A Principled Technologies report: Hands-on testing. Real-world results.

Run more VMs and get better performance with VMware vSphere 8

Compared to Red Hat OpenShift Virtualization 4.16.2, the VMware virtualization platform supported 62% more database transactions and sustained consistent database performance while scaling up VMs due to efficient memory management methods

Peak workloads can strain physical data center resources, potentially causing performance loss without proper configuration. In virtual environments, the memory management strategies of memory oversubscription and memory overcommitment can help maximize the use of physical resources, increase VM density, and meet workload demand. Different virtualization solutions manage memory in various ways, and VMware® vSphere® 8 Update 3 offers a dedicated memory overcommitment feature for helping balance VM performance with density. But how does the memory management of a vSphere solution stack up against its competitors?

We compared the performance and VM density of a VMware vSphere 8 Update 3 solution to that of a Red Hat® OpenShift Virtualization 4.16.2 solution. After measuring online transaction processing (OLTP) performance at a baseline level using memory oversubscription but without using memory overcommitment, we began overcommitting memory on both solutions and increased the VM density until we saw significant (10 percent or more) performance degradation.

At every level of VM density, including the baseline, the vSphere solution supported more database transactions in NOPM than the OpenShift solution. Additionally, the OpenShift solution supported fewer VMs before experiencing significant degradation whereas the vSphere solution ran double the number of VMs over baseline before crossing the performance threshold. By supporting more VMs and better OLTP performance, organizations with a vSphere environment can meet more transactional database demand without adding more servers and licenses.

Higher VM density with consistent performance

Support 1.5x the VMs and maintain consistent performance for longer under load

Better total OLTP performance

Up to 62% more SQL Server new orders per minute (NOPM)

No configuration

vSphere enables memory overcommitment by default

Run more VMs and get better performance with VMware vSphere 8

October 2024

The science behind the report:

Run more VMs and get better performance with VMware vSphere 8

This document describes what we tested, how we tested, and what we found. To learn how these facts translate into real-world benefits, read the report Host more VMs and see better performance with VMware vSphere 8.0U3.

We concluded our hands-on testing on September 9, 2024. During testing, we determined the appropriate hardware and software configurations and applied updates as they became available. The results in this report reflect configurations that we finalized on September 4, 2024 or earlier. Unavoidably, these configurations may not represent the latest versions available when this report appears.

Our results

To learn more about how we have calculated the wins in this report, go to http://facts.pt/calculating-and-highlighting-wins. Unless we state otherwise, we have followed the rules and principles we outline in that document.

Table 1: Results of our testing.

	VMware vSphere 8 Update 3 solution	Red Hat OpenShift Virtualization 4.16.2 solution
Baseline and slight memory overcommitment scenarios		
New orders per minute (NOPM) without memory overcommitment (10 VMs)	1,435,069	1,124,586
NOPM with slight memory overcommitment (12 VMs)	1,521,155	1,046,467
Significant performance degradation (10 percent decrease or more) scenario		
NOPM when experiencing significant performance degradation	1,303,432	803,748
Number of supported VMs when experiencing significant performance degradation	20	13

System configuration information

Table 2: Detailed information on the systems we tested.

System configuration information	Dell™ PowerEdge™ R650
BIOS name and version	Dell 1.14.1
Non-default BIOS settings	Intel® Turbo Boost enabled, Virtualization enabled
Operating system name and version/build number	Red Hat® OpenShift v4.16.6 VMware® ESXi™ v8.0 Update 3
Date of last OS updates/patches applied	09/04/2024
Power management policy	Balanced
Processor	
Number of processors	2
Vendor and model	Intel Xeon® Gold 6330
Core count (per processor)	28
Core frequency (GHz)	2.00
Memory module(s)	
Total memory in system (GB)	256
Number of memory modules	8
Vendor and model	Hynix DDR4
Size (GB)	32
Type	PC4-3200
Speed (MHz)	3,200
Speed running in the server (MHz)	3,200
Local storage	
Number of drives	4
Drive vendor and model	Dell Ent NVMe CM6
Drive size (TB)	1.92
Drive information (speed, interface, type)	16 GT/s SAS, SSD
Network adapter	
Vendor and model	Broadcom® NetXtreme Gigabit Ethernet / Broadcom Adv. Dual 10Gb Ethernet
Number and type of ports	2x 10GbE
Driver version	22.91.5
Cooling fans	
Vendor and model	Dell STD cooling fan
Number of cooling fans	16
Power supplies	
Vendor and model	Dell 06C11WA03
Number of power supplies	2
Wattage of each (W)	1,400

How we tested

Setting up the Microsoft SQL Server 2022 VMs on VMware vSphere®

We performed the following steps in a fully updated VMware vSphere environment.

Installing and configuring the base SQL image

We first created a base VM that we later customized for our client and SQL Server VMs.

