Performance and resilience for PostgreSQL

Crunchy Data and Red Hat OpenShift Container Storage

Table of contents

Highlights

Choose the PostgreSQL data protection and business continuity solution that matches your business needs.

Employ the Crunchy PostgreSQL Operator for database redundancy, failover, and workload scalability.

Deploy Red Hat OpenShift Container Storage to provide storage resiliency for Crunchy Data PostgreSQL.

Experience consistent database performance whether replicating within a single availability zone or across multiple availability zones.

Executive summary

As critical PostgreSQL database applications embrace cloud and container environments, retaining enterprise functionality remains a vital focus. Providing for business continuity is essential, as is protecting data against loss. Moreover, applications and the databases they depend on must continue to perform consistently in the face of any infrastructure failures without significant degradations.

Together, technologies from Crunchy Data and Red Hat can enable organizations to deliver resilience for critical PostgreSQL applications. Crunchy Data provides a high-availability PostgreSQL solution via the Crunchy PostgreSQL Operator for Red Hat® OpenShift®, offering replication, failover, and workload scalability at the database level. Designed for and with Red Hat OpenShift, Red Hat OpenShift Container Storage offers cloud-native, software-defined storage that can provide storage redundancy and failover for PostgreSQL and other applications across multiple Amazon Web Services (AWS) Availability Zones – without compromising performance.

Business continuity for PostgreSQL applications

PostgreSQL is one of the most popular databases in the Kubernetes domain, and organizations are using it more. Designers can choose from sophisticated mechanisms to provide resilience and business continuity for PostgreSQL through the database itself or through a resilient storage layer. Options include:

- **The Crunchy PostgreSQL Operator for Red Hat OpenShift.** The Crunchy PostgreSQL Operator for Red Hat OpenShift provides a means to create several PostgreSQL pods (multi-instance) with a single, primary read-write PostgreSQL instance and multiple read-only replica PostgreSQL instances. The operator offers database resilience and workload scalability through multiple, secondary, read-only instances. When a failure occurs, the Crunchy PostgreSQL Operator can promote a secondary instance to primary and continued operations.
- **Red Hat OpenShift Container Storage.** Based on container-native Ceph®, Rook, and Noobaa technologies, Red Hat OpenShift Container Storage can provide storage resiliency for PostgreSQL deployments where replicating the database is not required. OpenShift Container Storage can offer storage redundancy to a PostgreSQL pod (single instance). The same storage layer deployment can also provide resilient and scalable storage to other databases or other applications that need to store and access block, file, or object data.

As shown in Figure 1, these two approaches can be used individually as befits the needs of the application and business case or combined for maximum resiliency.

PostgreSQL is an enterprise-class, relational database management system with a proven architecture and more than 30 years of active development.

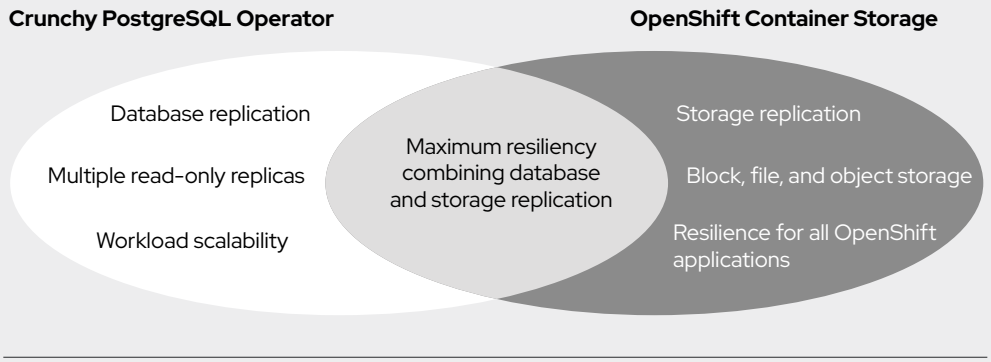

Figure 1. The Crunchy PostgreSQL Operator for Red Hat OpenShift and Red Hat OpenShift Container Storage can be used together or separately to achieve desired levels of resiliency.

This document describes testing conducted by Red Hat and Crunchy Data engineers. PostgreSQL database clusters were evaluated under load with induced failover events. The testing was conducted on AWS infrastructure using storage-optimized AWS instances and support for direct-attached storage in OpenShift Container Storage 4.3. Testing showed that OpenShift Container Storage can achieve good performance while still maintaining 3x data replication, even when the application and the storage are replicated across multiple AWS Availability Zones for high availability and business continuity.

PostgreSQL running on containers

Containers are highly portable, enabling container-based applications to be deployed on multiple platforms and operating systems (OS). The decoupling that containers provide allows developers to quickly deploy and manage applications at scale. With the rise in popularity in DevOps and container technologies, developers are looking to deploy applications and database code using continuous integration/continuous delivery (CI/CD). Many container-based, cloud-native applications use relational databases for persistent storage.

Crunchy Data is a leading provider of trusted open source PostgreSQL technology, support, and enterprise training. Crunchy Data offers enterprise-ready, open source solutions, including Crunchy PostgreSQL for Kubernetes, a cloud-agnostic, Kubernetes-native, enterprise PostgreSQL solution.

Crunchy PostgreSQL Operator overview

PostgreSQL is ACID-compliant (referring to fundamental properties of atomicity, consistency, isolation, and durability). It also has many built-in features to help with the ongoing operational management and durability of a database cluster. However, the core PostgreSQL database server leaves high-availability implementations to external applications.

Crunchy Data developed the PostgreSQL Operator as an open source controller that runs within a Kubernetes cluster, providing a means to deploy and manage production-ready, Kubernetes-native, PostgreSQL clusters at scale. By abstracting and automating specific operational tasks, the Crunchy PostgreSQL Operator reduces the level of domain-specific knowledge necessary to perform advanced tasks such as deploying highly available clusters, managing disaster recovery settings, and applying PostgreSQL configurations.

