PURESTORAGE®

Red Hat

WHITE PAPER

Virtual Machine Provisioning at Enterprise Scale

Sizing and scaling Red Hat OpenShift Virtualization with Portworx

Contents

Introduction

This document explores the results of a performance evaluation conducted on a large-scale Red Hat OpenShift® Virtualization environment backed by Portworx® by Pure Storage®, tested for up to 1,000 virtual machines (VMs). We carried out a series of experiments to measure VM provisioning time, boot times, and live migration times. These benchmarks offer crucial insights for situations like scaling operations and planned infrastructure maintenance and are a valuable reference for those who deploy, operate, and maintain infrastructure services.

It is important to note that the experiments and performance metrics presented in this guide do not represent the upper or maximum limits of the product. These experiments were conducted on specific hardware, and results may vary depending on your organization's hardware and needs.

Target audience

This document is intended as a reference for users who deploy, operate, or maintain infrastructure services. This includes customers, consultants, DevOps engineers, sales engineers, solution architects, and infrastructure administrators. Use this document as a reference for large-scale deployments of Red Hat OpenShift Virtualization backed by Portworx, and apply the insights and learnings to your own environments.

Executive summary

This document outlines the performance evaluation conducted on Red Hat OpenShift Virtualization backed by Portworx storage. The focus of this study was to gather insights and share the learnings from a successful, large-scale Red Hat OpenShift Virtualization cluster consisting of 29 worker nodes and three control plane nodes. The cluster was deployed on a bare metal infrastructure with directly attached solid-state drives (SSDs) and supported the deployment of 1,000 VMs without any failures.

Table 1 summarizes the experiment results.

Experiment Details	Description	Result
VM provisioning	Parallel provisioning of up to 1,000 VMs	1,000 VMs can be provisioned in <22 minutes
VM boot storms	Parallel boot storms of up to 1,000 VMs	1,000 VMs boot up in ~4 minutes; linear increase in boot time observed with growing number of VMs
VM live migration	Parallel live migrations of up to 1,000 VMs not running any load	1,000 VMs can be live migrated in ~18 minutes; average live migration time of 2–3 seconds
Maximum VMs per node	Test CPU and memory limits while creating VMs	113 small-size VMs (2 vCPU with 4:1 overcommit—500m requested vCPU, 4GB memory); VMs with 112 CPU nodes and 503GB memory (limited by memory)
Portworx	Recommended configuration	Portworx operates well when sized with 8 CPUs and 16GB memory for these experiments

TABLE 1 Experiment results summary

Value proposition

Benefits of Red Hat OpenShift

Red Hat OpenShift is a modern application development and deployment platform. It delivers a consistent, scalable, and secure platform for both virtualized and containerized applications from edge to core to cloud environments.

Developer productivity through accelerated development cycles: Red Hat OpenShift provides a comprehensive set of tools and services that allow developers to build, deploy, and run applications with speed and agility.

- **Multi-language and framework support**: supports a wide variety of programming languages and frameworks, giving developers the freedom to use the tools they know and love
- **Integrated CI/CD**: includes built-in continuous integration and continuous delivery (CI/CD) pipelines, which automates the process of building, testing, and deploying applications

Operational efficiency and hybrid-cloud consistency: Red Hat OpenShift delivers a consistent application platform across any environment, whether on premises, in the public cloud, or at the edge.

- **Automation**: automates many of the tasks associated with managing applications and infrastructure, such as provisioning, scaling, and patching
- **Self-service infrastructure**: empowers developers to provision the resources they need on demand, without having to wait for IT

Enterprise-grade security and built-in protection: Red Hat OpenShift provides a layered security approach that protects the entire application life cycle, from the container image to the running application.

- **Policy enforcement**: allows organizations to enforce consistent security policies across all their applications and environments
- **Vulnerability scanning**: includes a built-in container image scanner that helps identify and remediate security vulnerabilities

Flexibility and choice with an open source foundation: Red Hat OpenShift is built on open source technologies like Kubernetes and Linux, which provides greater flexibility and avoids vendor lock-in.

- **Extensive ecosystem**: has a large and growing ecosystem of certified partners that provide a wide variety of integrations and extensions
- **Multiple deployment options**: can be deployed in a variety of ways, including on premises, in the public cloud, or as a managed service

Benefits of Portworx

Portworx cloud-native storage offers a transformative solution for managing stateful applications in Kubernetes environments, far surpassing traditional hardware storage solutions and their basic container storage interface (CSI) drivers. While CSI connectors often fall short in handling creates, updates, and deletes (CRUD) at the scale provided by Kubernetes and containers, Portworx excels in delivering enterprise-grade features that enhance both operational efficiency and platform engineering productivity for containerized environments.