1. Log into the vCenter console.
2. Right-click the system under test, and select New Virtual Machine.
3. In Select a creation type, select Create a new virtual machine, and click Next.
4. In Select a name and folder, name the VM `gold-sql`, and click Next.
5. In Select a compute resource, click Next to accept defaults.
6. In Select storage, choose the storage you set up for your client VMs, and click Next.
7. In Select compatibility, click Next to accept defaults.
8. In Select a guest OS, choose Linux → Red Hat Enterprise Linux 9 (64-bit), and click Next.
9. Choose the following options for the new VM:
    - CPU: 6
    - RAM: 28 GB
    - Storage
        - Hard disk 1: 30 GB
        - Hard disk 2: 40 GB
        - Hard disk 3: 50 GB
    - Network
        - Network connection 1: Your network connection for the Internet (optional if your test network has a pre-configured gateway)
        - Network connection 2: Your test network connection
10. Verify that you chose the correct options, and click Next.
11. In Ready to complete, verify that you've applied the correct details, and click Next.
12. Right-click your new VM, and select Power → Power On.
13. Open a console to the new VM.
14. Attach a Red Hat Enterprise Linux 9.4 ISO to the VM, and accept defaults (with the following exceptions):
    - Select the minimal installation option
    - Configure a static IP for both network connections
15. Once the OS boots, log into the console for the system, and run a package update for the OS:

```
dnf update
```

16. Add the Microsoft SQL Server package repositories to your OS:

```
curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/9/mssql-
server-2022.repo
```

17. Install tools you will use later:

```
dnf install -y wget curl vim tar zip unzip nmon sysstat numactl ksh mssql-tools18 unixodbc-dev
```

18. Add the SQL Server tools to the console path:

```
echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bash_profile
echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bashrc
```

19. Change the IP address.
20. Change the SQL VM hostname:

```
hostnamectl set-hostname sql-gold.mssql.test
```

21. Download the SQL Server 2022 packages to the system:

```
dnf install -y mssql-server
```

22. Create the database and log directories:

```
mkdir -p /data
mkdir -p /logs/log
```

23. Format the database and log disks:

```
mkfs.xfs -f /dev/sdb
mkfs.xfs -f /dev/sdc
```

24. Open the filesystem table, and make the database and log disks attach to the OS on boot automatically:

```
vim /etc/fstab
```

25. In the filesystem table, add the following lines to the end:

```
/dev/sdb /data xfs rw,attr2,noatime 0 0
/dev/sdc /logs/log xfs rw,attr2,noatime 0 0
```

26. Apply the filesystem table to mount the database and log drives: (This also provides an opportunity to ensure that you edited your table correctly. If you did not, these commands will not work. Check `/etc/fstab`, and ensure you entered your disk information correctly)

```
mount -v /data
mount -v /logs/log/
```

27. Add new directories in the database drive for the database and backup, and add all permissions to the database, backup, and log directories:

```
mkdir -p /data/db
mkdir -p /data/backup
chmod -R 777 /data
chmod -R 777 /logs
```

28. Start the SQL Server database installation, and accept defaults (we chose `Password1` as our system administrator password):

```
/opt/mssql/bin/mssql-conf setup
```

29. (Optional) Verify that you have installed SQL Server:

```
systemctl status mssql-server --no-pager
```

30. For performance purposes, edit the SQL Server limits file:

```
vim /etc/security/limits.d/99-mssql-server.conf
```

31. Add the following lines to the end of the file, and save:

```
mssql hard nofile 32727
mssql soft nofile 16000
```

32. Add the following tuning options to SQL:

```
/opt/mssql/bin/mssql-conf traceflag 3979 on
/opt/mssql/bin/mssql-conf set control.writethrough 1
/opt/mssql/bin/mssql-conf set control.alternatewritethrough 0
```

33. Edit the Tuned settings for SQL:

```
vim /usr/lib/tuned/mssql/tuned.conf
```

34. Verify that the SQL Tuned file looks like the following:

```
#
# tuned configuration
#

[main]
summary=Optimize for MS SQL Server
include=throughput-performance

[cpu]
force_latency=5

[vm]
transparent_hugepage.defrag=always

[sysctl]
vm.swappiness = 1
vm.dirty_background_ratio = 3
vm.dirty_ratio = 80
vm.dirty_expire_centisecs = 500
vm.dirty_writeback_centisecs = 100
vm.transparent_hugepages=always
vm.max_map_count=1600000
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048576
kernel.numa_balancing=0
```

35. Make the SQL Tuned file executable:

```
chmod +x /usr/lib/tuned/mssql/tuned.conf
```

36. Change the Tuned profile to mssql:

```
tuned-adm profile mssql
```

37. Reboot the server:

```
reboot now
```

Setting up Red Hat OpenShift

Our OpenShift testbed involved three whitebox systems for infrastructure, a fourth whitebox assigned as a worker to behave as VM image storage (we will refer to this system as the "secondary worker"), and the system under test configured as a worker with all VMs assigned to run only on the system under test. We will describe how we achieved this later in the methodology.

We wrote this methodology with the assumption that the environment already contains a fully configured DHCP and DNS server and established DHCP and DNS reservations for the MAC addresses of the systems in the testbed. In our testbed, we used pfsense for DHCP and Microsoft DNS for DNS, but as long as you have properly configured your DHCP and DNS, the remaining configuration will remain the same.

During initial configuration, it is important to have only one active network connection for the systems you are using for your OpenShift testbed. You will need to add an extra network connection after the initial installation but adding that cable early can create configuration issues in OpenShift.

Using Assisted Installer to install Red Hat OpenShift and OpenShift Virtualization

1. Log into the Red Hat Hybrid Cloud Console.
2. Make sure you're on the OpenShift tab, and click Scale your applications.
3. Select Clusters, and click Next.
4. Click Create cluster.
5. At the top of the page, click Datacenter.
6. Under Assisted Installer, click Create Cluster.
7. In the cluster name field, type `memory-ocp` as the cluster name (this part is used as part of the FQDN for the hosts).
8. In the base domain field, enter the domain you are using (we used testbed.local).
9. Choose OpenShift Container Platform 4.16 as your version.
10. Click Install OpenShift Virtualization, and click Next.
11. In the cluster view, click Add hosts.
12. In the image file selection, choose Minimal image file: Provision with virtual media, add an SSH publick key, and click Generate Discovery ISO.
13. Download the discovery ISO.
14. Look into the BMC of the systems you are using for your OpenShift cluster.
15. Open a remote console.
16. Mount the discovery ISO to the system, and boot into the discovery ISO.
17. Wait for the installation to complete. Once it completes, log into your OpenShift cluster.