From an enterprise user perspective, the Crunchy PostgreSQL Operator lets organizations confidently deploy PostgreSQL clusters that address common enterprise requirements such as auto-provisioning, high-availability, disaster recovery, monitoring, and management. The Crunchy PostgreSQL Operator automates and simplifies deploying and managing open source PostgreSQL clusters on Kubernetes and other Kubernetes-enabled platforms with features that include:

- PostgreSQL cluster provisioning.
- High availability.
- Disaster recovery.
- Monitoring.
- PostgreSQL user management.
- Upgrade management.
- Connection pooling.

The Crunchy PostgreSQL Operator makes it easy to rapidly deploy your PostgreSQL-as-a-Service on Kubernetes-enabled platforms and allows further customizations, including:

- Selecting different storage classes for your primary, replica, and backup storage.
- Determining resource allocation for each PostgreSQL cluster.
- Bringing your own trusted certificate authority (CA) for use with the operator application programming interface (API) server.
- Overriding your PostgreSQL configuration.

High availability using the Crunchy PostgreSQL Operator

In the database context, high availability implies the ability to access a database without interruption. Certain events could cause a database to become unavailable during normal operations, including:

A database storage (disk) failure or any other hardware failure.

A network event that renders the database unreachable.

Host operating system instability or crashes.

A corrupted key database file.

Total loss of a data center due to an outage or catastrophe.

Planned downtime events can happen during normal operations, such as performing a minor upgrade, patching an operating system for security, upgrading hardware, or doing other maintenance.

High-availability systems for databases are architected to recover from these and many other situations. An archiving system, such as the one used for disaster recovery, should be in place as well. An archiving system should be able to do a point-in-time-recovery (PITR) to ensure that a database can be reset to a specific point in the timeline before a disaster occurred.

The Crunchy PostgreSQL Operator addresses these needs by integrating trusted and tested open source components that provide a high-availability environment complete with disaster recovery features. These components build on core Kubernetes and PostgreSQL capabilities to perform the following functions:

- Detect downtime events using distributed consensus algorithms and failover to the most appropriate available instance
- Provide connection pooling, holding, and cutover to minimize any disruption to users and applications
- Provide continuous archiving of data to facilitate point-in-time recovery
- Provide a variety of mechanisms to create backups that are stored efficiently and can be restored quickly to achieve recovery point objectives (RPOs)

The Crunchy PostgreSQL Operator provides safe and automated failover to support enterprise high-availability requirements, backed by a distributed consensus-based high-availability solution. As a result, users can easily add or remove nodes, providing automatic failover and resiliency. Figure 2 provides a high-level perspective showing how Kubernetes interacts with the Crunchy PostgreSQL Operator to provide database replication and disaster recovery.

Each PostgreSQL database replica node has its container backed by its copy of the data in a persistent volume claim (PVC). Replica nodes in PostgreSQL can function as read-only nodes as well as failover targets for high availability. Any hardware or network failure event affecting the PostgreSQL primary node will cause failover to one of the available replicas, and that replica is then promoted to the primary node.

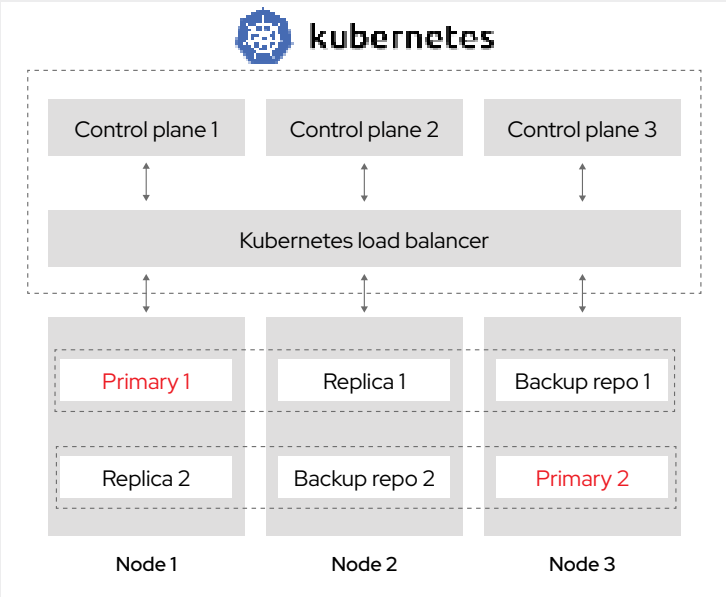

Figure 2. The Crunchy PostgreSQL Operator allows for replication and backup

Crunchy Data PostgreSQL and Red Hat OpenShift Container Storage

Red Hat OpenShift Container Storage services provide PostgreSQL databases with protection against data loss.

There are certain scenarios where read-only replicas are not necessary. In those cases, OpenShift Container Storage can be an effective solution for providing storage replication and high availability for PostgreSQL. OpenShift Container Storage features built-in replication for block, file, and object data. This ability allows PostgreSQL to be deployed as a standalone node with OpenShift Container Storage providing storage replication (default 3x). In the event of a node failure or any other failure, the PostgreSQL pod will be rescheduled quickly to another node. The pod will bond to another OpenShift Container Storage node with the persistent data intact.

The sections that follow describe how the Crunchy PostgreSQL Operator works together with the OpenShift Container Storage Operator.

The OpenShift Container Storage Operator

The OpenShift Container Storage Operator is a single, meta-operator providing one interface for installation and management for three components, including:

- Ceph, as the storage building block providing all storage needs.
- Rook, an operator to manage storage (Ceph) in a Kubernetes cluster.
- NooBaa, an operator for a true multicloud object gateway and management.

Direct-attached storage

With Ceph as the storage technology for OpenShift Container Storage, performance directly correlates to the type of storage provided to the Ceph components. Direct-attached storage, now available from multiple public cloud vendors, offers compelling performance. AWS provides Storage Optimized Instances, offering storage devices directly connected to the instance. Importantly, these devices are not shared among other instances. With support for direct-attached storage in Openshift Container Storage 4.3, the Ceph object storage daemon (OSD) pods consume storage locally attached to the node, instance, or virtual machine (VM) where the OSD pods are running.

