファサイズのチューニング

次のステップは、ソケットの「送信バッファサイズ」と「受信バッファサイズ」を計算することです。一般に、各ソケットの読み取り/書き込みバッファが保持できるのは、最小 2 パケット、デフォルトの 4 パケット、または最大 10 パケットのいずれかです。ネットワーク・ソケット・バッファが小さすぎると、バッファがいっぱいになって実効スループットが低下し、パフォーマンスに影響します。ネットワーク・ソケット・バッファが十分な大きさに設定されている場合は、パフォーマンスをある程度向上させることができます。

まず、いくつかの用語を説明します。

- rmem_max - 最大受信バッファサイズ
- wmem_max - 最大送信バッファサイズ
- wmem_default - デフォルトの送信バッファサイズ
- max_backlog - 受信キューの最大サイズ
- Netdev_budget - 1 回のポーリングサイクルですべてのインタフェースから取得されるパケットの最大数

最適なバッファサイズを算出するためにパケットサイズ方式を使用しました。ただし、レイテンシーが高いセットアップの場合はレイテンシー方式を使用することをお勧めします。

レイテンシー方式

レイテンシーを使用して 1 つの TCP 接続の最大スループットを算出することで最適化します。

最適なサイズ = (マイクロ秒単位の往復遅延) × (Mb/秒単位のリンクのサイズ) × 1,024^2

たとえば、ボンディングは 50,000Mb/秒で実行されています。Ceph ノードから OpenShift クラスタまでのレイテンシーは、マイクロ秒に換算すると、0.208 を 1,000 で割った値になります。次に、50,000 = 10.4 を掛けてバイトに変換すると、10,905,190 になります。

つまり、

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

ここで必要なのは wmem_default の設定です。これには 4 つのパケット、つまり 10,905,190 の 1/4 が含まれています。設定は次のようになります。

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

パケットサイズ方式

パケットサイズによる最適化 - もう 1 つの方式では、ファイルあたりの平均パケットサイズが 512KB であると仮定し、各ソケットの最適なサイズは、最大サイズ = (Mb/秒単位のパケットサイズ) × 1,024^2 となります。

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

レイテンシーによる最適化は、定期的に高いレイテンシーが発生するネットワークの場合に推奨される手法であることに注意してください。

アダプターのチューニング

NIC バッファ

複数のホストを含む大規模セットアップでは、受信トラフィックの速度がカーネルの能力を超えてしまい、バッファを十分な速さで排出できない可能性があります。その場合、NIC バッファがオーバーフローし、トラフィックが失われ、softirq ミスとなります。netdev_budget として知られるこのシナリオを回避するためには、softirq の CPU 時間を増やすことができます。また、必要に応じてバジェットを増やすことができます。私たちのセットアップでは、バジェットを 1,000 に増やしました。つまり、softirq は CPU から離れる前に NIC 上で 1,000 個のメッセージを排出します。

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

/proc/net/softnet_stat の 3 列目が時間の経過とともに徐々に増加している場合

```
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
```

これは、softirq が十分な CPU 時間を取得できなかったことを示しています。その場合、バジェットを増やすことが可能であり、できれば少しずつ増やす必要があります。

バックログキュー

Linux カーネル内には、NIC からトラフィックを受信した後、プロトコルスタック (TCP/IP/ISCSI) のいずれかによって処理される前に、トラフィックが保存されるキューがあります。
各 CPU コアにはトラフィックが保存されるバックログキューがあり、キューがすでに最大容量に達している場合、追加のパケットはドロップされます。

/proc/net/softnet_stat の 2 列目が時間の経過とともに徐々に増加している場合

```
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
```

これは、netdev バックログキューがオーバーフローしているため、netdev_max_backlog を、できれば少しずつ増やす必要があることを示しています。

このスケールのセットアップでは、値を 5,000 に設定します。

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

Red Hat

RHCS のチューニング

このセクションでは、この大規模環境に対応するために Ceph ノードで行った Ceph 固有のチューニングについて説明します。

配置グループ (PG) のチューニング

PG はオブジェクトストレージ・デバイス (OSD) によって複製されるオブジェクトのコレクションであり、各オブジェクトはデータとメタデータを保存するコンテナです。

プールあたりの PG の最適な数を得るには、目標を OSD あたり 100 PG に設定します (rbd および librados のベストプラクティスによる)。次に、プールの最大使用容量 (デフォルトは 85%) を掛けて、レプリカの数で割り、最も近い 2 の累乗に丸めます - 2^(round(log2(x)))。

```
( OSD あたりの目標 PG 数 ) x ( OSD の数 ) x ( %Data ( プールの最大使用容量 ))
-------------------------------------------------------------------------
                              ( 3 レプリカ )
```

あるいは、私たちのセットアップでは (100 * 141*0.85) / 3 = 3,995 を 2 の累乗に丸めると、PG の合計は 4,096 になります。

basic calculator (bc) を使用してスクリプトを作成しました。

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

OSD あたりの PG の数をさらに増加できることに注意してください。これにより、クラスタ全体で OSD あたりの負荷の差異を減らすことができる可能性がありますが、各 PG を保存している OSD 上ではより多くの CPU とメモリが必要になります。したがって、OSD の数は環境ごとにテストし、チューニングする必要があります。

次に、これらの設定をクラスタに適用します。

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

Red Hat

Prometheus のチューニング

Ceph クラスタを監視するには、Ceph ダッシュボードを使用して Ceph プールの統計を表示します。次のコマンドを実行できます。

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

大規模クラスタのシステムの負荷を軽減するには、次のコマンドを使用してプール統計の収集を調整できます。

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

また、Ceph マネージャーがボトルネックになって他の ceph-mgr プラグインが実行できなくなることを避けるために、Prometheus のポーリングレートを下げるのも有効です。

今回は、コマンドで収集間隔を 60 秒に設定します。

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

Red Hat

OpenShift Virtualization

はじめに

OpenShift Virtualization の機能と安定性を大規模に実証するために、以下のワークフローの動作確認を行います。

- VM のデプロイメント
- VM のブートストーム
- VM の追加レイテンシー (ワークロードあり、ワークロードなし)
- VM の移行 (ワークロードあり、ワークロードなし)

このセットアップの密度目標は、クラスタ全体で 3,000 台の VM と 21,400 個の Pod に設定しました。これは、以下の構成によって実現しました。

- 1,500 台の RHEL 8.5 永続ストレージ VM
- 500 台の Windows10 永続ストレージ VM
- 1,000 台の Fedora Ephemeral ストレージ VM
- 21,400 個のアイドル状態の Pod

もっと簡単に言えば、密度はノードあたり 30 台の VM と 214 個の Pod です。

KubeletConfig

冒頭で述べたスケールを達成するには、次の KubeletConfig を適用して、ノードあたりの Pod のデフォルト制限をバイパスする必要がありました。

```
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
```

`maxPods` を 500 (デフォルトは 250) に増やす以外に、バースト増加の可能性に対応するためにデフォルトの `kubeAPIBurst` を 200 (デフォルトは 100) に、`kubeAPIQPS` を 100 (デフォルトは 50) に増やしました。一般的な比較として、標準的な Kubernetes のデフォルトの最大 Pod 数はノードあたり 110 Pod です。

上記の変更はいずれも必須ではありません。長期的なテストが行われていないため、ノードあたりの Pod 数が 250 を超えることは現在推奨されていません。ただし、私たちのテストでは、密度に関連する問題は発生しませんでした。

私たちが適用した追加の Kubeletconfig は BZ#1984442 に関連しており、これによってすべてのノードに VM Pod を均等に分散させることができます。

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

ラベルを使用して、両方のカスタム Kubeletconfig をワーカーノードに対して有効にすることができます。

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

KubeletConfig を変更すると、関連付けられているノードが再起動されます。

テンプレート

私たちが使用した OS テンプレートはすべて、OpenShift Virtualization テンプレートウィザードを通じて利用できるデフォルトのテンプレートであり、カスタムネットワークにいくつかの変更が加えられています。

Red Hat Linux

使用されているテンプレートは以下のコマンドで取得できます。

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

加えた変更も含めてここに全体を記載します。

```
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

使用されているテンプレートは以下のコマンドで取得できます。

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

加えた変更も含めてここに全体を記載します。

```
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

使用されているテンプレートは以下のコマンドで取得できます。

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

加えた変更も含めてここに全体を記載します。

```
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
```

VM のデプロイメント

VM のデプロイメントは、あらゆる仮想環境の基盤です。大規模環境を目指す場合、多数の VM をどれだけ速くデプロイできるかがプロダクションの効率に直接影響します。

このセクションでは、Ceph CSI クローニング手法を使用してゴールデンイメージソースから複数の VM イメージのクローンを作成する場合、どのようなパフォーマンスが期待できるかについて説明します。