Configuring the infrastructure servers and system under test post-installation

Creating VM image storage partitions on the secondary worker

On a system with the OpenShift CLI, create VM image storage partitions by typing the following command (the target disk at the end of the command will change based on your secondary worker disk layout):

```
oc debug node/node4 -- bash -c "(echo g; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo
+40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G;
echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo
n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n;
echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo
; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ;
echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo
; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ;
echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo w) | fdisk /dev/sdf"
```

Creating VM disk partitions on the system under test

On a system with the OpenShift CLI, create VM OS, database, and log partitions on the system under test by typing the following commands (the target disk at the end of the command may change):

```
oc debug node/dell-sut -- bash -c "(echo g; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo
+40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G;
echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo
n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +60G; echo n;
echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo
; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ;
echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo
; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ;
echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo
+50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G;
echo w) | fdisk /dev/nvme1n1"
oc debug node/dell-sut -- bash -c "(echo g; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo
+40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G;
echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo
n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +60G; echo n;
echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo
; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ;
echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo
; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ;
echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo
+50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G;
echo w) | fdisk /dev/nvme2n1"
oc debug node/dell-sut -- bash -c "(echo g; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo
+40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G;
echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo
n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +40G; echo n; echo ; echo ; echo +60G; echo n;
echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo
; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ;
echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo ; echo +60G; echo n; echo ; echo
; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ;
echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo
+50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G; echo n; echo ; echo ; echo +50G;
echo w) | fdisk /dev/nvme3n1"
```

Attaching partitions as Local Storage persistent volumes (PVs)

1. Create a file called `vmi-disks.yaml`, and put the Appendix's `vmi-disks.yaml` contents into that file.
2. Attach the secondary worker's disk partitions as PVs by typing the following:

```
oc apply -f vmi-disks.yaml
```

3. Create a file called `sut-vmis.yaml`, and put the Appendix's `sut-vmis.yaml` contents into that file.
4. Attach disk partitions of the system under test as PVs by typing the following:

```
oc apply -f sut-vmis.yaml
```

5. Assign the secondary worker PVs as the default volume by typing the following:

```
oc patch storageclass vmi-disk-template-sc -p '{"metadata": {"annotations": {"storageclass.
kubernetes.io/is-default-class": "true"}}}'
```

Setting up SSH for VMs

1. In the OpenShift console, Navigate to Virtualization → Overview → Settings → User → Manage SSH keys.
2. Click Add public SSH key to project.
3. In the dropdown list, choose the project you intend to use for the SSH key.
4. In Public SSH key, click Add new.
5. Paste the contents of your previously configured SSH key public keyfile.
6. In Secret name, choose an appropriate name for your SSH key.
7. Make sure that Automatically apply this key to any new VirtualMachine you create in this project is checked, and click Save.

Configuring the VM networking

Before you perform any of the steps here, add a second network cable to your system under test's 10GbE network device and ensure it is on a network that can connect to the client VMs.

Installing the Kubernetes NMState Operator

1. In the OpenShift web console, navigate to Operators → OperatorHub.
2. In OperatorHub, search for and select the Kubernetes NMState Operator.
3. In the pop-up for the operator, accept defaults and click Install.
4. After the NMState operator finishes installing, navigate to Operators, Installed Operators, and click on Kubernetes NMState Operator.
5. In Kubernetes NMState Operator, under the NMState API, click Create instance.
6. Accept defaults, and click Create.
7. After the NMState API has been created, refresh the OpenShift web console to see new options.

Creating a NodeNetworkConfigurationPolicy (NNCP)

1. In the OpenShift web console, click Networking → NodeNetworkConfigruationPolicy.
2. In NodeNetworkConfigurationPolicy, click Create → From Form.
3. Check Apply this NodeNetworkConfigurationPolicy only to specific subsets of nodes using the node selector.
4. Choose the system under test.
5. In Policy Interfaces, leave defaults, but add the second 10GbE NIC to the Port field.

Creating a NetworkAttachDefinition (NAD)

1. In the OpenShift web console, click Networking → NetworkAttachmentDefinitions.
2. In Network Attachment Definitions, click Create Network Attachment Definition.
3. In Create network attachment definition, enter your preferred name, and select the following options:
   - Network Type: Linux bridge
   - Bridge name: br0
4. Click Create.

Configuring memory oversubscription

1. Create a privileged service account to run the memory oversubscription services by typing the following:

```
oc adm new-project wasp
oc create sa -n wasp wasp
oc create clusterrolebinding wasp --clusterrole=cluster-admin --serviceaccount=wasp:wasp
oc adm policy add-scc-to-user -n wasp privileged -z wasp
```

2. Create a file called `wasp-agent.yaml`, and put the Appendix's `wasp-agent.yaml` contents into that file: (Note that if your gold VM disk names are different, you will need to change the source file names.)
3. Apply the newly created file by typing the following:

```
oc create -f wasp-agent.yaml
```

4. Create a file called `wasp-kubeconfig.yaml`, and put the Appendix's `wasp-kubeconfig.yaml` contents into that file: (Note that if your gold VM disk names are different, you will need to change the source file names.)
5. Apply the newly created file by typing the following:

```
oc create -f wasp-kubeconfig.yaml
```

6. The servers will need to reboot to apply the new configuration, so wait until they have finished rebooting before continuing.
7. Create a file called `swap-config.yaml`, and put the Appendix's `swap-config.yaml` contents into that file: (Note that if your gold VM disk names are different, you will need to change the source file names.)
8. Apply the newly created file by typing the following:

```
oc create -f swap-config.yaml
```

9. Create a file called `dv-deploy.yaml`, and put the Appendix's `dv-deploy.yaml` contents into that file: (Note that if your gold VM disk names are different, you will need to change the source file names.)