Cluster layout

For performance testing, Red Hat and Crunchy Data engineers chose to have three worker nodes dedicated to the Crunchy Data pods (the PostgreSQL databases) and three worker nodes dedicated to providing storage to these databases via OpenShift Container Storage (the storage cluster). Nodes were configured as follows:

- **Database nodes** used AWS m5.4xlarge instances, each with 64GB of RAM and 16 vCPUs.
- **Storage nodes** used AWS i3en.2xlarge instances, each with 64GB of RAM, eight vCPUs, and two direct-attached 2.3TB NVMe devices.

Tests ran on two separate clusters. One cluster was created within a single AWS Availability Zone (us-west-2b). The second cluster was spread across three AWS Availability Zones (us-west-2a/b/c). Figure 3 shows the single availability zone layout. Figure 4 illustrates the layout across multiple availability zones.

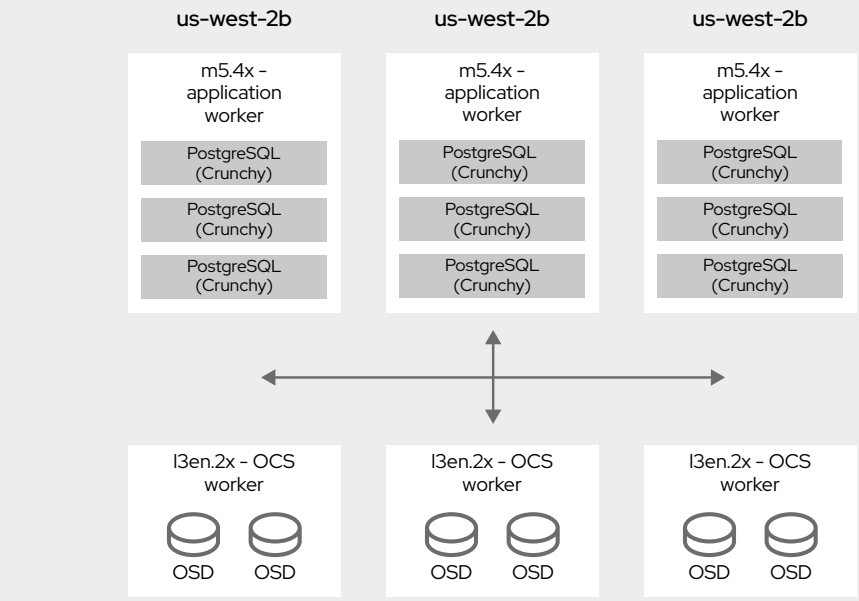

Figure 3. Single availability zone layout

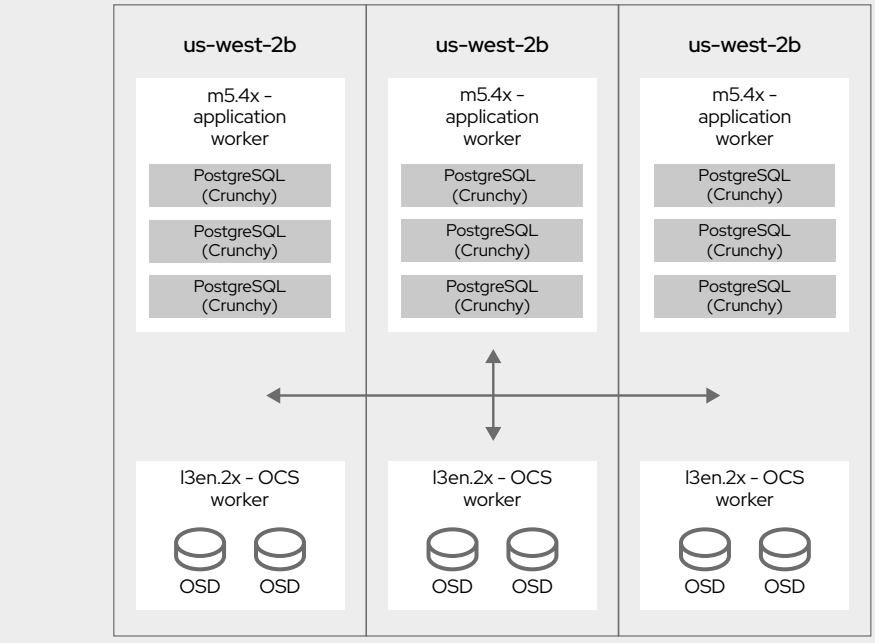

Figure 4. Layout for multiple availability zones

As shown, each Openshift Container Storage worker node contains two Ceph OSD pods. Each of these pods uses a single NVMe device, whether it is in a single or multiple availability zone environment. For the nodes running Crunchy Data PostgreSQL, we've used a pod resource of four vCPUs and 16GB of memory for requests and limits. As such, the configuration uses 12 vCPUs and 48GB of RAM for the PostgreSQL pods. This approach retains extra resources in each node to be used by OpenShift or in the case we need to migrate pods.

Test workload and process

In Red Hat testing, Sysbench was used to benchmark the performance of the cluster. Engineers chose Sysbench because it closely emulates an online web application in processing input/output (I/O) operations. A small bash script spread the Crunchy instances (PostgreSQL pods) equally among the three application worker nodes. Another bash script started statistics collection and Sysbench pods via Kubernetes jobs.

The initial sysbench job loaded data into all the databases. With each database containing 400 tables of 1,000,000 rows, the resulting database saved to storage was 100GB in size. Once the data was loaded into all the databases, a five-minute Sysbench warmup ran in parallel on all nine Crunchy instances simultaneously. The warm-up job consisted of 70% read operations, and 30% write operations.

Once the warm-up phase was complete, ten consecutive runs were started in parallel. Each job consisted of 70% reads and 30% writes. Each job ran for 10 minutes, followed by one minute of rest. In total, engineers collected 100 minutes of Sysbench workload on each database and in parallel across the nine Crunchy nodes.