CSI クローンのクローニング戦略によって 100 を超える VM のクローンを作成する場合、Ceph CSI がクローンをパージせず、手動でのクローンの削除も失敗する可能性があります (BZ#2055595)。したがって、現時点ではスナップショットのクローニングを避け、代わりに `cloneStrategy: copy` を使用することが最善です。

CSI スナップショットのクローニングを有効にするには、OpenShift Virtualization ストレージプロファイルを編集する必要があります。

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

そして、以下の仕様を追加します。

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

まず、RHEL QCOW イメージをホストの 1 つからデータボリューム (DV) にインポートします。

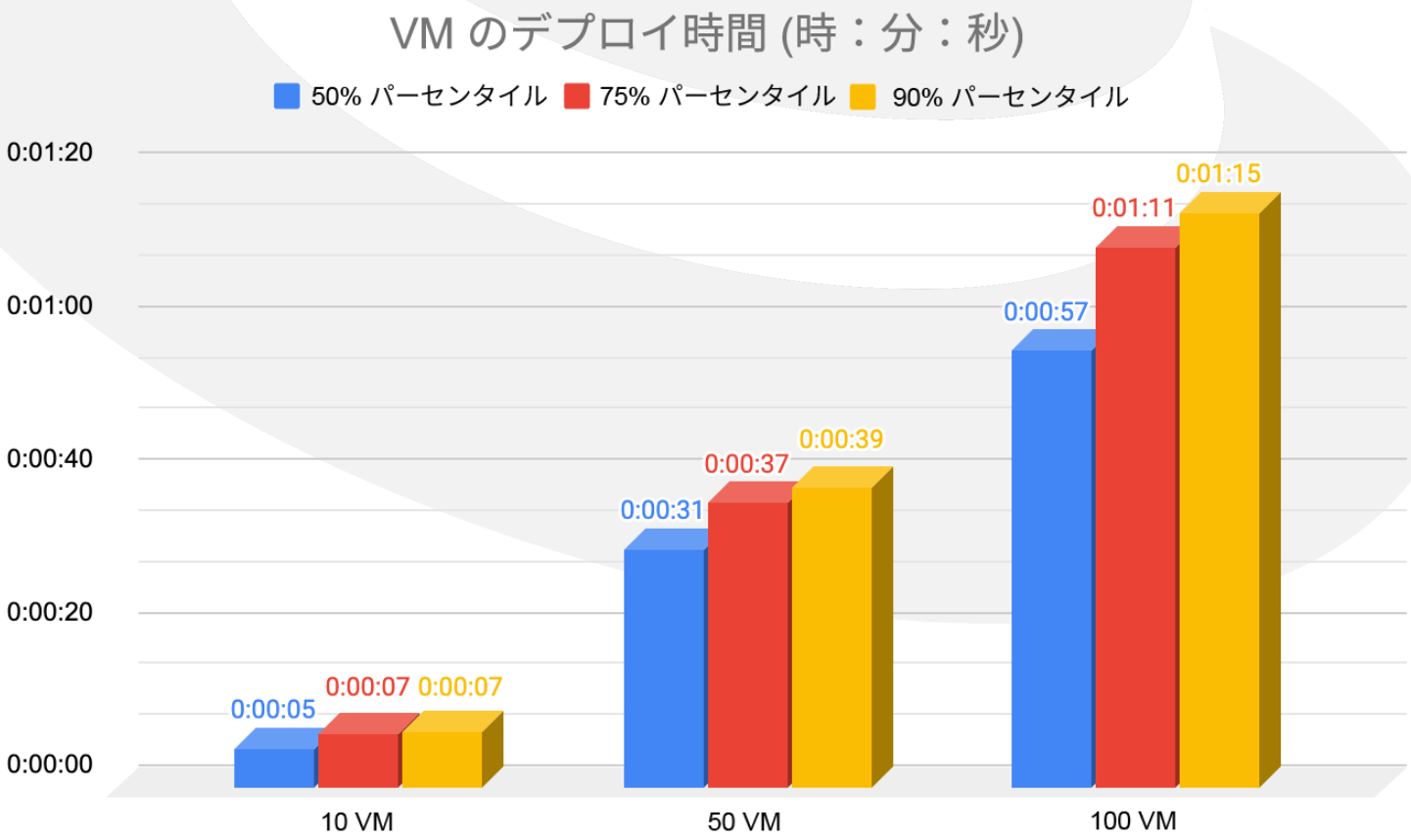
```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: rhel-clone-dv
spec:
  source:
      http:
         url: http://internel.server.com/ISO/rhel8.qcow2
  pvc:
    accessModes:
      - ReadWriteMany
    resources:
      requests:
        storage: 40Gi
    volumeMode: Block
    storageClassName: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

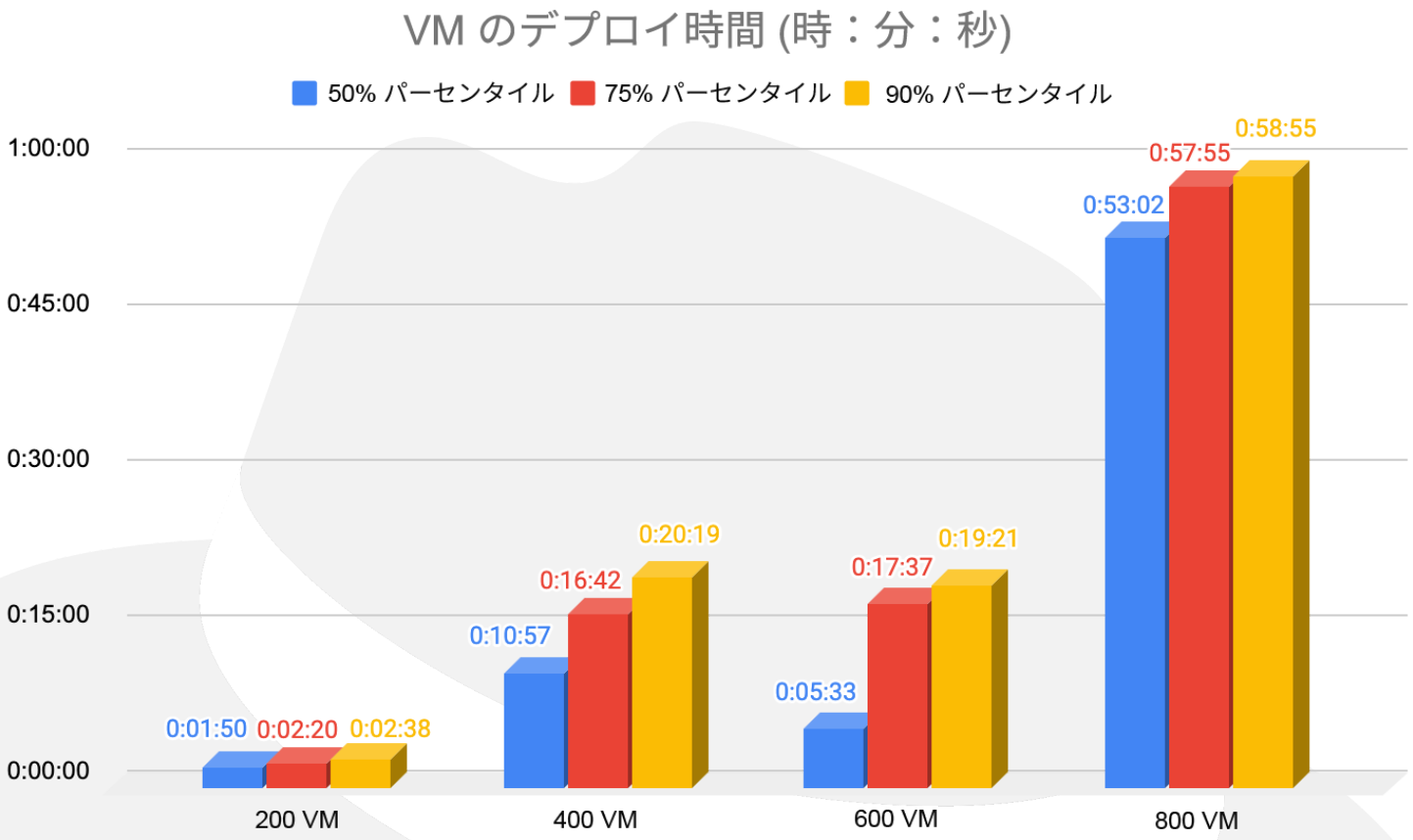
インポートが完了したら、必要な数の VM を並行してデプロイし、各 VM のクローニングが完了するまでにかかる時間を測定します。そのためには、クローンが完了するまで 2 秒間隔ですべての VM にクエリを実行します。

VM をデプロイする最も効果的な方法は 100 個のグループでデプロイすることです。つまり、100 台の VM をデプロイし、クローニングの完了を待って、次の 100 台の VM をデプロイします。

一般に、10 台の VM の並列デプロイメントを超えるとペナルティが発生します。これは、Ceph CSI レベルで親イメージの vol-id に対するロックが取得されるからです。そのため、同じ親イメージからのクローニングは並列ではなく実際には直列であり、外部プロビジョナーが CSI ドライバーに一度に送信できる gRPC 並列呼び出しは常に 10 個だけです。