10. Apply the newly created file by typing the following:

```
oc create -f swap-alert.yaml
```

11. Go to Operators → Installed Operators, and click OpenShift Virtualization.
12. Click OpenShift Virtualization Deployment.
13. Click kubevirt-hyperconverged.
14. In the YAML tab, find the `memoryOvercommitPercentage` value, and change it from 150 to 200.

Creating a SQL gold VM

1. Change the default StorageClass to a disk storage class on the system under test by typing the following:

```
oc patch storageclass vmi-disk-template-sc -p '{"metadata": {"annotations": {"storageclass.
kubernetes.io/is-default-class": "false"}}}'
 oc patch storageclass vmi-disk-1-sc -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/
is-default-class": "true"}}}'
```

2. In the VirtualMachines view, click Create → From template.
3. In Create new VirtualMachine, select Red Hat Enterprise Linux 9 VM.
4. In the template customization view, change the VirtualMachine name to an appropriate name, change the CPU and Memory to the correct amount (6 CPUs and 28 GiB of memory for our VMs), and click Customize VirtualMachine.
5. In Customize and create VirtualMachine, navigate to the Scheduling tab.
6. In the Scheduling tab, click No selector.
7. In the Node selector pop-up, click Add Label to specify qualifying nodes.
8. Name the Key `kubernetes.io/hostname`, add the Value to the name of your system under test, and click Save.
9. Click LiveMigrate.
10. In Eviction strategy, uncheck LiveMigrate, and click Save.
11. In Network interfaces, click Add network interface.
12. In Add network interface, name your NIC (or accept the default name), and make sure that the network is the external network you configured earlier, and click Save.
13. In Disks, click Add disk.
14. In Add disk, choose the following values:
    - Name: sql-gold-db [example value]
    - Source: Empty disk (blank)
    - PersistentVolumeClaim size: 40GiB
15. Click Save.
16. Click Add disk.
17. In Add disk, choose the following values:
    - Name: sql-gold-log [example value]
    - Source: Empty disk (blank)
    - PersistentVolumeClaim size: 50GiB
18. Click Save.
19. Click Create VirtualMachine.
20. After the VM has finished creating, change the default StorageClass back to the OS disk storage class on the infra server by typing the following command:

```
oc patch storageclass vmi-disk-1-sc -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/
is-default-class": "false"}}}'
oc patch storageclass vmi-disk-template-sc -p '{"metadata": {"annotations": {"storageclass.
kubernetes.io/is-default-class": "true"}}}'
```

21. Once the OS has booted, log into the console for the system, and run a package update for the OS:

```
dnf update
```

22. Add the Microsoft SQL Server package repositories to your OS:

```
curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/9/mssql-
server-2022.repo
```

23. Install tools for later use:

```
dnf install -y wget curl vim tar zip unzip nmon sysstat numactl ksh mssql-tools18 unixodbc-dev
```

24. Add the SQL Server tools to the console path:

```
echo ‘export PATH=”$PATH:/opt/mssql-tools18/bin”’ >> ~/.bash_profile
echo ‘export PATH=”$PATH:/opt/mssql-tools18/bin”’ >> ~/.bashrc
```

25. Change the IP address.
26. Change the SQL VM hostname:

```
hostnamectl set-hostname sql-gold.mssql.test
```

27. Download the SQL Server 2022 packages to the system:

```
dnf install -y mssql-server
```

28. Create the database and log directories:

```
mkdir -p /data
mkdir -p /logs/log
```

29. Format the database and log disks:

```
mkfs.xfs -f /dev/sdb
mkfs.xfs -f /dev/sdc
```

30. Open the filesystem table, and make the database and log disks attach to the OS on boot automatically:

```
vim /etc/fstab
```

31. In the filesystem table, add the following lines to the end:

```
/dev/sdb /data xfs rw,attr2,noatime 0 0
/dev/sdc /logs/log xfs rw,attr2,noatime 0 0
```

32. Apply the filesystem table to mount the database and log drives: (This also provides an opportunity to ensure that you edited your table correctly. If you did not, these commands will not work. Check `/etc/fstab`, and ensure you entered your disk information correctly.)

```
mount -v /data
mount -v /logs/log/
```

33. Add new directories in the database drive for the database and backup, and add all permissions to the database, backup, and log directories:

```
mkdir -p /data/db
mkdir -p /data/backup
chmod -R 777 /data
chmod -R 777 /logs
```

34. Start the SQL Server database installation, and accept defaults (we chose `Password1` as our system administrator password):

```
/opt/mssql/bin/mssql-conf setup
```

35. (Optional) Verify that you have installed SQL Server:

```
systemctl status mssql-server --no-pager
```

36. For performance purposes, edit the SQL Server limits file:

```
vim /etc/security/limits.d/99-mssql-server.conf
```

37. Add the following lines to the end of the file, and save:

```
mssql hard nofile 32727
mssql soft nofile 16000
```

38. Add the following tuning options to SQL:

```
/opt/mssql/bin/mssql-conf traceflag 3979 on
/opt/mssql/bin/mssql-conf set control.writethrough 1
/opt/mssql/bin/mssql-conf set control.alternatewritethrough 0
```