Performance testing

Tests were run simultaneously on two different clusters. One cluster was configured entirely within a single availability zone, while the other cluster was spread across three availability zones in the AWS region. The tests were always run in parallel on all nine databases (Crunchy PostgreSQL pods) on the three nodes. Figures 5 and 6 illustrate performance for a single zone configuration as measured from PostgreSQL.

Figure 5 shows average transactions per second (TPS) for the nine databases, the total TPS for the whole cluster, as well as average latency and 95th percentile latency per database.

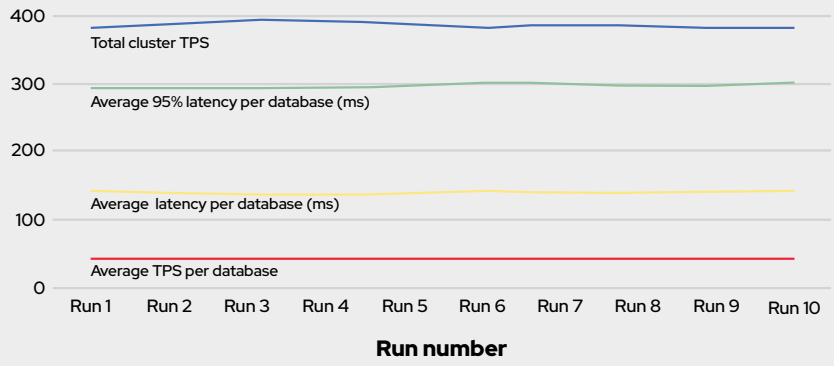

Figure 5. Transactions per second and latency for a single availability zone configuration

Figure 6 shows read and write I/O operations per second (IOPS) across the Ceph block devices (Ceph RADOS Block devices or RBDs).

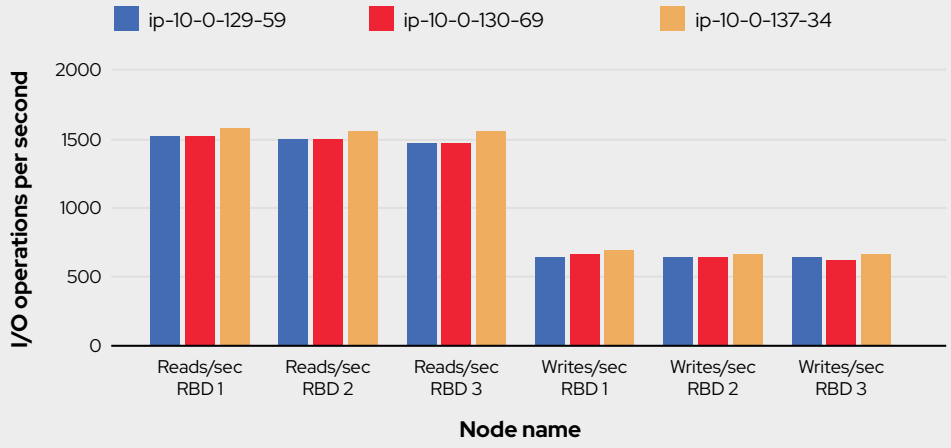

Figure 6. Read and write performance for a single availability zone configuration

Figure 7 shows TPS and latency data for a multizone configuration, with performance results that are highly similar to those obtained in a single availability zone (Figure 5).

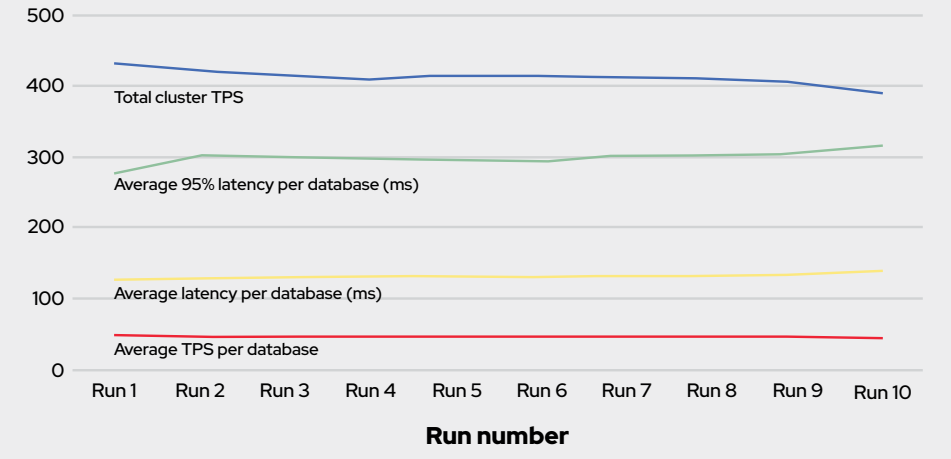

Figure 7. TPS and latency for a multiple availability zone configuration

Figure 8 shows OSD performance per node for a multiple availability zone configuration. Again, these results closely matched those for a single availability zone (Figure 6).

Support business through consistent PostgreSQL workload performance during infrastructure failures.

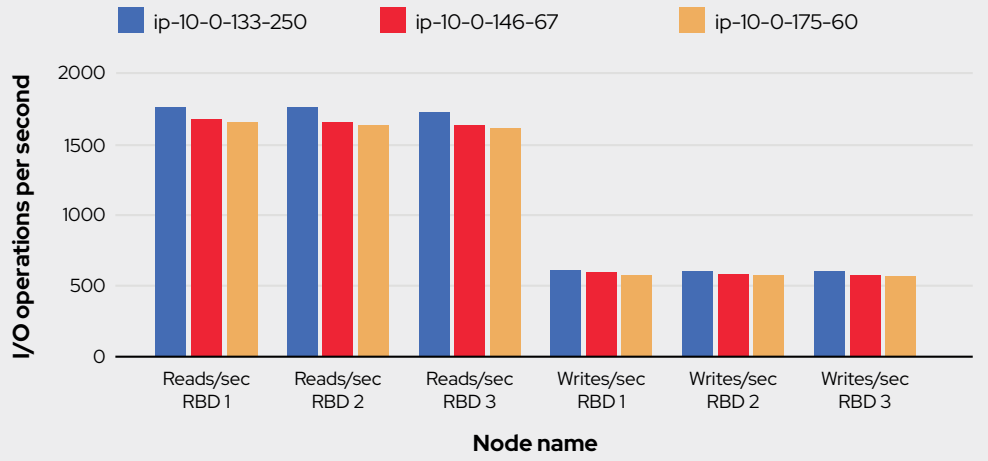

Figure 8. OSD per node performance for a multiple availability zone configuration