さらに、クローン数が 250 を超えると rbd イメージが平坦化し始め、クローニングのペナルティがさらに増加します。クローンの平坦化にかかる時間は、スナップショットのサイズに応じて増加します。





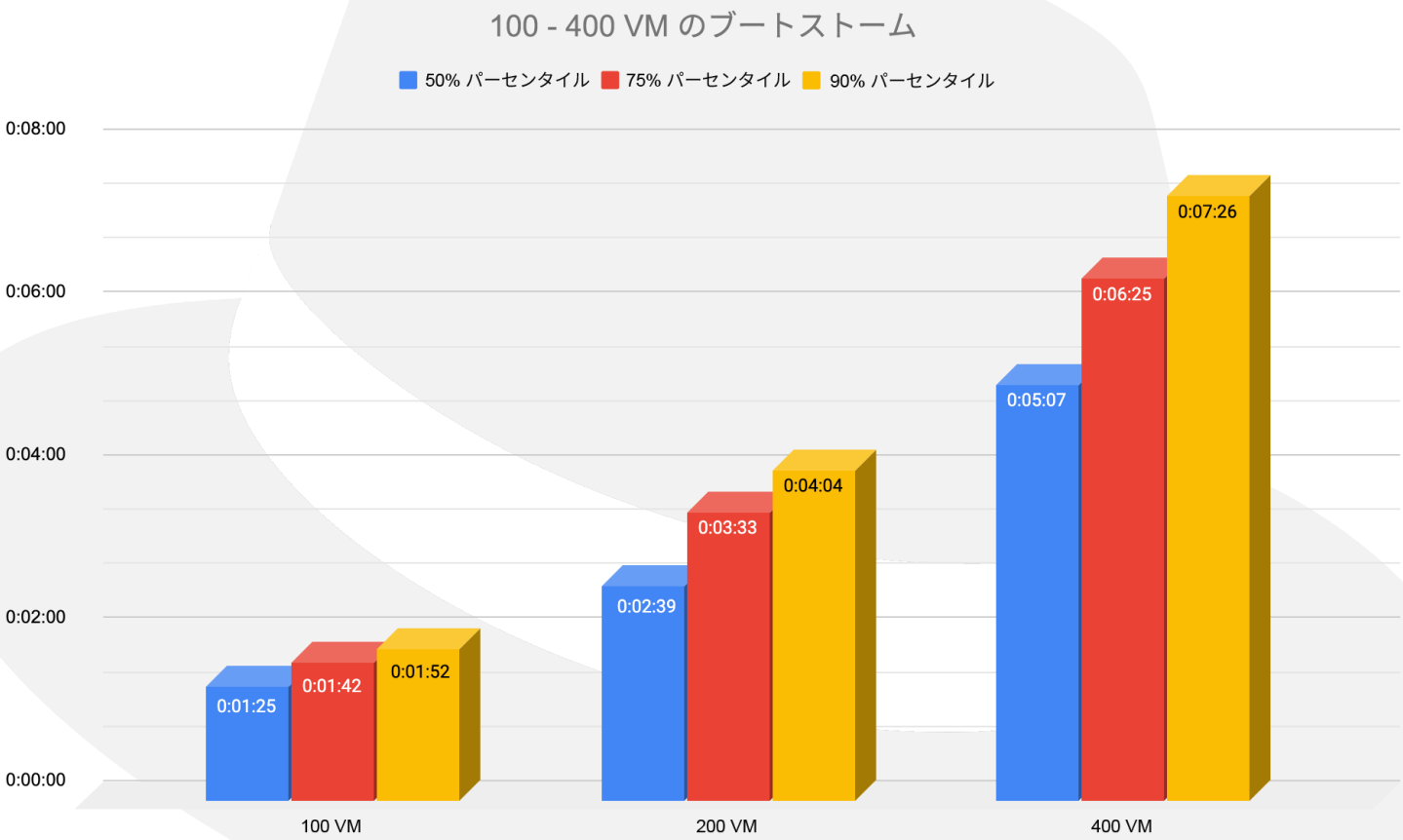
VM のブートストーム

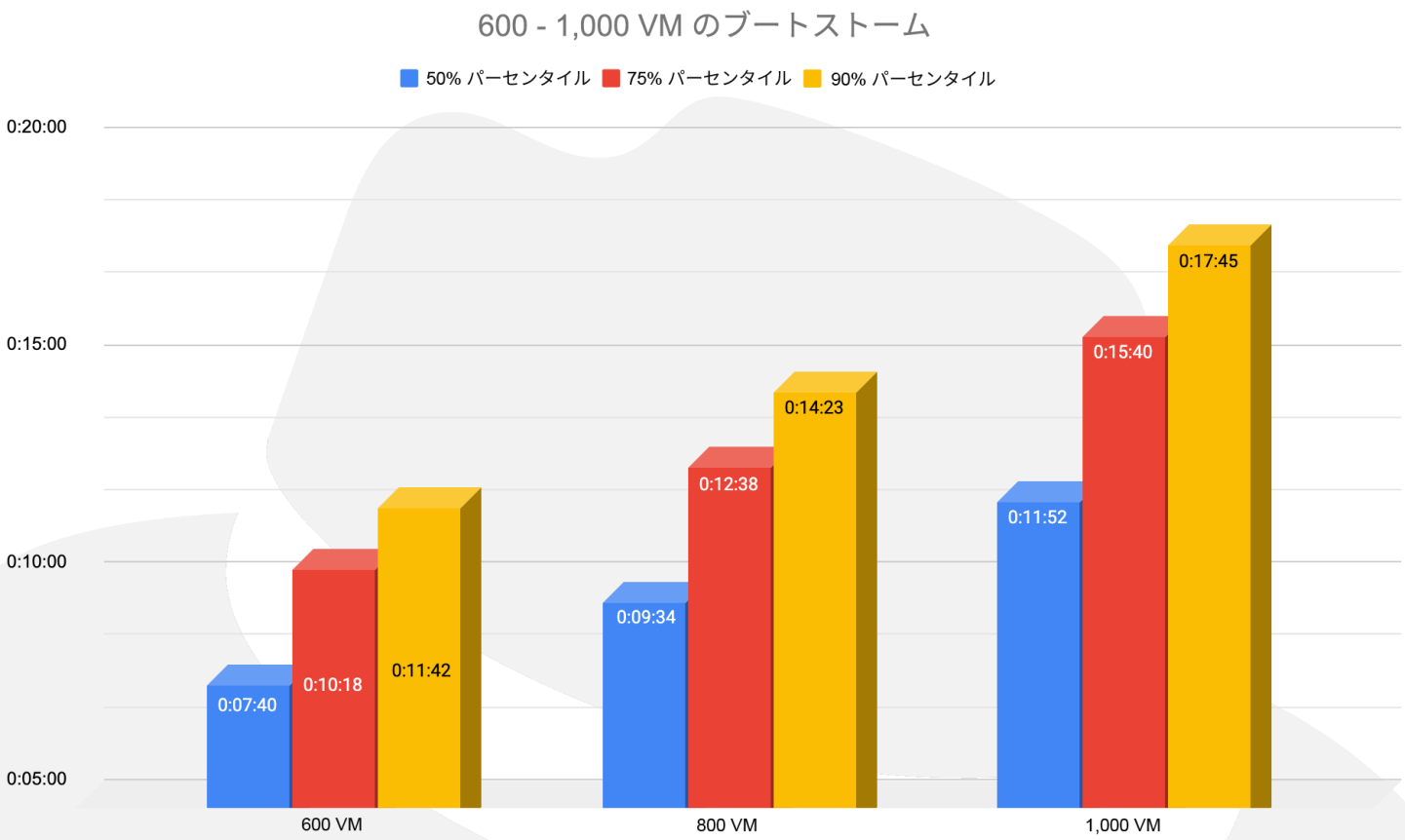
このシナリオでは、多数の VM の起動にかかる時間をテストしました。これは、OpenShift Virtualization とコントロールプレーンの両方の回復力がどの程度高いかを示しています。これは、停電などの災害から環境を回復する場合に発生することが多いシナリオです。

測定はすべての VM に対してリクエスト時に開始され、VM が実行されて SSH アクセスを通じてアクセス可能になった (ホストが正常に起動し、SSH デーモンが起動) 後に停止するまで行われます。私たちは、各 VM に対してクエリを実行することで時間を測定しました。ステータスが「Running」に変わると、SSH 接続が成功するまで 2 秒ごとに VM への SSH 接続が試行されます。

デプロイメントのシナリオと同様に、すべての VM の開始リクエストが並行して実行され、クラスタのすべての可動部分に負荷がかかります。

以下のグラフからわかるように、同種の OCP クラスタでは、VM が 1,000 個までの場合は起動時間はほぼ線形ですが、キューが大きくなると起動時間が遅くなる可能性があります。




600 - 1,000 VM のブートストーム
50% パーセンタイル
75% パーセンタイル
90% パーセンタイル
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
1,000 VM

Red Hat

VM のレイテンシー

複数の永続ボリューム要求 (PVC) があり、かつ `dedicatedIOThread:` が true に設定されていない限り、各 VM には IO を処理するための独自の IO スレッドがあります。その場合、各 PVC は独自の IO スレッドを持つことになります。

以下のシナリオでは、ワーカーノードごとに 15 台の VM を使用し、最大 64 のワーカーノード、つまり 960 台の VM をテストして、RHCS クラスタにアクセスしている同時スレッドが複数存在する場合であっても、それら自体ではレイテンシーペナルティが発生しないことを実証しました。