39. Edit the Tuned settings for SQL:

```
vim /usr/lib/tuned/mssql/tuned.conf
```

40. Verify that the SQL Tuned file looks like the following:

```
#
# tuned configuration
#

[main]
summary=Optimize for MS SQL Server
include=throughput-performance

[cpu]
force_latency=5

[vm]
transparent_hugepage.defrag=always

[sysctl]
vm.swappiness = 1
vm.dirty_background_ratio = 3
vm.dirty_ratio = 80
vm.dirty_expire_centisecs = 500
vm.dirty_writeback_centisecs = 100
vm.transparent_hugepages=always
vm.max_map_count=1600000
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048576
kernel.numa_balancing=0
```

41. Make the SQL Tuned file executable:

```
chmod +x /usr/lib/tuned/mssql/tuned.conf
```

42. Change the Tuned profile to mssql:

```
tuned-adm profile mssql
```

43. Reboot the server:

```
reboot now
```

Setting up the HammerDB clients

We ran the HammerDB clients as VMs from a separate vSphere server. We monitored the separate vSphere server during testing to ensure there were no bottlenecks. Bottlenecks would indicate that the client VMs were affecting the results.

Installing and configuring the client image

1. Log into the infrastructure vCenter console.
2. Right-click the infrastructure host, and select New Virtual Machine.
3. In Select a creation type, select Create a new virtual machine, and click Next.
4. In Select a name and folder, name the VM `client-VM`, and click Next.
5. In Select a compute resource, click Next to accept defaults.
6. In Select storage, choose the storage you set up for your client VMs, and click Next.
7. In Select compatibility, click Next to accept defaults.
8. In Select a guest OS, choose Linux → Red Hat Enterprise Linux 9, and click Next.
9. Choose the following options for the new VM:
    - CPU: 8
    - RAM: 8 GB
    - Hard disk: 60 GB
    - Network
        - Network connection 1: Your network connection for the Internet (optional if your test network has a pre-configured gateway)
        - Network connection 2: Your test network connection
10. Verify that you chose the correct options, and click Next.
11. In Ready to complete, verify that you've applied the correct details, and click Next.
12. Right-click your new VM, and select Power → Power On.
13. Open a console to the new VM.
14. Attach a Red Hat Enterprise Linux 9.4 ISO to the VM, and accept defaults (with the following exceptions):
    - Select the minimal installation option
    - Configure a static IP for both network connections
15. Once the OS boots, log into the console for the system, and run a package update for the OS:

```
dnf update -y
```

16. Add the Microsoft SQL Server package repositories to your OS:

```
curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/9/mssql-
server-2022.repo
```

17. Install tools for later use:

```
dnf install -y wget curl vim tar zip unzip nmon sysstat numactl ksh mssql-tools18 unixodbc-dev
```

18. Add the SQL Server tools to the console path:

```
echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bash_profile
echo 'export PATH="$PATH:/opt/mssql-tools18/bin"' >> ~/.bashrc
```

19. Change the IP address.
20. Change the client VM hostname:

```
hostnamectl set-hostname client-gold.mssql.test
```

21. Download and untar HammerDB:

```
wget https://github.com/TPC-Council/HammerDB/releases/download/v4.11/HammerDB-4.11-Linux.tar.gz
tar -zxvf HammerDB-4.11-Linux.tar.gz
```

Initializing the HammerDB TPROC-C database and backing up the database

We used a combination of Microsoft SQL Server Management Studio (SMSS) and the Red Hat Enterprise Linux command line to perform the following steps.

1. Open SMSS, and connect to the SQL Server installation.
2. In SMSS, right-click Databases, and select New Database.
3. In the New Database window, name your database `tpcc`, and give the database four database files and one log file. Ensure the database files are in `/data/db/`, and the log file is in `/logs/log/`.
4. In the Options tab, change the Recovery mode to Simple, and click OK.
5. Log into the client VM via SSH, and navigate to the HammerDB directory:

```
cd HammerDB-4.11
```

6. Open the HammerDB CLI:

```
./hammerdbcli
```

7. Inside the HammerDB CLI, set the benchmark to Microsoft SQL and TPROC-C:

```
dbset db mssqls
dbset bm tpcc
```

8. Configure the connection details for the SQL server:

```
diset connection mssqls_linux_server 192.168.0.20
diset connection mssqls_uid sa
diset connection mssqls_pass Password1
diset connection mssqls_trust_server_cert true
```

9. Configure the size of the database and the number of users to create the database:

```
diset tpcc mssqls_count_ware 250
diset tpcc mssqls_num_vu 4
```

10. Change the HammerDB client so it doesn't use locally cached data when initializing: (Note: this is for compatibility purposes with our setup. If you have configured your setup to use locally cached data, you can leave this at default):

```
diset tpcc mssqls_use_bcp false
```

11. Start the database creation:

```
buildschema
```

12. While the database is initializing, create a TPROC-C automation file for HammerDB:

```
vi tproc-c.tcl
```

13. Inside the tproc-c.tcl, add the following lines:

```
dbset db mssqls
dbset bm TPC-C

diset connection mssqls_linux_server 192.168.0.21
diset connection mssqls_linux_authent sql
diset connection mssqls_uid sa
diset connection mssqls_pass Password1

diset tpcc mssqls_count_ware 250
diset tpcc mssqls_use_bcp false
diset tpcc mssqls_total_iterations 1000000000
diset tpcc mssqls_driver timed
diset tpcc mssqls_rampup 10
diset tpcc mssqls_duration 20
diset tpcc mssqls_allwarehouse false

loadscript
puts “TEST STARTED”
vuset vu 16
vuset logtotemp 1
vucreate
tcstart
tcstatus
set jobid [ vurun ]
vudestroy
tcstop
puts “TEST COMPLETE”
```

14. After you create the database, use SSMS to log into the SQL server.
15. Right-click the tpcc database, and select Properties.
16. In the Files tab of Database Properties, click the log file, and expand it to 50GB.
17. In the Options tab of Database Properties, change Recovery model to full, and click OK.
18. Right-click the tpcc database, and select Tasks → Back Up.
19. In the Back Up Database window, change the destination to `/data/backup/backup.bak`.
20. In the Backup Options tab of the Back Up Database window, set backup compression to Compress backup, and click OK.
21. After a few minutes, the database backup will complete.
22. Shut down the HammerDB client and SQL VMs.