Performance summary

In summary, engineers made the following observations based on the testing:

1. The OSD per node shows a stable and consistent spread of I/Os between the databases. This behavior demonstrates that Ceph handles the RBD volumes equally among all the databases.
2. Testing produced an average of roughly 50 TPS per database and 450 TPS for the entire cluster (all nine databases). These results are consistent with expectations based on the type of test (Sysbench 70r/30w), database size, the CPU and RAM resources per Crunchy pod (PostgreSQL database), and the number of Crunchy pods per node.
3. Importantly, OpenShift Container Storage kept the same performance capabilities when moving from a single availability zone to running across multiple availability zones. This finding is remarkable because the devices Ceph used were spread across three instances, with each instance located in a different availability zone. Latency is typically lowest within a single availability zone (e.g., servers located in the same rack or the same row in the data center). Latency is generally higher across multiple availability zones (e.g., servers in different rooms in the datacenter or in different datacenters all together). OpenShift Container Storage made these differences essentially moot.
4. Testing configured resources with the headroom for flexibility. From the Red Hat OpenShift perspective, resources were retained in each instance to provide for failover. From the OpenShift Container Storage perspective, the devices that the Ceph OSDs used averaged 90% utilization, leaving headroom for accommodating device failures.

Resilience and business continuity

Resilience is one of the greatest challenges for any enterprise application, and this is no different for a PostgreSQL database. Performance during failover is another critical aspect because services must continue to operate at sufficient performance, even while underlying infrastructure services are down or recovering from failure. The Crunchy PostgreSQL Operator and the OpenShift Container Storage operator work together to provide resilience that can help with business continuity. This section provides results from testing and validating three common failure scenarios:

1. Simulating human operator error (deleting a Ceph OSD pod)
2. Simulating maintenance operations (rebooting an OpenShift Container Storage instance)
3. Simulating a node failure (shutting down an OpenShift Container Storage instance)

All the scenarios were evaluated with the same infrastructure, cluster, and databases that were used for the performance testing described above. Engineers also ran the same Sysbench workload continuously while triggering these failover scenarios.

Simulating human operator error

Human error is a common source of downtime, and an important aspect to simulate and understand. OSDs are the building blocks of the Ceph data plane, and the Ceph cluster consumes and aggregates all the OSDs into a logical storage layer for the application. In Openshift Container Storage, an OSD pod corresponds to a storage device that the Ceph cluster consumes. Accidentally deleting an OSD pod would result in losing a storage device momentarily. The AWS i3en.2xlarge instances used in testing each have two direct-attached NVMe devices. Two OSDs are run on each instance for six OSDs in total because three nodes were used for Openshift Container Storage.

The Rook operator monitors all the Ceph components, and the configuration used Kubernetes deployments for the OSD pods. As such, deleting an OSD pod causes Kubernetes to immediately bring up another OSD to support the same NVMe device the deleted OSD pod had used before. In testing, the time it took for a new OSD pod to start averaged 2–4 seconds. In fact, recovery occurred so quickly in our tests that the Ceph cluster never required any kind of rebuild process, and I/O operations were barely impacted.

As shown in Figure 9, the same Sysbench workload was run 10 times within the single availability zone cluster. During each run, engineers failed one of the Ceph OSDs in the cluster by deleting one of the OSD pods. A different OSD pod was deleted in a random manner on each run at different points of the run timeline. The impact on performance was minimal, as demonstrated by the nearly identical performance with and without the pod deletion.

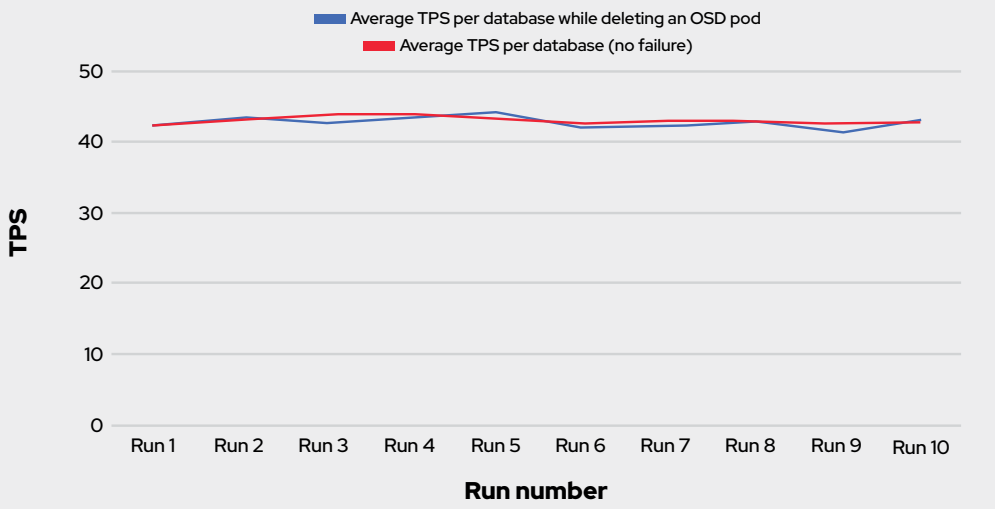

Figure 9. TPS with and without OSD pod deletion across 10 runs

Simulating maintenance operations

Planned downtime for maintenance operations like security patching can also impact operations. In this scenario we concentrate on rebooting a server, or in our case, an AWS instance. An instance reboot may be caused by scheduled maintenance or human error. The OpenShift Container Storage cluster consisted of three AWS i3en.2xlarge instances with two OSDs each. As such, rebooting a node resulted in taking two Ceph OSDs down for 1-2 minutes. Kubernetes recognizes when the AWS instance reboots, notices when it is back up, and restarts all pods on it.