Key benefits of Portworx cloud-native storage include:

- **Accelerated time to revenue**: By streamlining storage management and reducing complexities, Portworx helps organizations achieve faster deployment and time to market.
- **Data resiliency**: Portworx ensures high availability and disaster recovery with advanced data replication and backup capabilities, protecting against data loss and downtime.
- **Enterprise scalability**: Designed to scale with your needs, Portworx supports seamless growth, whether you're operating in a single data center or across multiple clouds.
- **Self-service storage access**: Developers gain the flexibility to provision and manage storage independently through storage classes, empowering them to innovate without waiting for IT intervention.
- **Integrated storage management**: Portworx offers comprehensive management features, including rule-based automation, thin provisioning, and support for multicloud, hybrid-cloud, and on-premises environments agnostic to the hardware providers.
- **Boosted platform engineering productivity**: By providing a unified and efficient storage solution, Portworx enhances the productivity of platform engineers, allowing them to focus on building and maintaining robust applications in multiple environments.

With Portworx, organizations can achieve unmatched data agility and reliability, ensuring their Kubernetes storage and databases are always performant and available, regardless of the underlying infrastructure.

Key features of **Red Hat OpenShift Virtualization with Portworx** include support for live migration of VMs, automated storage provisioning, and integration with Portworx advanced storage capabilities for high-performance workloads. This document outlines critical considerations for storage configuration, performance optimization, and capacity planning, offering best practices for setting up ReadWriteMany (RWX) storage for live migrations, managing persistent volumes, and scaling the infrastructure.

Experiments

This section covers the hardware and software configuration across the Red Hat OpenShift cluster. At a high level, the purpose of these experiments is to systematically characterize the performance and behavior of VMs in a Red Hat OpenShift Virtualization cluster backed by Portworx storage.

Configuration

Table 2 shows the physical components of the experiment.

Component	Specifications	Model/Make/Type
Nodes	29 worker nodes 3 control plane nodes	Bare metal
Node CPU	Total: 112 Thread(s) per core: 2 Core(s) per socket: 28 Socket(s): 2	Intel Xeon Gold 6330 CPU at 2.00GHz
Node memory	503GiB	Micron
Storage (29 worker nodes)	1.7TiB /dev/sdb (metadata path) 1.7TiB /dev/sdc (drive/device) 2.9TiB /dev/nvme0n1 (drive/device)	Intel SSD D7-P5600 and Samsung Electronics Co Ltd NVMe SSD Controller PM173X

TABLE 2 Physical components

Table 3 shows the software components of the experiment.

Component	Version	Description
Red Hat OpenShift	4.17.12	Red Hat OpenShift is a trusted and consistent platform for developing, modernizing, and deploying applications at scale. It accelerates innovation and simplifies AI adoption with a comprehensive set of tools on your choice of infrastructure.
Red Hat OpenShift Virtualization	4.17.3	Red Hat OpenShift Virtualization is an included feature of Red Hat OpenShift that provides a modern platform for organizations to run and deploy their new and existing VM workloads. The solution allows for easy migration and management of traditional VMs onto a trusted, consistent, and comprehensive hybrid-cloud application platform.
Portworx	Portworx Enterprise 3.3.0	Portworx Enterprise is a leading cloud-native storage and data management platform for Kubernetes that is designed to enable enterprises to manage storage for their containerized stateful applications across various infrastructure environments.
Golden template	RHEL 9 default template	NA
Golden template size	30GB	NA
VM scheduler	STORK	NA

TABLE 3 Software components

Portworx sizing

To achieve optimal performance within modern virtualization environments, we recommend that you size the Portworx cluster. For these experiments, we used the recommended Portworx sizing:

- Cluster size: 29 nodes
- Portworx requirements
  - 8 CPU cores
  - 16GB memory

NOTE: Portworx currently supports up to 256 volume attachments per node, so be sure to take this into account when sizing your cluster and deploying VMs. The number of disk attachments across all of the VMs on the node cannot exceed this limit. This limit will be increased to 1,024 in an upcoming Portworx release.