Cloning VMs to prepare for testing

Cloning and configuring the HammerDB client VMs

1. Right-click the gold client VM, and select Clone → Clone to Virtual Machine.
2. In Select a name and folder, enter the name for the first HammerDB client, and click Next.
3. In Select a compute resource, select your infrastructure host, and click Next.
4. In Select storage, select your client VM storage, and click Next.
5. In Select clone options, accept defaults, and click Next.
6. In Ready to complete, verify your options, and click Finish.
7. When the VM has finished cloning, log into it, and make the IP address of the VM unique
8. Change the VM hostname:

```
sudo hostnamectl set-hostname hammerdb-01.mssql.test
```

9. Edit the HammerDB automation file to target the correct SQL Server:

```
vi HammerDB-4.11/tproc-c.tcl
```

10. Complete steps 1 through 9 for all remaining clients.

Cloning and configuring the vSphere SQL Server host VMs

Perform the following steps only after the HammerDB clients have initialized the SQL database on the gold VM, as described in the Initializing the SQL database section.

1. Right-click the gold SQL VM, and select Clone → Clone to Virtual Machine.
2. In Select a name and folder, enter the name for the first SQL Server VM, and click Next.
3. In Select a compute resource, select your system under test host, and click Next.
4. In Select storage, select Configure per Disk.
5. Clone the configuration file, operating system, and log disks to one drive, and clone the database disk to a second drive. Keep track of which drives you used—when cloning new VMs, we recommend spreading out the OS, log, and database drives to reduce resource contention. Ensure the log and database drives are thick provision eager zeroed, and click Next.
6. In Select clone options, accept defaults, and click Next.
7. In Ready to complete, verify your options, and click Finish.
8. When the VM has finished cloning, log into it, and make the IP address of the VM unique.
9. Change the VM hostname:

```
hostnamectl set-hostname mssql-01.mssql.test
```

10. Complete steps 1 through 9 for all remaining SQL Server VMs.

Cloning and configuring the OpenShift VMs

Perform the following steps only after the HammerDB clients have initialized the SQL database on the gold VM, as described in the Initializing the SQL database section.

1. Power down the gold VM.
2. Create a file called `dv-deploy.yaml`, and put the Appendix's `dv-deploy.yaml` contents into that file. (If your gold VM disk names are different, you will need to change the source file names).
3. Clone out the SQL VM disks by typing the following command:

```
oc create -f dv-deploy.yaml
```

4. In the OpenShift console, navigate to VirtualMachines, and click Create → From template.
5. In Create new VirtualMachine, select Red Hat Enterprise Linux 9 VM.
6. In the template customization view, change the VirtualMachine name to an appropriate name, change the CPU and Memory to the correct amount (6 CPUs and 28 GiB of memory for our VMs), and click Customize VirtualMachine.
7. In Customize and create VirtualMachine, navigate to the Scheduling tab.
8. In the Scheduling tab, click No selector.
9. In the Node selector pop-up, click Add Label to specify qualifying nodes.
10. Name the Key kubernetes.io/hostname, add the Value to the name of your system under test, and click Save.
11. Click LiveMigrate.
12. In Eviction strategy, uncheck LiveMigrate, and click Save.
13. In Network interfaces, click Add network interface.
14. In Add network interface, name your NIC (or accept the default name), and make sure that the network is the external network you configured earlier, and click Save.
15. In Disks, delete the starting rootdisk.
16. Click Add disk.
17. In Add disk, choose the following values:
    - Make sure to check Use this disk as a boot source
    - Name: sql-1-os [example value]
    - Source: PVC
    - PersistentVolumeClaim name: `sql-gold-os`

Click Save.

18. Complete steps 16 through 18 for the database and log disks but leave Use this disk as a boot source unchecked.
19. Click Create VirtualMachine.
20. Complete steps 4 through 20 for the remaining VMs, but make sure that Start this VirtualMachine after creation is unchecked or the disk cloning will fail.

21. After your first VM finishes cloning, power on the next VM. Complete this step for all remaining VMs, ensuring that only one VM is cloning at a time.
22. After a VM has finished cloning, log into it, and make the IP address of the VM unique.
23. Change the VM hostname:

```
hostnamectl set-hostname mssql-01.mssql.test
```

24. Complete steps 23 and 24 for all remaining SQL Server VMs.

Running the HammerDB benchmark

Performing a test run

1. On all HammerDB client VMs, navigate to the HammerDB directory:

```
cd HammerDB-4.11
```

2. On all HammerDB client VMs, type the following command, but do not execute it:

```
./hammerdbcli auto ~/HammerDB-4.11/tpcroc-c.tcl
```

3. On all HammerDB client VMs, simultaneously execute the command from step 2.

Performing a restore after a test run

We performed the following steps on all database VMs after completing a test run.