This failure scenario gave engineers the opportunity to study impacts to the workload (in terms of TPS), and also at the recovery time for the Ceph cluster after losing 1/3 of its OSDs. The impact of the recovery on the workload was also of interest. The scenario was run 10 times, as with the previous tests. The Sysbench workload had an average I/O pause[1] of approximately 10 seconds on all nine databases during the 10 runs.

To test the impact on recovery time, engineers needed to run the Sysbench workload for a longer period of time. While previous runs lasted 10 minutes, testing for this scenario increased the workload time to 65 minutes. Results from the testing are shown in Figure 10.

1 *An I/O pause denotes a period of time where the Sysbench workload recorded zero transactions.*

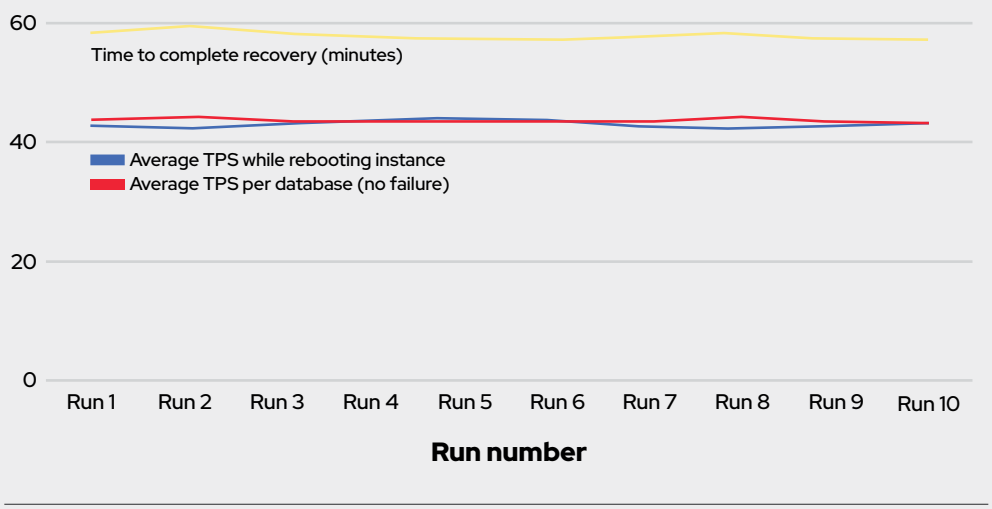

Figure 10. Rebooting an instance with two OSD pods

The red line represents the TPS performance per database while running without failures. The blue line represents the TPS performance per database while running during a reboot of an AWS instance (losing two Ceph OSDs). Importantly, these results closely mirror those obtained when a single OSD was deleted. For all of the test runs, the time to complete the recovery was slightly less than 60 minutes. In this testing, there was negligible impact on the performance of the Sysbench workload during a Ceph cluster recovery, even though a third of the cluster devices were impacted.[2]

Simulating maintenance operations with more OSDs

Importantly, the OpenShift Container Storage cluster under test represented a bare minimum from a Ceph cluster perspective. With only six devices for the OpenShift Container Storage cluster (two in each AWS Elastic Compute Cloud (EC2) instance), this test temporarily removed 1/3 of the devices. Larger OpenShift Container Storage/Ceph clusters—with more nodes, more devices per node, or both—would realize a smaller impact. This is true not only for the impact on the transactions, but also for the recovery time.

Because Ceph can handle significantly more OSDs than were deployed in the test cluster, engineers wanted to explore the impact on performance with more OSDs. An additional six AWS EC2 i3en.2xlarge instances were added to the OpenShift Container Storage cluster. The same three instances were retained for the PostgreSQL databases (via the Crunchy PostgreSQL Operator). In fact, the database remained up and running while the additional storage instances were added. Figure 11-13 illustrates the impact of adding an additional six OSD nodes on TPS, I/O pause, and the full recovery of the cluster.

2 *To capture the recovery time, engineers ran "ceph pg dump pgs_brief|awk '{print $2}'|sort|uniq -c" and then counted the time from when all Ceph Placement Groups (PGs) changed from an "active+clean" state until they returned to an "active+clean" state again. This command can be used via the upstream utility in the Ceph toolbox pod.*

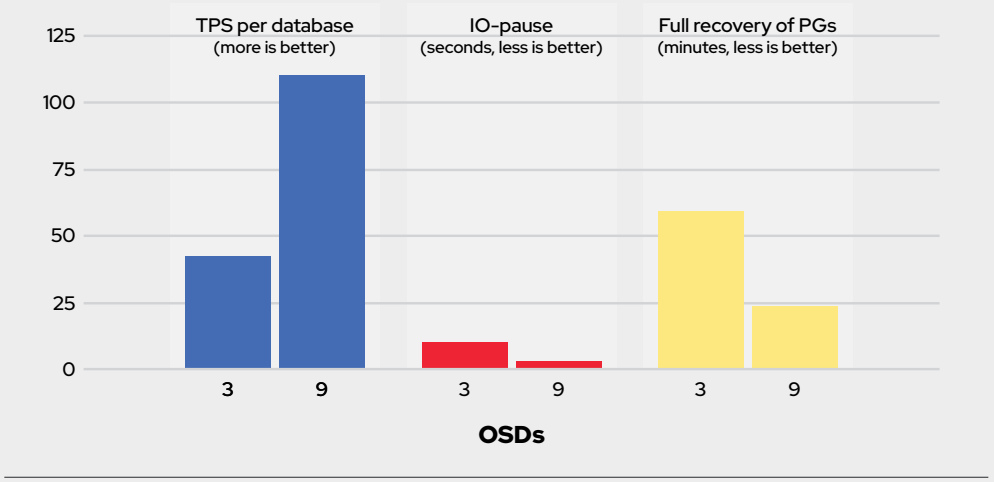

Figure 11. Adding more OSDs to a cluster dramatically improves performance, reduces I/O pause, and achieves full recovery in less time.