このシナリオでは、ランダム読み取りとランダム書き込みの両方に 4KB ブロックサイズを使用することとし、以下のテストを実行しました。

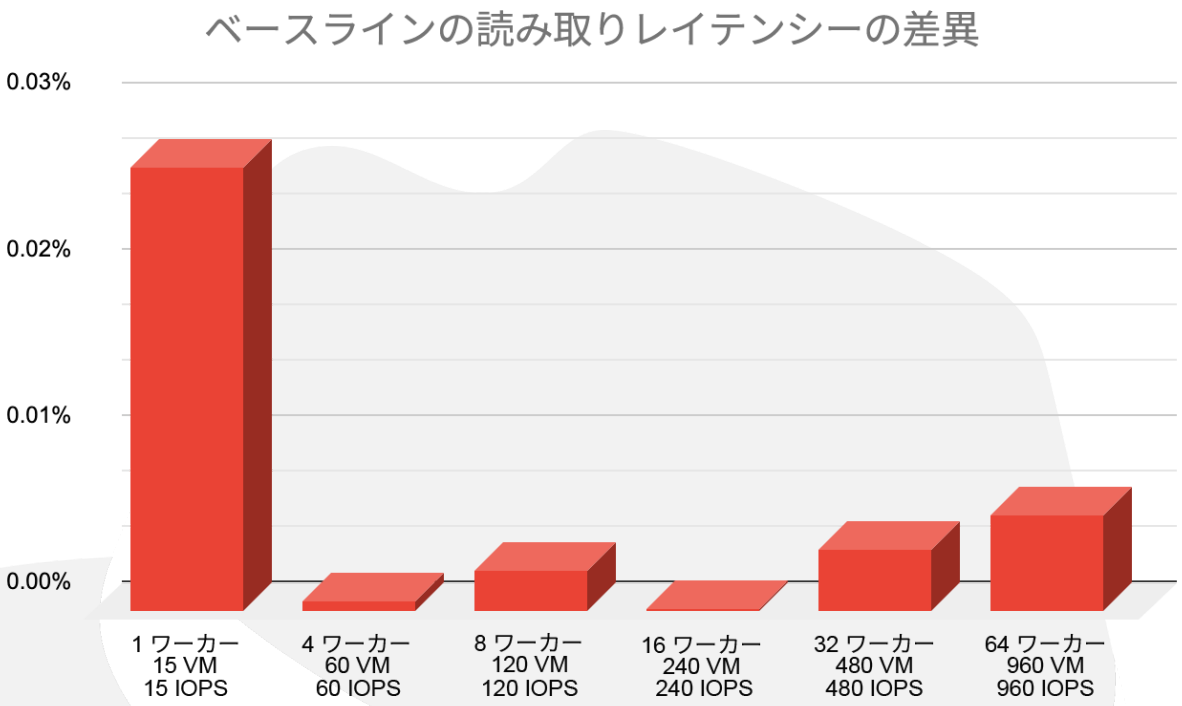
- ベースライン - ワーカーノードごとに 15 台の VM を使用しました。各 VM は単一の IOPS を生成、15 台の VM (15 IOPS、単一ノード) から始まり合計 960 VM (960 IOPS) となる 64 ノードで終了します。
- ワークロード - ワーカーノードごとに 15 台の VM を使用しました。各 VM は 1,000 IOPS を生成、15 台の VM (15,000 IOPS、単一ノード) から始まり合計 960 VM (960,000 IOPS) となる 64 ノードで終了します。

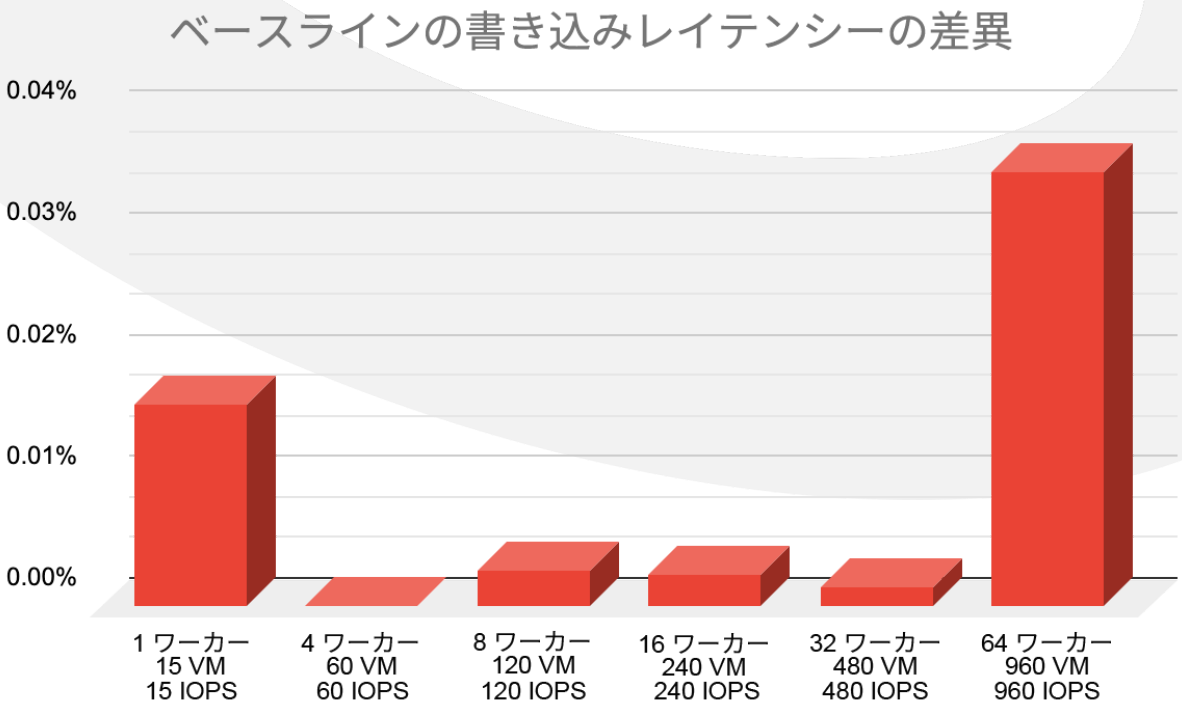
各 VM のファイルシステム・データセットは 300 のディレクトリで構成されます。各ディレクトリには 8 つのファイルが含まれ、各ファイルのサイズは 20 MiB です。つまり簡単に言うと、各 VM には 4.8 GiB のデータセットがあります。

RHCS クラスタ上のネットワーキングと Ceph の異種ディスクによって発生する可能性のある不整合を可能な限り回避するために、小さなブロックを選択しました。

テストでは最大 960 台の VM を使用しましたが、実際には合計 3,000 台の VM が 21,400 個の Pod とともにクラスタ上で実行されていました。

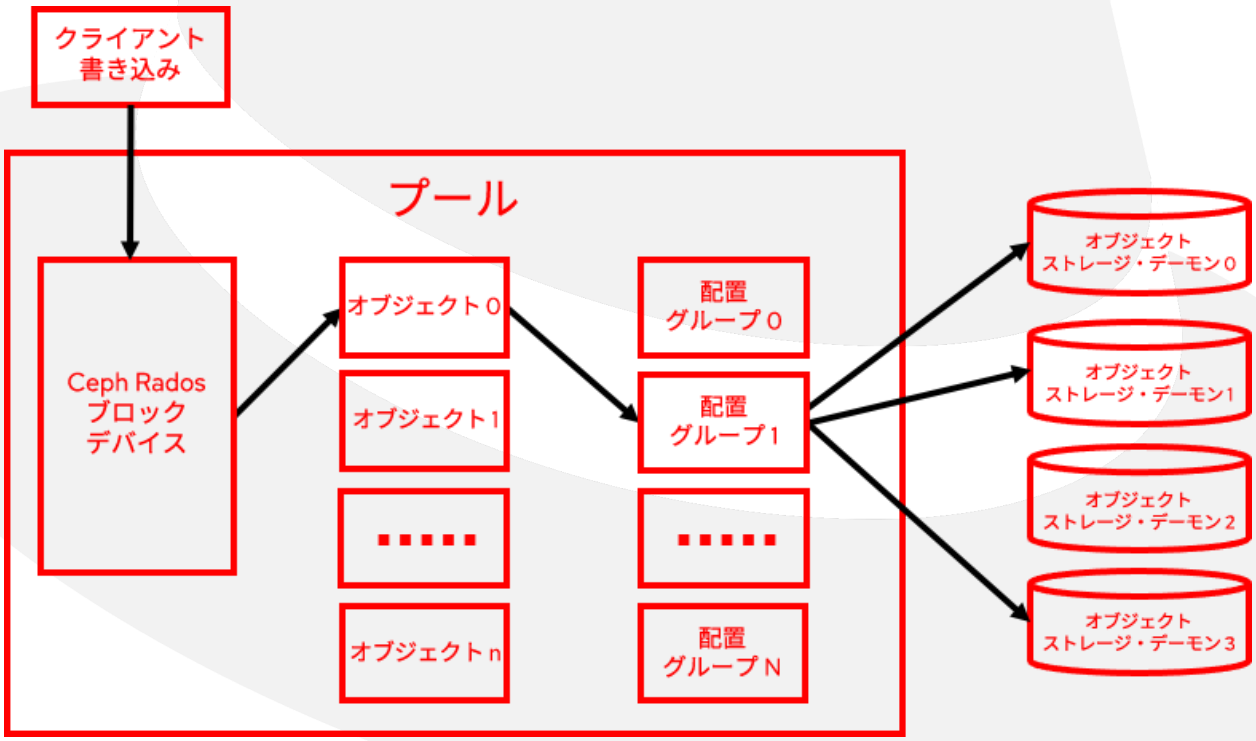
以下のグラフからわかるように、ベースラインテストの実行中、ランダム読み取りとランダム書き込みの両方のレイテンシーの差異は 0.04% 未満でした。


ベースラインの読み取りレイテンシーの差異
0.03%
0.02%
0.01%
0.00%
1 ワーカー
15 VM
15 IOPS
4 ワーカー
60 VM
60 IOPS
8 ワーカー
120 VM
120 IOPS
16 ワーカー
240 VM
240 IOPS
32 ワーカー
480 VM
480 IOPS
64 ワーカー
960 VM
960 IOPS

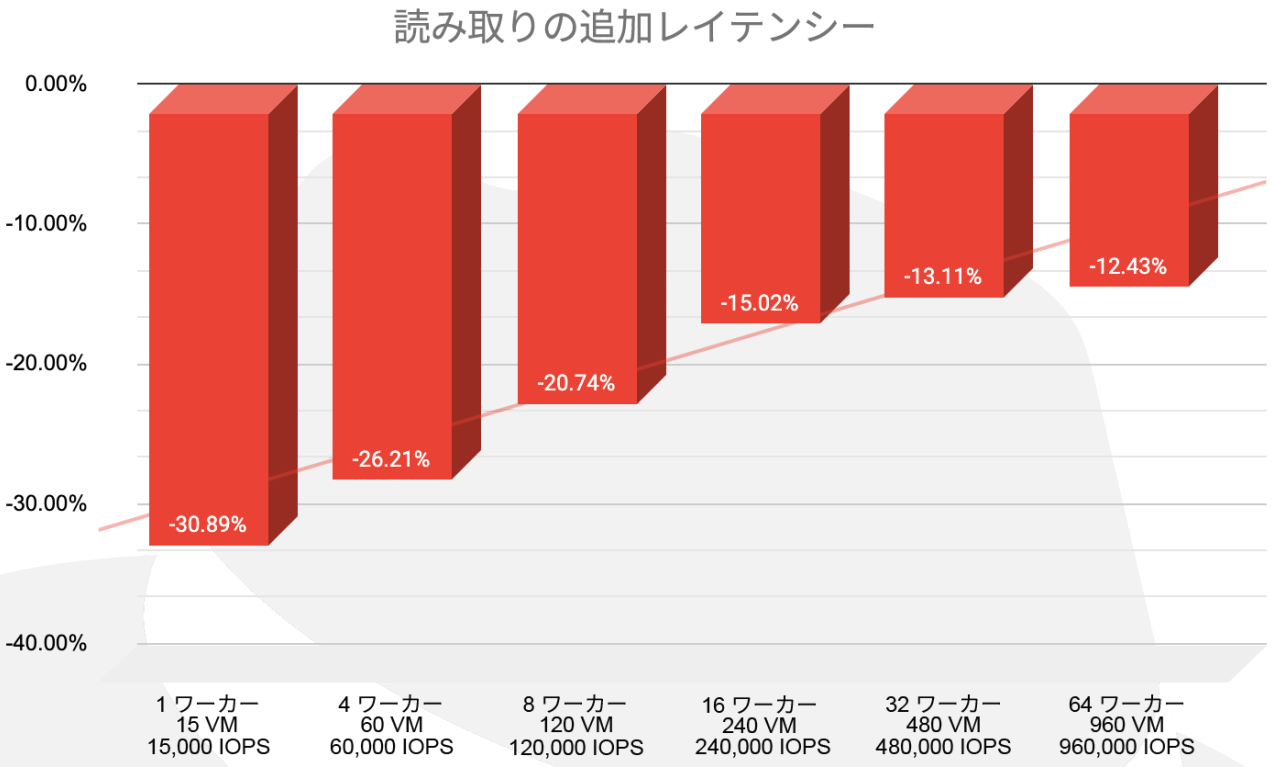

ベースラインの書き込みレイテンシーの差異
0.04%
0.03%
0.02%
0.01%
0.00%
1 ワーカー
15 VM
15 IOPS
4 ワーカー
60 VM
60 IOPS
8 ワーカー
120 VM
120 IOPS
16 ワーカー
240 VM
240 IOPS
32 ワーカー
480 VM
480 IOPS
64 ワーカー
960 VM
960 IOPS