1. Shut down the database VM.
2. Power on the database VM.
3. Use SSMS to log into the VM database.
4. Right-click the tpcc database, and select Delete.
5. In Delete Object, check Close existing connections, and click OK.
6. Right-click Databases, and select Restore Files and Filegroups.
7. In Destination to restore, type `tpcc`.
8. In Source for restore, select From Device.
9. In Select backup devices, navigate to `/data/backup/backup.bak`, and click OK.
10. Verify that the system is restoring the files to their appropriate locations, and click OK.
11. When the database has restored, click OK.
12. Complete steps 1 through 10 for the remaining database VMs.

Scripts used in the test

vmi-disks.yaml

```
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: vmi-disk-template
  namespace: openshift-local-storage
spec:
  logLevel: Normal
  managementState: Managed
  nodeSelector:
    nodeSelectorTerms:
    - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
        - node4
  storageClassDevices:
  - devicePaths:
    - /dev/sdf1
    - /dev/sdf2
    - /dev/sdf3
    - /dev/sdf4
    - /dev/sdf5
    - /dev/sdf6
    - /dev/sdf7
    - /dev/sdf8
    - /dev/sdf9
    - /dev/sdf10
    - /dev/sdf11
    - /dev/sdf12
    - /dev/sdf13
    - /dev/sdf14
    - /dev/sdf15
    - /dev/sdf16
    - /dev/sdf17
    - /dev/sdf18
    - /dev/sdf19
    - /dev/sdf20
    - /dev/sdf21
    - /dev/sdf22
    - /dev/sdf23
    - /dev/sdf24
    - /dev/sdf25
    - /dev/sdf26
    storageClassName: vmi-disk-template-sc
    volumeMode: Filesystem
```

sut-vmis.yaml

```
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: vmi-disk-1
  namespace: openshift-local-storage
spec:
  logLevel: Normal
  managementState: Managed
  nodeSelector:
    nodeSelectorTerms:
    - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
        - dell-sut
  storageClassDevices:
  - devicePaths:
    - /dev/nvme1n1p1
    - /dev/nvme1n1p2
    - /dev/nvme1n1p3
    - /dev/nvme1n1p4
    - /dev/nvme1n1p5
    - /dev/nvme1n1p6
    - /dev/nvme1n1p7
    - /dev/nvme1n1p8
    - /dev/nvme1n1p9
    - /dev/nvme1n1p10
    - /dev/nvme1n1p11
    - /dev/nvme1n1p12
    - /dev/nvme1n1p13
    - /dev/nvme1n1p14
    - /dev/nvme1n1p15
    - /dev/nvme1n1p16
    - /dev/nvme1n1p17
    - /dev/nvme1n1p18
    - /dev/nvme1n1p19
    - /dev/nvme1n1p20
    - /dev/nvme1n1p21
    - /dev/nvme1n1p22
    - /dev/nvme1n1p23
    - /dev/nvme1n1p24
    - /dev/nvme1n1p25
    - /dev/nvme1n1p26
    - /dev/nvme1n1p27
    - /dev/nvme1n1p28
    - /dev/nvme1n1p29
    - /dev/nvme1n1p30
    storageClassName: vmi-disk-1-sc
    volumeMode: Filesystem
---
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: vmi-disk-2
  namespace: openshift-local-storage
spec:
  logLevel: Normal
  managementState: Managed
  nodeSelector:
    nodeSelectorTerms:
    - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
        - dell-sut
  storageClassDevices:
  - devicePaths:
    - /dev/nvme2n1p1
    - /dev/nvme2n1p2
    - /dev/nvme2n1p3
    - /dev/nvme2n1p4
    - /dev/nvme2n1p5
    - /dev/nvme2n1p6
    - /dev/nvme2n1p7
    - /dev/nvme2n1p8
    - /dev/nvme2n1p9
    - /dev/nvme2n1p10
    - /dev/nvme2n1p11
    - /dev/nvme2n1p12
    - /dev/nvme2n1p13
    - /dev/nvme2n1p14
    - /dev/nvme2n1p15
    - /dev/nvme2n1p16
    - /dev/nvme2n1p17
    - /dev/nvme2n1p18
    - /dev/nvme2n1p19
    - /dev/nvme2n1p20
    - /dev/nvme2n1p21
    - /dev/nvme2n1p22
    - /dev/nvme2n1p23
    - /dev/nvme2n1p24
    - /dev/nvme2n1p25
    - /dev/nvme2n1p26
    - /dev/nvme2n1p27
    - /dev/nvme2n1p28
    - /dev/nvme2n1p29
    - /dev/nvme2n1p30
    storageClassName: vmi-disk-2-sc
    volumeMode: Filesystem
---
apiVersion: local.storage.openshift.io/v1
kind: LocalVolume
metadata:
  name: vmi-disk-3
  namespace: openshift-local-storage
spec:
  logLevel: Normal
  managementState: Managed
  nodeSelector:
    nodeSelectorTerms:
    - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
        - dell-sut
  storageClassDevices:
  - devicePaths:
    - /dev/nvme3n1p1
    - /dev/nvme3n1p2
    - /dev/nvme3n1p3
    - /dev/nvme3n1p4
    - /dev/nvme3n1p5
    - /dev/nvme3n1p6
    - /dev/nvme3n1p7
    - /dev/nvme3n1p8
    - /dev/nvme3n1p9
    - /dev/nvme3n1p10
    - /dev/nvme3n1p11
    - /dev/nvme3n1p12
    - /dev/nvme3n1p13
    - /dev/nvme3n1p14
    - /dev/nvme3n1p15
    - /dev/nvme3n1p16
    - /dev/nvme3n1p17
    - /dev/nvme3n1p18
    - /dev/nvme3n1p19
    - /dev/nvme3n1p20
    - /dev/nvme3n1p21
    - /dev/nvme3n1p22
    - /dev/nvme3n1p23
    - /dev/nvme3n1p24
    - /dev/nvme3n1p25
    - /dev/nvme3n1p26
    - /dev/nvme3n1p27
    - /dev/nvme3n1p28
    - /dev/nvme3n1p29
    - /dev/nvme3n1p30
    storageClassName: vmi-disk-3-sc
    volumeMode: Filesystem
```