To achieve this configuration, we updated the StorageCluster CRD. In **portworx.io/misc-args**, add the memory and CPU annotation as described in Portworx documentation. You will typically do this during Portworx installation (that is, before applying the spec generated from Portworx Central). If you already created the cluster, then you can edit the StorageCluster with the following annotations.

```
annotations:
  portworx.io/misc-args: ' -T px-storev2 --memory 17179869184 --cpus 8
```

For VM sizing, refer to the Maximum VMs per node applying the generated spec from

For the VMs created in the cluster, we configured a Linux bridge network on a secondary interface. First, we created the Linux bridge on the desired interface.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1
spec:
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge test
        type: linux-bridge
        state: up
        ipv4:
          enabled: false
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: ens1f1
              vlan: {}
```

Then, we created the net-attach-def network name, which the VM definitions configured as a Multus network attachment.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network
  annotations:
        k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
spec:
  config: '{
        "cniVersion": "0.3.1",
        "name": "bridge-network",
        "type": "cnv-bridge",
        "bridge": "br1",
        "macspoofchk": true
  }'
```

Finally, for this specific lab configuration, we wanted to enforce a certain DNS ordering so we configured a dns-resolver policy.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
 name: dns-ordering
spec:
  desiredState:
    dns-resolver:
      config:
        search:
        - bm.example.com
        server:
        - 198.18.10.1
        - 198.18.16.0
        - 10.1.x.x
```

Virtualization control plane tuning

We applied the recommended highBurst setting to support high VM burst creation rates.

```
oc patch -n openshift-cnv hco kubevirt-hyperconverged --type=json -p='[{"op": "add", "path": "/spec/
tuningPolicy", "value": "highBurst"}]'
```

Migration settings

We changed the live migration settings in order to increase the amount of parallel migrations across the cluster by editing the HyperConverged CR.

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

We also set the live migration configuration.

```
 liveMigrationConfig:
    completionTimeoutPerGiB: 800
    parallelMigrationsPerCluster: 25 # default 5
    parallelOutboundMigrationsPerNode: 5 # default 2
    progressTimeout: 150
```

Portworx storage class

We used the following storage class for the experiments.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: datavol-raw-immediate
parameters:
  repl: "3"
provisioner: pxd.portworx.com
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

Portworx installation

We followed the installation documentation to install Portworx Enterprise 3.3 in the cluster. Table 4 shows the Portworx Central configuration options chosen for these experiments. Other settings were left unchanged (default).

Platform	DAS/SAN
Kubernetes distribution	Red Hat OpenShift 4+ version with Kubernetes 1.30.7
Namespace	Portworx
etcd	Built in
Storage	Manually specified disks 1.7TiB /dev/sdb (metadata path) 1.7TiB /dev/sdc (drive/device) 2.9TiB /dev/nvme0n1 (drive/device)
PX-StoreV2	Selected
Network	Default settings/auto

TABLE 4 Portworx Central configuration options

Be sure to update **portworx.io/misc-args** before applying the generated spec from Portworx Central.

```
portworx.io/misc-args: ' -T px-storev2 --memory 17179869184 --cpus 8
```

Golden template spec

We created the golden template using the following spec, which uses the default RHEL 9 guest image.

```
apiVersion: cdi.kubevirt.io/v1beta1

kind: DataVolume

metadata:

  name: rhel9-manual-1

  namespace: openshift-virtualization-os-images

spec:

  source:

    registry:

      url: /docker://registry.redhat.io/rhel9/rhel-guest-image@
sha256:b81af29a801fcc2a2410b70f642dab0dbab961fb8c6cae7907ccd32efcfa289d

      pullMethod: node

  storage:

    storageClassName: datavol-raw-immediate

    accessModes:

      - ReadWriteMany

    resources:

      requests:

        storage: 30Gi
```

Sample VM spec

We started with the default RHEL 9 template provided by Red Hat OpenShift Virtualization templates and customized it to suit our environment—modifying the network settings, applying node selectors, and adjusting resource allocations. This customized template was then used in a Python script to automate the parallel creation of multiple VMs. Depending on your requirements, you can implement your own method for deploying multiple VMs at scale.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
 labels:
   app: <vm_name>
 name: <vm_name>
spec:
 running: false
 template:
   metadata:
     annotations:
       vm.kubevirt.io/workload: server
   spec:
     nodeSelector:
       kubernetes.io/hostname: "<node_name>"
     domain:
       cpu:
         cores: 2
       devices:
         disks:
           - bootOrder: 1
             disk:
               bus: virtio
             name: rootdisk
         interfaces:
           - name: br1
             bridge: {}
         networkInterfaceMultiqueue: true
         rng: {}
       features:
         acpi: {}
         smm:
           enabled: true
       resources:
         requests:
           memory: 2Gi
           cpu: 1
         limits:
           cpu: 2
     networks:
     - name: br1
       multus:
         networkName: bridge-network
     tolerations:
       - effect: NoSchedule
         key: node-role.kubernetes.io/master
         operator: Exists
     evictionStrategy: LiveMigrate
     terminationGracePeriodSeconds: 180
     volumes:
       - dataVolume:
           name: <root_disk_name>
         name: rootdisk
 dataVolumeTemplates:
   - metadata:
       name: <root_disk_name>
       annotations:
         cdi.kubevirt.io/cloneType: csi-clone
     spec:
       source:
         pvc:
           name: rhel-manual-1
           namespace: openshift-virtualization-os-images
       pvc:
         accessModes:
           - ReadWriteMany
         resources:
           requests:
             storage: 30Gi
         storageClassName: datavol_raw_immediate
         volumeMode: Block
```

Experiment results

This section reviews the results of the experiments performed to measure VM provisioning time, boot times, and live migration times.