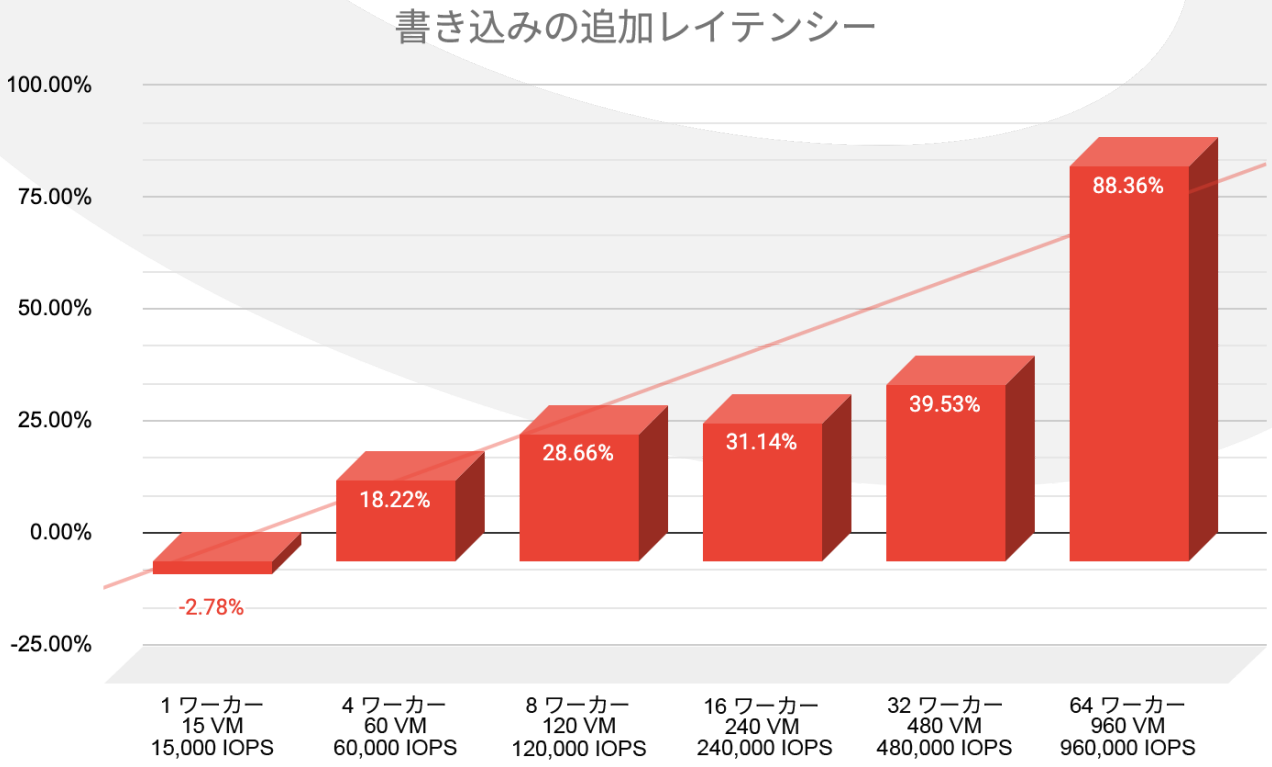
以下のグラフは、ベースラインの結果と比較した、ワークロードのシナリオのレイテンシーの傾向を示しています (低いほど優れていることを意味します)。読み取りパフォーマンスに関しては、IOPS 速度が高くなると、レイテンシーがある程度低くなります。これは、VM 上のワークロードがアイドル/低 IOPS である場合と比較して、バーストが高くなるとリソース割り当てが発生するためです。

ただし、書き込みのデータフローはこれと異なっており、高可用性を維持するためにはそうである必要があります。以下の図に示すように、Ceph が生成する書き込みごとに、データの 3 つのコピーがネットワークを介してそれぞれの OSD に送信されます。したがって、書き込みレイテンシーが影響を受けます。



レイテンシーの影響を受けやすいアプリケーションの場合、データレプリカが削減されるごとに、書き込みレイテンシーのオーバーヘッドは 3 分の 1 削減されます。


読み取りの追加レイテンシー
0.00%
-10.00%
-20.00%
-30.00%
-40.00%
-30.89%
-26.21%
-20.74%
-15.02%
-13.11%
-12.43%
1 ワーカー
15 VM
15,000 IOPS
4 ワーカー
60 VM
60,000 IOPS
8 ワーカー
120 VM
120,000 IOPS
16 ワーカー
240 VM
240,000 IOPS
32 ワーカー
480 VM
480,000 IOPS
64 ワーカー
960 VM
960,000 IOPS


書き込みの追加レイテンシー
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 ワーカー
15 VM
15,000 IOPS
4 ワーカー
60 VM
60,000 IOPS
8 ワーカー
120 VM
120,000 IOPS
16 ワーカー
240 VM
240,000 IOPS
32 ワーカー
480 VM
480,000 IOPS
64 ワーカー
960 VM
960,000 IOPS

Red Hat

VM の移行

以下のシナリオでは、1,000 台の VM の移行をテストしました。現実的な移行のシミュレーションを行うために、VM を移行するだけでなく、それらの VM が存在するワーカーノードの再起動も行いました。そのために、ノードを 3 つのゾーンに分割し、空のマシン構成を特定のゾーンに適用しました。これにより、ワーカーノードに存在するすべての VM が削除されると、その特定のゾーンに関連付けられていたすべてのノードが再起動されます。

まず、すべてのワーカーを特定のゾーンにラベル付けしました。