As shown, the results showed that average TPS per database went from 42.6 TPS to 109.18 TPS, almost tripling the performance and moving the bottleneck to the compute resources on the instances running the database pods. More importantly, the I/O pause went from an average of 10 seconds with only three OpenShift Container Storage instances to an average of 2 seconds with the six additional instances. The full recovery time of the cluster went from roughly 59 minutes with three OpenShift Container Storage nodes to 23 minutes with nine nodes.

These results demonstrate the power of decoupled software-defined storage. Storage resources can be scaled independently from computational (database) resources, potentially realizing dramatic performance gains.

Simulating a node failure

Node failure was the final resilience scenario tested. This scenario is particularly important because the AWS EC2 storage instances have storage devices that are directly attached to these virtual instances for much better performance. This increased performance comes with a caveat—if the AWS instance is shut down and then started again, it will get new storage devices attached to it. Note that this is distinct from rebooting an AWS instance with direct-attached storage. If an AWS storage instance is rebooted (either via the AWS console or the operating system) it will retain the same storage devices (as discussed in the section on simulating maintenance operations).

With OpenShift Container Storage and its inherent Ceph data replication, other copies of the data spread are distributed on other OSDs. In the event of an instance shutdown, OpenShift Container Storage can automatically update the new direct-attached devices to be part of the cluster.

For the instance shutdown test, engineers reverted to a three-node OpenShift Container Storage cluster. With the same Sysbench workload running as before, they performed an instance shutdown from the AWS console (though it can also be done from the operating system). The process is summarized here, and full details are provided in the appendix.

- Under load, engineers shut down one of the OpenShift Container Storage nodes.
- After one minute, they used the console to restart the instance. When the instance came back online, it had two new NVMe devices.
- Engineers then logged into the instance and updated symbolic links for the two new NVMe devices.
- Next, the OpenShift Local Storage Operator (LSO) LocalVolume custom resource (CR) was updated with the new NVMe device names.

Once the CR was saved, new symbolic links were created by the LSO, and the OpenShift Container Storage operator started to take action. In our testing, the new pods were created in seconds. At this point, Ceph began using the two new OSDs and started to copy pages of Placement Groups to the new OSDs, making the new NVMe devices part of the existing OpenShift Container Storage cluster. Resiliency was restored to 100% (three copies). This rebuild process was similar to what happened in our second test case ("rebooting an instance") and it had no impact on the performance of the running workloads.

Conclusion

Running Crunchy Data PostgreSQL databases on Red Hat OpenShift Container Storage provides a simple, general solution to business continuity during infrastructure failure and data protection against loss. The Crunchy PostgreSQL Operator provides a convenient mechanism to configure primary and replica database nodes for database failover and workload scalability through multiple read-only database instances.

In instances where additional read-only database replicas are not required, resilience and data redundancy can be achieved by using Openshift Container storage along with the Crunchy PostgreSQL Operator. Internal OpenShift Container Storage data replication provides highly available PostgreSQL. That replication can be easily extended across multiple AWS availability zones with little or no performance penalty to the application in terms of both TPS and latency, even during failover events. As a common storage services layer for Red Hat OpenShift, OpenShift Container Storage can also provide this same data protection to other applications running in the same Red Hat OpenShift cluster.

Appendix: Instance shutdown and recovery procedure

This section illustrates the instance shutdown testing process used in Red Hat testing in detail.

To execute the test, the workload was started and engineers waited several minutes before using the AWS console to shut down one of our OpenShift Container Storage nodes. They then waited another minute and used the AWS console to restart the instance. Once the instance restarted and was running in "Ready" state, two of the OSD pods were shown as not being in the "Running" state.

```
[sagy@ip-172-31-37-35 ~]$ oc get pods -n openshift-storage|grep osd
rook-ceph-osd-0-7fb596d79f-94z7c                          1/1     Running
rook-ceph-osd-1-6f99fc8b6d-pfzbz                          1/1     Running
rook-ceph-osd-2-564799d754-9nndd                          1/1     Running
rook-ceph-osd-3-8545f4968b-28pwd                          1/1     Running
rook-ceph-osd-4-7b85fd57bc-lgpzd                          0/1
PodInitializing
rook-ceph-osd-5-79b7c66699-nl84g                          0/1
PodInitializing
rook-ceph-osd-prepare-ocs-deviceset-0-0-mwdhg-ss8g6       0/1     Completed
rook-ceph-osd-prepare-ocs-deviceset-0-1-mpbqj-z9tfn       0/1     Completed
rook-ceph-osd-prepare-ocs-deviceset-1-0-rc5dw-rjkxn       0/1     Completed
rook-ceph-osd-prepare-ocs-deviceset-1-1-g4lzn-7kzhv       0/1     Completed
rook-ceph-osd-prepare-ocs-deviceset-2-0-w2z6h-qxxkg       0/1     Completed
rook-ceph-osd-prepare-ocs-deviceset-2-1-zxqtr-vlxpc       0/1     Completed
```

When the AWS i3en.2xlarge instance came back online, it had two new NVMe devices. First, engineers needed to change the OpenShift Local Storage Operator (LSO) Persistent Volumes (PVs) to point to the new NVMe devices. To do this, engineers logged into the instance that was shutdown.[3] Once gaining access to the node, the engineers checked the symbolic link that the LSO created previously, before the node shutdown. The links are located at /mnt/local-storage/localblock and are named nvme1n1 and nvme2n1.

```
[core@ip-10-0-142-99 ~]$ ls -l $(readlink /mnt/local-storage/localblock/
nvme1n1)
ls: cannot access '/dev/disk/by-id/
nvme-Amazon_EC2_NVMe_Instance_Storage_AWSBBA5226476FB42CBA'
[core@ip-10-0-142-99 ~]$ ls -l $(readlink /mnt/local-storage/localblock/
nvme1n2)
ls: cannot access '/dev/disk/b -id/
nvme-Amazon_EC2_NVMe_Instance_Storage_AWSB3B810595C5501A68
```

3 *Note: Additional information on accessing and troubleshooting OpenShift Container Platform can be found at: https://docs.openshift.com/container-platform/4.5/support/troubleshooting/troubleshooting-installations.html.*