VM provisioning

This experiment was performed to demonstrate the performance of Red Hat OpenShift Virtualization with Portworx when provisioning multiple VM images from the golden template using the csi-clone strategy.

We tested using different batch sizes ranging from 50 to 1,000 VMs, with VMs deployed on multiple nodes at once. We measured the time it takes for every VM to complete the cloning (that is, the time it takes from issuing the Create VM command to VM cloning completion). The VM will be in the Stopped state after cloning is complete, as the VM spec has spec.running: false.

Based on our experiment results, we can deploy 1,000 VMs on a cluster with 29 worker nodes in under 22 minutes. We expect improved performance for the same batch size on larger clusters with more available nodes.

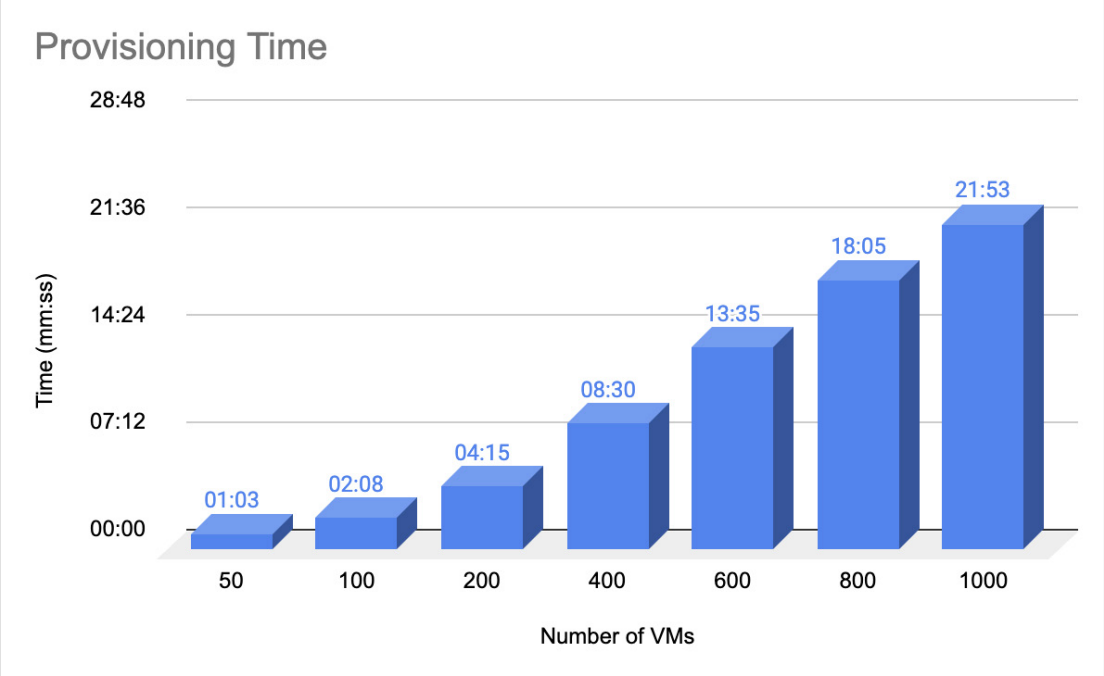

FIGURE 1 VM provisioning time

VM boot storm

We conducted a boot storm experiment to simulate a scenario where all VMs are initially in a powered off state—such as during a maintenance window—and then powered back on in parallel.

For this test, we measured the two durations: **SSH Ready time** and **Portworx Volume Ready time**.

SSH Ready time

The first duration we measured is the point from issuing the Power On command to the point when each VM becomes accessible via SSH (this includes time for the VM's operating system to boot up after the Portworx volumes are made available to the VM).

The VMs were evenly distributed across all nodes. As shown in Figure 2, we captured the 50%, 75%, and 90% boot storm times for different batch sizes.