```
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=""
```

次に、ゾーンごとにマシン構成プールを作成しました。`maxUnavailable: 10` は、ゾーンが存続する間いつでも停止できるノードの数を設定します。

```
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
```

また、ハイパーコンバージド・クラスタ Operator も編集しました。

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

さらに、以下の移行設定をして、並列移行の量を増やしました。

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

テストを通じて、virt-api Pod の数を **750 VM ごとに 1 kubevirt-api Pod** の比率に増やすことが強く推奨されることがわかっています。このセットアップでは 3,000 台の VM を実行していたため、kubevirt API Pod の数を 4 にスケールしました。

このシナリオの自動スケーリング機能はすでに開発中であり、Github#7101 で追跡できます。現時点では、ハイパーコンバージド Operator にパッチを適用することによって手動で行うことができます。

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

次に、以下のマシン構成を作成して移行をトリガーします。

```
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: ""
```

移行された 1,000 台の VM は以下のような構成でした。

- 400 台の RHEL VM
- 400 台の Fedora VM
- 200 台の Windows VM

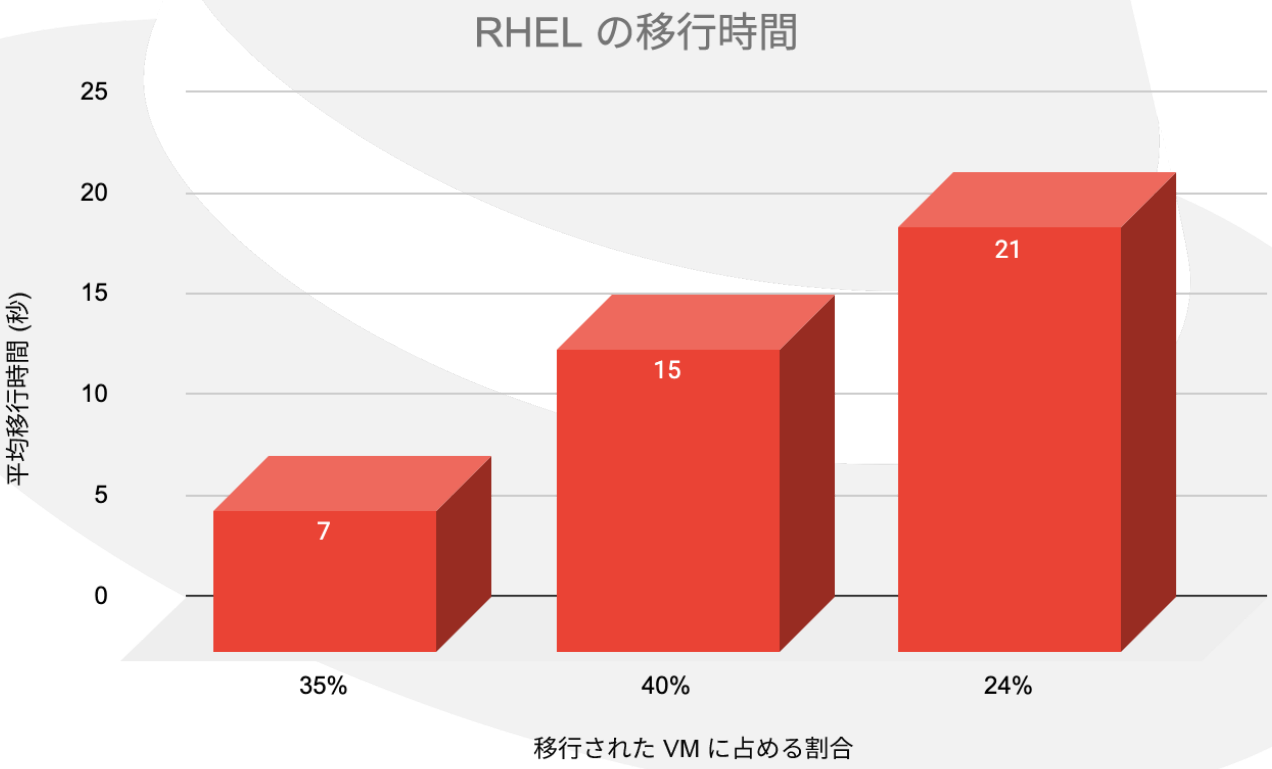
VM と同じ場所に配置された 7,000 個の Pod も、別のワーカーノードで再起動されました。

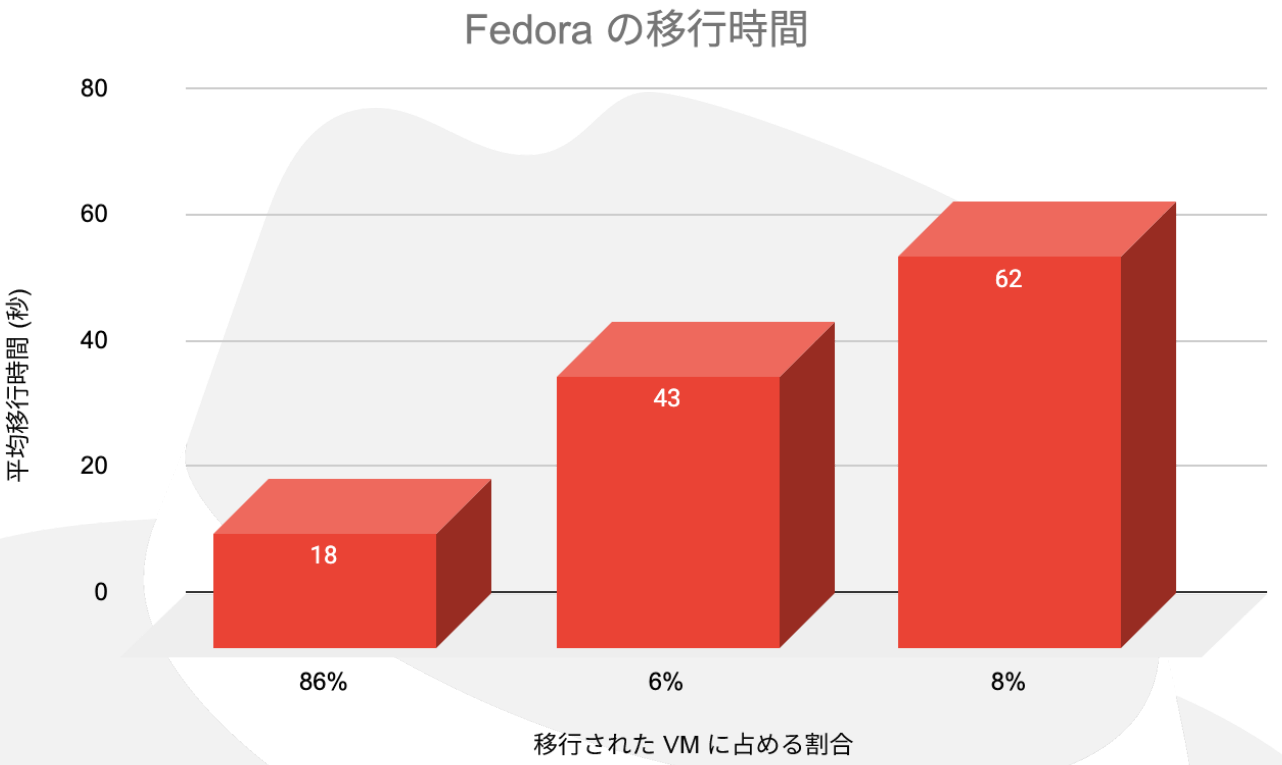