After the shutdown, the targets of these symbolic links were missing. These links had to be deleted and replaced with links to the new NVMe devices that AWS provided to the new i3en.2xlarge instance once it was up and running again.

```
[core@ip-10-0-142-99 ~]$ sudo rm /mnt/local-storage/localblock/nvme1n1

[core@ip-10-0-142-99 ~]$ sudo rm /mnt/local-storage/localblock/nvme2n1

[core@ip-10-0-142-99 ~]$

[core@ip-10-0-142-99 ~]$ ls /dev/disk/by-id/
nvme-Amazon_EC2_NVMe_Instance_Storage*

/dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
AWSBC37B6978678E4DD2

/dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
AWSBDB094A84CECD5A70
```

With the new NVMe devices, the LSO LocalVolume custom resource can be updated using:
oc edit LocalVolume local-block -n local-storage

As shown below in the devicePaths array, the paths still depict the "old" NVMe devices that the AWS instance had prior to the shutdown (the first two devices listed, in this case).

```
- devicePaths:

   - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
     AWSB3B810595C5501A68

   - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
     AWSBBA5226476FB42CBA

   - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
     AWS22288A6726F4ACBC9

   - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
     AWS2C62109858ACCA8B7

  - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
    AWS27C9641EB4D9DF877

   - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
     AWS2C5DF1B52A264A99A

   storageClassName: localblock

   volumeMode: Block
```

The old devices can now be replaced with the two new devices that are associated with the new i3en.2xlarge instance.

```
- devicePaths:
    - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
      AWSBC37B6978678E4DD2
    - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
      AWSBDB094A8E4CCD5A70
    - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
      AWS22288A6726F4ACBC9
    - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
      AWS2C62109858ACCA8B7
    - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
      AWS27C9641EB4D9DF877
    - /dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
      AWS2C5DF1B52A264A99A
    storageClassName: localblock
    volumeMode: Block
```

Once the CR is saved, new symbolic links will be created by the LSO.

```
[core@ip-10-0-142-99 ~]$ ls $(readlink /mnt/local-storage/localblock/
nvme1n1)
/dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
AWSBC37B6978678E4DD2
[core@ip-10-0-142-99 ~]$ ls $(readlink /mnt/local-storage/localblock/
nvme2n1)
/dev/disk/by-id/nvme-Amazon_EC2_NVMe_Instance_Storage_
AWSBDB094A84CECD5A70
```

Once the symbolic links were recreated, the Ceph OSDs that were using the old devices needed to be deleted. OpenShift Container Storage and the Rook operator then recreated new OSDs and new OSD pods. The OpenShift Container Storage operator has a specific, built-in job template for this use case.[4] Viewing the previous output of OSD pods, OSD numbers 4 and 5 needed to be deleted.

4 *Refer to the Red Hat OpenShift Container Storage documentation section 9.3.2.4 for how to remove an OSD via an OpenShift Container Storage job.*

```
[sagy@ip-172-31-37-35 ~]$ osd_id_to_remove=4
[sagy@ip-172-31-37-35 ~]$ ocprocess -n openshift-storage ocs-osd-removal
-p \
> FAILED_OSD_ID=${osd_id_to_remove} | oc create -n openshift-storage -f -
job.batch/ocs-osd-removal-4 created
[sagy@ip-172-31-37-35 ~]$ osd_id_to_remove=5
[sagy@ip-172-31-37-35 ~]$ ocprocess -n openshift-storage ocs-osd-removal
-p \
> FAILED_OSD_ID=${osd_id_to_remove} | oc create -n openshift-storage -f -
job.batch/ocs-osd-removal-5 created
[sagy@ip-172-31-37-35 ~]$ oc get jobs -n openshift-storage
NAME                          COMPLETIONS         DURATION       AGE
/ocs-osd-removal-4            1/1                  5s            32s
/ocs-osd-removal-5            1/1                  4s            7s
```

Once the two jobs were completed, the OSDs disappeared, but the deployments, and thus the pods, still existed. These had to be removed.

```
[sagy@ip-172-31-37-35 ~]$ oc get deployments -n openshift-storage |grep
osd
rook-ceph-osd-0                                    1/1     1
rook-ceph-osd-1                                    1/1     1
rook-ceph-osd-2                                    1/1     1
rook-ceph-osd-3                                    1/1     1
rook-ceph-osd-4                                    0/1     1
rook-ceph-osd-5                                    0/1     1
```

```
[sagy@ip-172-31-37-35 ~]$ oc delete deployment rook-ceph-osd-4 -n open-
shift-storage deployment.apps “rook-ceph-osd-4” deleted
[sagy@ip-172-31-37-35 ~]$ oc delete deployment rook-ceph-osd-5 -n open-
shift-storage deployment.apps “rook-ceph-osd-5” deleted
```

Once the deployments were deleted, triggering a new reconcile of the Ceph cluster CR caused the OpenShift Container Storage and Rook operators to take note and compare to the original Openshift Container Storage setup. They noticed that two OSDs were missing and started deployment. The easiest way to perform this operation is to update the settings in the CR via:

oc edit cephcluster ocs-storagecluster-cephcluster -n openshift-storage

Then look for "storageClassDeviceSets" and update the first count from 2 to 0 (it really doesn't matter which storageClassDeviceSets you update – OpenShift Container Storage will always compare to what was initially installed and make sure we are at the same level, so if you edit the CR again, you will see the count is back at 2)

In this example, two new deployments for OSD 4 and 5 were created and the corresponding osd-prepare and OSD pods were up and running. It is important to mention that the number of OSDs in the Ceph cluster can impact the timing of when you will see the new OSD pods being created.

facebook.com/redhatinc
@RedHat
linkedin.com/company/red-hat

NORTH AMERICA
1 888 REDHAT1

EUROPE, MIDDLE EAST, AND AFRICA
00800 7334 2835
europe@redhat.com

ASIA PACIFIC
+65 6490 4200
apac@redhat.com

LATIN AMERICA
+54 11 4329 7300
info-latam@redhat.com

redhat.com
#F25145_0920