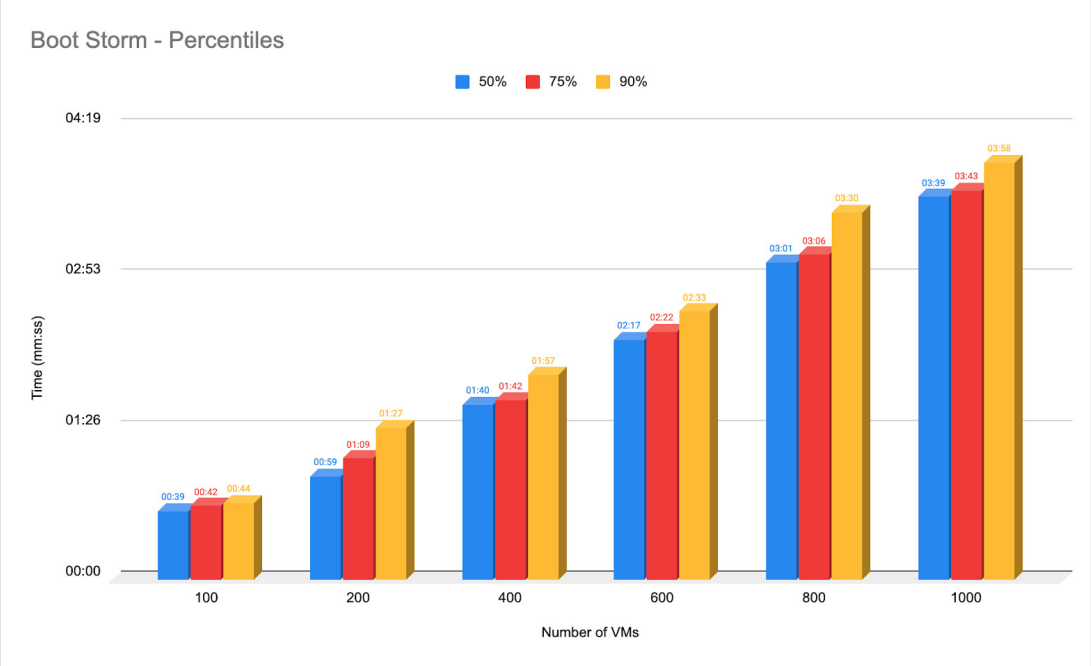

FIGURE 2 VM boot storm times by boot storm percentile

Figure 3 represents the total boot storm time.

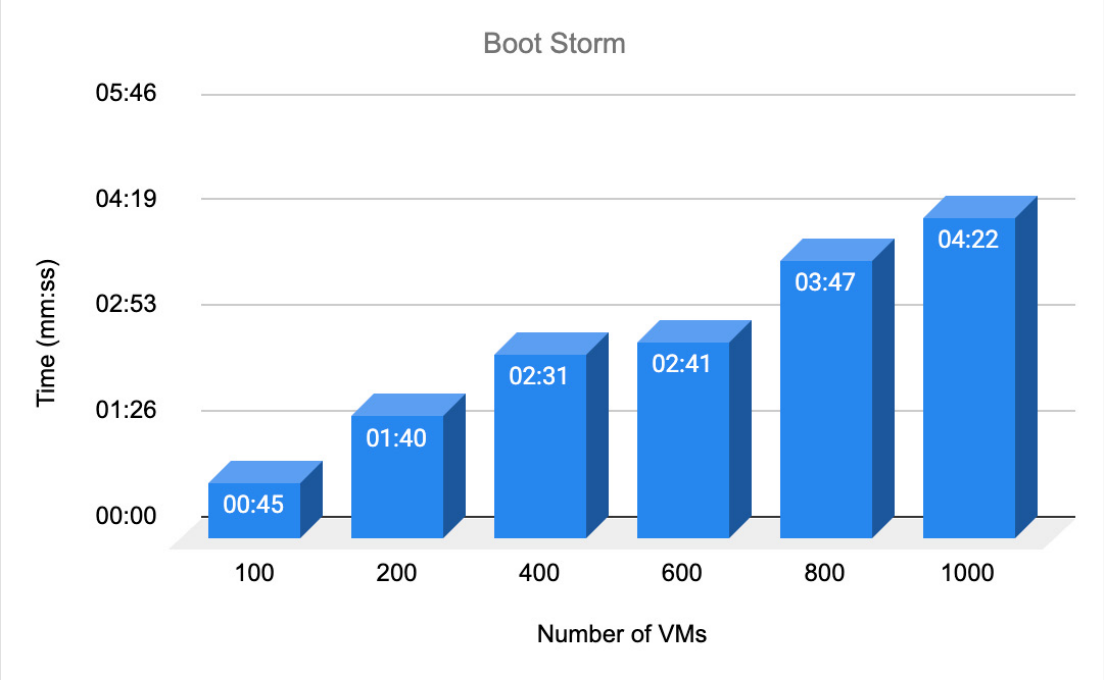

FIGURE 3 Total VM boot storm time by number of VMs

Based on the results, the boot times scale nearly linearly. We were able to power on 1,000 VMs across a 29-node worker cluster in approximately four minutes. Increasing the number of nodes in the cluster would allow for more even distribution of VMs, potentially resulting in faster boot times.