移行時には、どの VM についても、移行の完了までにかかった時間を VMI ログで確認できます。

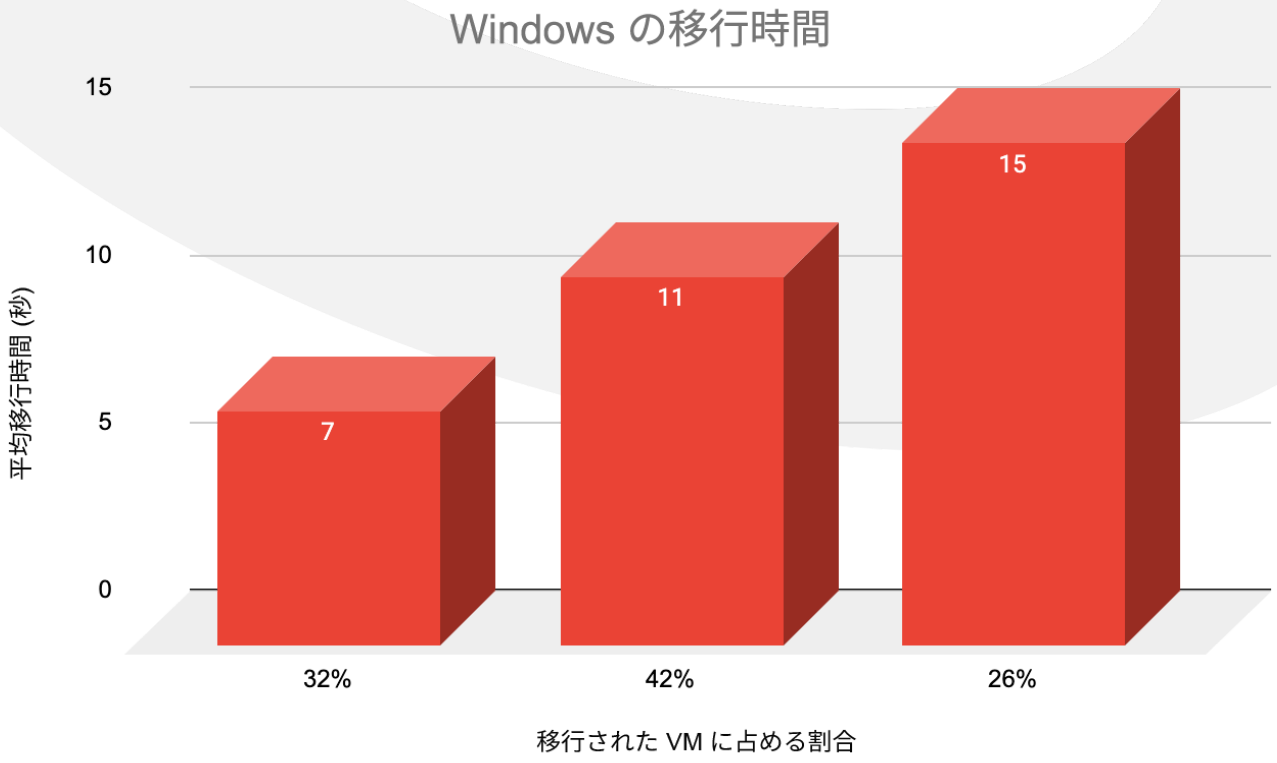
```
Phase Transition Timestamps:
  Phase:                         Scheduling
  Phase Transition Timestamp:    2022-04-10T07:15:13Z
  Phase:                         Scheduled
  Phase Transition Timestamp:    2022-04-10T07:15:23Z
  Phase:                         Running
  Phase Transition Timestamp:  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

以下のグラフは、各 OS の移行時間をパーセンテージで示したものです。たとえば、RHEL VM の移行では、VM の 35% が平均 7 秒で移行を完了しました。

移行時間は必ずしも OS に関係するわけではなく、その時点のゲスト負荷、ホスト負荷、ネットワーク負荷、ストレージ・テクノロジー、移行ポリシー、イメージサイズなどに関係します。したがって、結果は異なる場合があります。




Fedora の移行時間
80
60
40
20
0
平均移行時間 (秒)
18
43
62
86%
6%
8%
移行された VM に占める割合


Windows の移行時間
15
10
5
0
平均移行時間 (秒)
7
11
15
32%
42%
26%
移行された VM に占める割合

OS ごとの平均移行時間は以下のとおりでした。

OS	平均移行時間 (秒)	コメント
RHEL	14	40GiB PVC
Fedora	23	コンテナディスク evictionStrategy: Restart
Windows	12	40GiB PVC

まとめると、移行にかかった時間は最初から最後までで 118 分、さらに、前述した `maxUnavailable: 10` によりノードが「Ready」のステータスに達するまでの待ち時間が 35 分でした。