wasp-agent.yaml

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: wasp-agent
  namespace: wasp
  labels:
    app: wasp
    tier: node
spec:
  selector:
    matchLabels:
      name: wasp
  template:
    metadata:
      annotations:
        description: >-
          Configures swap for workloads
      labels:
          name: wasp
    spec:
      serviceAccountName: wasp
      hostPID: true
      hostUsers: true
      terminationGracePeriodSeconds: 5
      containers:
        - name: wasp-agent
          image: >-
            registry.redhat.io/container-native-virtualization/wasp-agent-rhel9:v4.16
          imagePullPolicy: Always
          env:
          - name: “FSROOT”
            value: “/host”
          resources:
            requests:
              cpu: 100m
              memory: 50M
          securityContext:
            privileged: true
          volumeMounts:
          - name: host
            mountPath: “/host”
      volumes:
      - name: host
```

```
        hostPath:
          path: "/"
      priorityClassName: system-node-critical
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 10%
      maxSurge: 0
status: {}
```

wasp-kubeconfig.yaml

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ''  # MCP
      #machine.openshift.io/cluster-api-machine-role: worker # machine
      #node-role.kubernetes.io/worker: '' # node
  kubeletConfig:
    failSwapOn: false
    evictionSoft:
      memory.available: "1Gi"
    evictionSoftGracePeriod:
      memory.available: "10s"
```

swap-config.yaml

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 90-worker-swap
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
      - contents: |
          [Unit]
          Description=Provision and enable swap
          ConditionFirstBoot=no

          [Service]
          Type=oneshot
          Environment=SWAP_SIZE_MB=262144
          ExecStart=/bin/sh -c "sudo dd if=/dev/zero of=/var/tmp/swapfile count=${SWAP_
SIZE_MB} bs=1M && \
          sudo chmod 600 /var/tmp/swapfile && \
          sudo mkswap /var/tmp/swapfile && \
          sudo swapon /var/tmp/swapfile && \
          free -h && \
          sudo systemctl set-property --runtime system.slice MemorySwapMax=0
IODeviceLatencyTargetSec=\"/ 50ms\""

          [Install]
          RequiredBy=kubelet-dependencies.target
        enabled: true
        name: swap-provision.service
```

swap-alert.yaml

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: wasp-alerts
  namespace: openshift-monitoring
spec:
  groups:
  - name: wasp.rules
    rules:
    - alert: NodeSwapping
      annotations:
        description: Node {{ $labels.instance }} is swapping at a rate of {{ printf
"%.2f" $value }} MB/s
        runbook_url: https://github.com/openshift-virtualization/wasp-agent/tree/main/runbooks/alerts/
NodeSwapping.md
        summary: A node is swapping memory pages
      expr: |
        # In MB/s
        irate(node_memory_SwapFree_bytes{job="node-exporter"}[5m]) / 1024^2 > 0
      for: 1m
      labels:
        severity: critical
```

dv-deploy.yaml

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-1-os
spec:
  source:
    pvc:
      namespace: "default"
      name: "rhel9-gold"
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-1-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-1-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-2-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-2-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-2-db
spec:
  source:
    pvc:
      namespace: "default"
      name: "rhel9-gold-db"
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-3-os
spec:
  source:
    pvc:
      namespace: "default"
      name: "rhel9-gold"
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-3-log
spec:
  source:
    pvc:
      namespace: "default"
      name: "rhel9-gold-log"
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-3-db
spec:
  source:
    pvc:
      namespace: "default"
      name: "rhel9-gold-db"
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
```

```
kind: DataVolume
metadata:
  name: sql-4-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-4-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-4-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-5-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-2-sc
```

```
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-5-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-5-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-6-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-6-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-6-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-7-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-7-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-7-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-8-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-8-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-8-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-9-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-9-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-9-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-10-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-10-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-10-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-11-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-11-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-11-db
```

```
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-12-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-12-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-12-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
```

```
metadata:
  name: sql-13-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-13-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-13-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-14-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-2-sc
---
```

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-14-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-14-db
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-db”
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-15-os
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold”
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-3-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-15-log
spec:
  source:
    pvc:
      namespace: “default”
      name: “rhel9-gold-log”
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-15-db
spec:
  source:
    pvc:
      namespace: "default"
      name: "rhel9-gold-db"
  storage:
    resources:
      requests:
        storage:         42Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-16-os
spec:
  source:
    pvc:
      namespace: "default"
      name: "rhel9-gold"
  storage:
    resources:
      requests:
        storage:         32Gi
    storageClassName:  vmi-disk-1-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-16-log
spec:
  source:
    pvc:
      namespace: "default"
      name: "rhel9-gold-log"
  storage:
    resources:
      requests:
        storage:         52Gi
    storageClassName:  vmi-disk-2-sc
---
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: sql-16-db
spec:
  source:
    pvc:
      namespace: "default"
```

```
        name: “rhel9-gold-db”
    storage:
      resources:
        requests:
          storage:         42Gi
      storageClassName:  vmi-disk-3-sc
```

Read the report at https://facts.pt/ebLHw9P ▶

This project was commissioned by Broadcom.

Principled Technologies®

Facts matter.®