Portworx Volume Ready time

The other duration we measured is the point from issuing the Power On command to the point when each VM is in the Running state. This duration includes the time taken by Portworx to make the volumes available to the VMs. It took a total of three minutes and 41 seconds for all 1,000 VMs to reach the Running state.

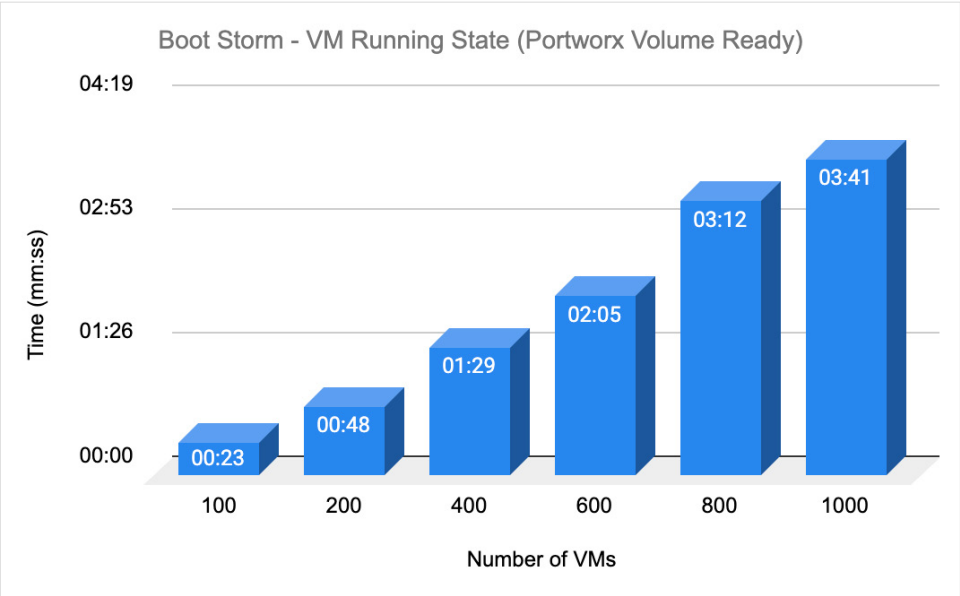

FIGURE 4 Boot storm time for Portworx Volume Ready

VM live migration

In this experiment, we performed live migration of up to 1,000 VMs, recording both the average migration time per VM and the total time to migrate all VMs. We had already set the **parallelMigrationsPerCluster** to **25** and **parallelOutboundMigrationsPerNode** to **5**. Prior to initiating the migrations, we removed the **nodeSelector** field from the VM specifications to allow unrestricted placement.

In each iteration, we migrated all the running VMs in the cluster, starting with 100 VMs and scaling up to 1,000. On average, each VM took approximately three seconds to migrate, and the entire set of 1,000 VMs successfully live migrated in about 18 minutes without any failures. Note that the **parallelMigrationsPerCluster** and **parallelOutboundMigrationsPerNode** parameters affect the overall migration times and can be configured as per your requirement. In a larger cluster with more nodes, this setting is expected to reduce the overall migration time for 1,000 VMs.