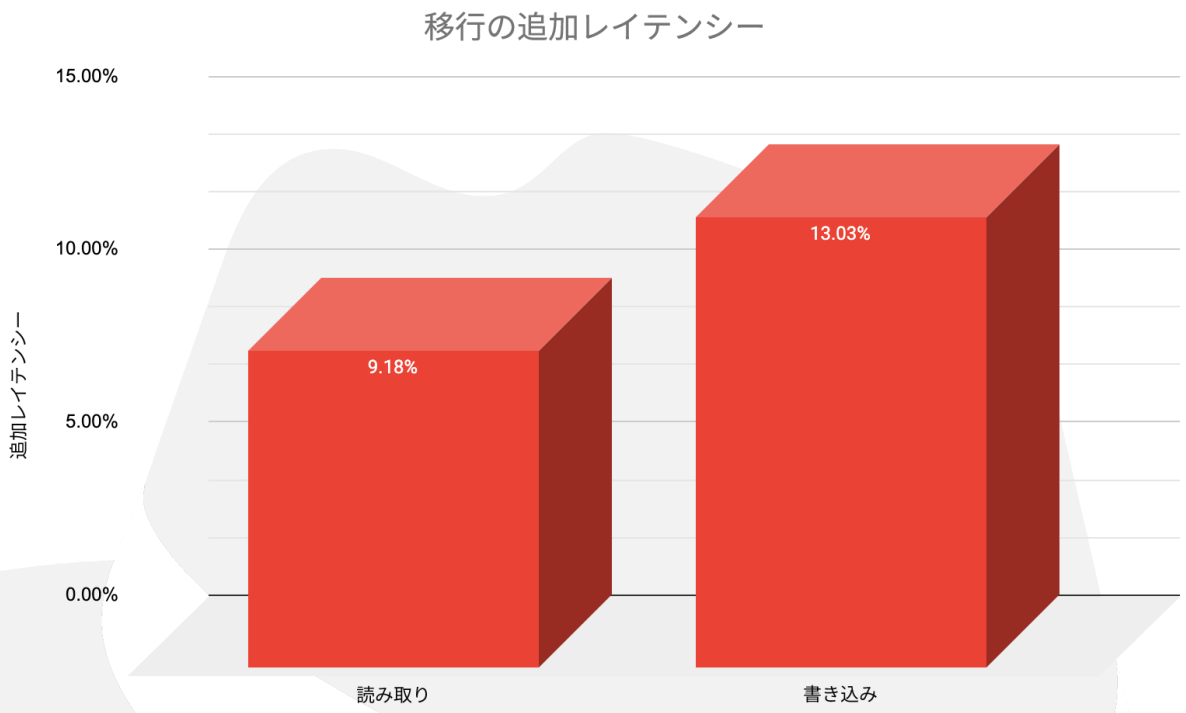
VM 移行の追加レイテンシー

以下のシナリオでは、1,000 台の VM の移行をテストしましたが、このテストでは RHEL VM のみを使用しました。RHEL VM のみを使用することとしたのは、異なるオペレーティングシステムによる IO の処理方法の違いによって生じる可能性のある不一致を避けるためです。

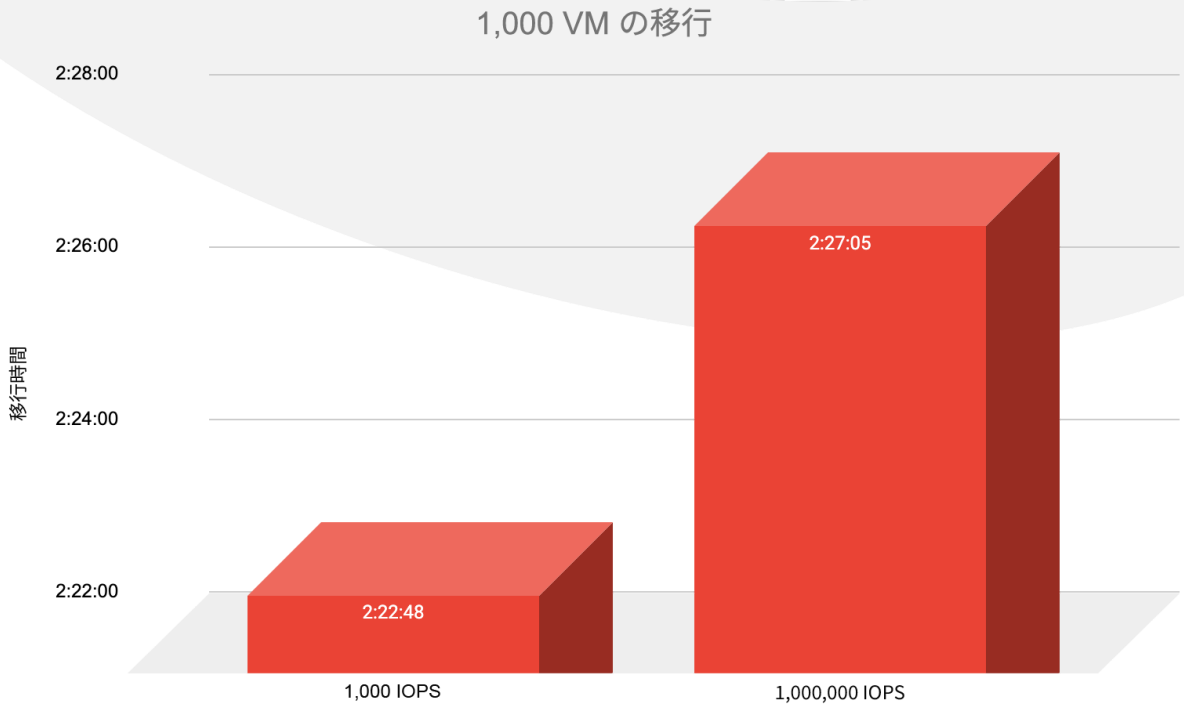
先程と同様に、ランダム読み取りとランダム書き込みの両方に 4KB ブロックサイズを使用することとし、以下のテストを実行しました。

- ベースライン - 1,000 台の VM がそれぞれ 1 IOPS を生成、合計 1,000 IOPS が生成されます。
- ワークロード - 1,000 台の VM がそれぞれ 1 秒あたり 1,000 IOPS を生成、合計 1,000,000 IOPS が生成されます。

以下のグラフは、ベースラインと比較した、読み取りと書き込みの両方における移行中の平均レイテンシーペナルティを示しています (RHEL VM のみ)。



さらに、以下のグラフからわかるように、移行時間への影響は 5% でした。移行時間の結果は [BZ#2069098](BZ#2069098) により異なる場合があります。



大規模なクラスタアップグレード

以下のシナリオでは、マイナーアップグレードとメジャーアップグレードの両方をテストしました。

まず、実際のプロダクション・アップグレードのシミュレーションを行いながら、つまり 3,000 台の VM と 21,400 個の Pod がすべてクラスタ上で実行されている状態で、クラスタをバージョン 4.9.15 から 4.9.23 にアップグレードすることから始めました。さらに、1,500 台の VM 上で 4KB の軽量ワークロードが、VM あたり 100 IOPS の速度で生成されました。

以下のコマンドを実行してアップグレードを開始しました。

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

アップグレードの進行状況は、以下のコマンドを使用して追跡できます。

```
$ 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)
```

マイナーアップグレードのプロセスの所要時間は合計 **35 分**でした。

次のステップでは、クラスタをバージョン 4.9.23 から 4.10.9 にアップグレードし、先程と同じ条件で再度アップグレードを実行することで、メジャーアップグレードをテストしました。以下のコマンドを実行してアップグレードを開始しました。

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

先程と同様に、アップグレードの進行状況は、以下のコマンドを使用して追跡できます。

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

メジャーアップグレードのプロセス全体の所要時間は合計 **136 分**でした。

一部のアップグレードには、すべてのノードでソフトリセットの実行を必要とする変更が含まれる場合があり、これにより移行時間が追加されるため、アップグレード時間が大幅に増加することに注意してください。

まとめ

このリファレンスアーキテクチャでは、OpenShift Virtualization の機能と回復力を大規模に実証しました。Red Hat OpenShift Container Platform の OpenShift Virtualization 機能は、Red Hat Ceph Storage や Red Hat OpenShift Data Foundation と連携し、コンテナ、仮想マシン、高可用性ストレージを組み込んだ完全なプロダクション・ソリューションを提供でき、また、ハードウェアの最小要件を満たすあらゆるホストにデプロイできます。

このリファレンスアーキテクチャは、設定した目標の達成方法を概説していますが、環境条件と要件を前提として適切な回復力、拡張性、シームレスな日常運用を実現するために、他のアーキテクチャを考慮することが重要です。たとえば、一定の制限を超える場合、ノード数またはワークロードが大きくなりすぎる場合やクラスタ上で発生するチャーンが多すぎる場合は、マルチクラスタアプローチを検討する必要があります。

その他の資料

システムと環境の要件：
https://docs.openshift.com/container-platform/3.11/install/prerequisites.html

OpenShift テンプレート：
https://docs.openshift.com/container-platform/4.9/openshift_images/using-templates.html

マシン構成 Operator:
https://docs.openshift.com/container-platform/4.9/post_installation_configuration/machine-configuration-tasks.html

ライブ移行とタイムアウト:
https://docs.openshift.com/container-platform/4.9/virt/live_migration/virt-live-migration-limits.html

CLI を使用したクラスタの更新:
https://docs.openshift.com/container-platform/4.10/updating/updating-cluster-cli.html

Red Hat について

エンタープライズ・オープンソース・ソフトウェア・ソリューションのプロバイダーとして世界をリードする Red Hat は、コミュニティとの協業により高い信頼性と性能を備える Linux、ハイブリッドクラウド、コンテナ、および Kubernetes テクノロジーを提供しています。Red Hat は、クラウドネイティブ・アプリケーションの開発、既存および新規 IT アプリケーションの統合、複雑な環境の自動化および運用管理を支援します。受賞歴のあるサポート、トレーニング、コンサルティングサービスを提供する Red Hat は、フォーチュン 500 企業に信頼されるアドバイザーであり、オープンな技術革新によるメリットをあらゆる業界に提供します。Red Hat は企業、パートナー、およびコミュニティのグローバルネットワークの中核として、企業の成長と変革を支え、デジタル化が進む将来に備える支援を提供しています。