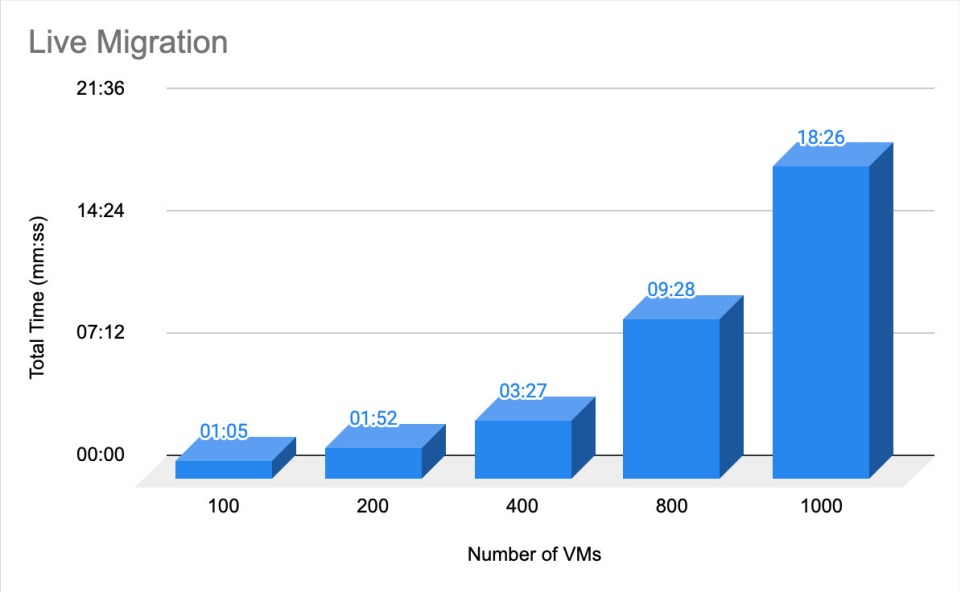

FIGURE 5 Total live migration time by number of VMs

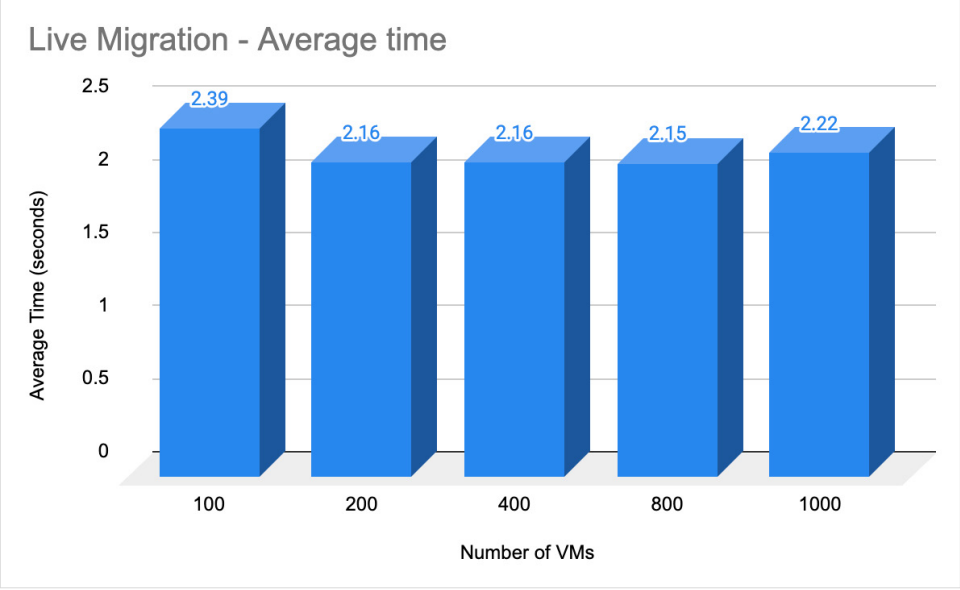

FIGURE 6 Average time per individual VM migration

Maximum VMs per node

In this experiment, we tested the capacity limits of the physical hardware (CPU and memory) by determining the maximum number of VMs that could be deployed on a single node.

We tested different resource configurations in the VM spec by updating resources.requests for memory and cpu, and we tested using various VM configurations and overcommit ratios. Each VM was provisioned with two disks, ensuring that storage consumption remained within allowable limits.

```
resources:
        requests:
          memory: 2Gi
          cpu: 500m
        limits:
          cpu: 2
```

VM sizing

Table 5 outlines the VM sizing configurations used in this experiment.

VM Size	vCPU Requested	vCPU Limit	Memory
X-small	500m (2:1 overcommit)	1	2GiB
Small	500m (4:1 overcommit)	2	8GiB
Small (config 2)	500m (4:1 overcommit)	2	4GiB
Medium	1 (4:1 overcommit)	4	24GiB
Large	8 (no overcommit)	8	48GiB

TABLE 5 VM sizing configurations

Based on the experiment results, we derived a formula with the following parameters to estimate the theoretical maximum number of VMs per node.

- **CPU_total:** total logical CPU cores on the node
- **MEM_total:** total memory on the node (in gibibytes)
- **vCPU_request:** CPU request per VM (in cores)
- **Mem_request:** memory request per VM (in gibibytes)
- **Max_VMs_CPU**: maximum number of VMs possible in theory on a node based on CPU
- **Max_VMs_Mem:** maximum number of VMs possible in theory on a node based on memory
- **Theoretical_Max_VMs:** Maximum number of VMs possible in theory on a node

Formula

```
Max_VMs_CPU = FLOOR (CPU_total / vCPU_request)
Max_VMs_Mem = FLOOR (MEM_total / Mem_request)

Theoretical_Max_VMs = MIN (Max_VMs_CPU, Max_VMs_Mem)
```

Example

```
CPU_total = 112
MEM_total = 503
vCPU_request = 1
Mem_request = 24

Then,

Max_VMs_CPU = 112 / 1 = 112
Max_VMs_Mem = 503 / 24 = 20 (floor)

Theoretical_Max_VMs = MIN (112, 20) = 20.
```

You can use this formula to estimate the maximum number of VMs that can be deployed per node based on available CPU and memory resources.

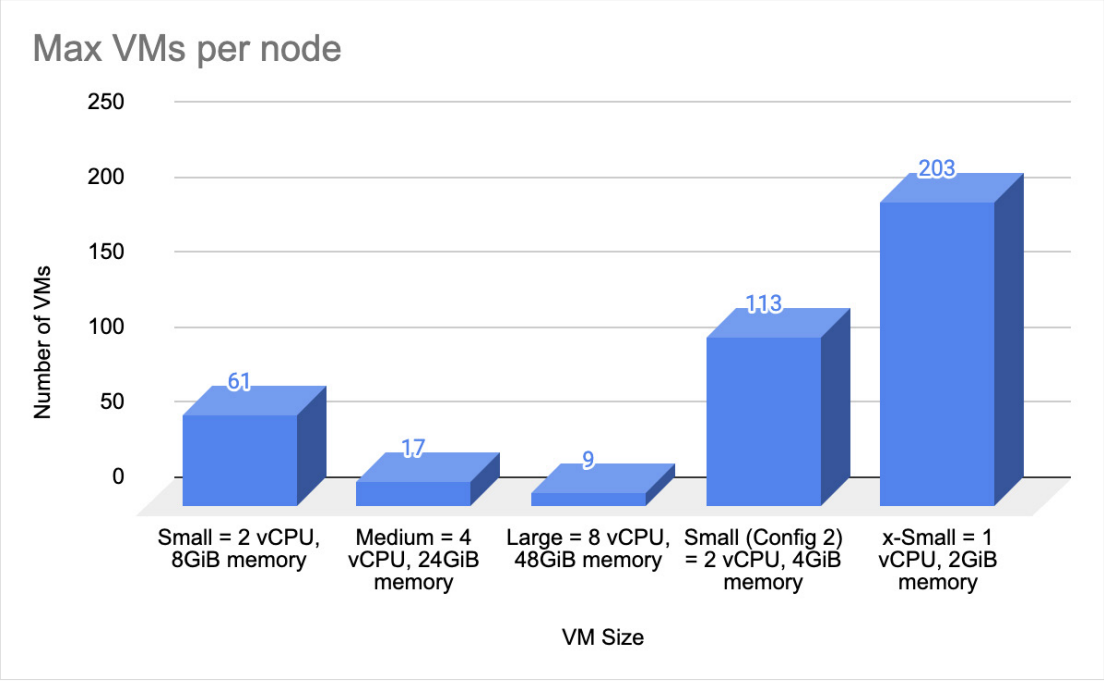

FIGURE 7 Maximum number of VMs per node

Conclusion

This document demonstrates the scalability, performance, and operational behavior of **Red Hat OpenShift Virtualization with Portworx** in a large-scale bare metal environment. We validated the robustness and efficiency of this solution in handling real-world infrastructure demands by running a series of experiments on **29 worker nodes** for up to **1,000 VMs**.

Our testing included:

- **Parallel VM provisioning** of up to 1,000 VMs completed in under 22 minutes.
- **Boot storm** simulation, where all VMs were powered on simultaneously, showing consistent linear boot times and a completion time of approximately four minutes for 1,000 VMs.
- **Live migration** of up to 1,000 VMs, with average migration times of approximately three seconds per VM, totaling approximately 18 minutes for migration of all VMs.
- **Maximum VM per node** calculations to approximate the maximum VMs possible per node.

Going beyond these experiments, we also stress-tested the environment under sustained I/O load using FIO workloads within the VMs. Even at a **simulated average cluster load of 65%**, system performance remained stable, and CPU and memory consumption for Portworx storage services did not become a bottleneck.

This performance evaluation provides a strong reference point for organizations considering **Red Hat OpenShift Virtualization with Portworx** for large-scale VM workloads. The learnings from these experiments are directly applicable to real-world production scenarios, offering guidance on effective deployment and migration strategies. These findings can support the successful planning, design, and operation of a Red Hat OpenShift Virtualization environment powered by Portworx.

Additional resources

Portworx

- Consult the Portworx installation instructions.
- Learn more about StorageCluster.
- Review the Portworx on Red Hat OpenShift Bare Metal reference architecture.

Red Hat OpenShift

- Consult the Red Hat OpenShift installation instructions.

ID6814-01-en-10/25