SQL Server 2019 Containers on Linux

Software Development Use Cases Using Dell EMC Infrastructure

July 2020

H17857.1

White Paper

Abstract

This white paper demonstrates the advantages of using Microsoft SQL Server 2019 containers for an application development and testing environment that is hosted on a Dell EMC platform.

Dell Technologies Solutions

Copyright

Contents

Executive summary

Business challenge

Implementing reliable transaction processing for large-scale systems is beyond the capability of many software developers. However, commercial relational database management system (RDBMS) products enable developers to create many applications that they otherwise could not. Although using an RDBMS solves many software development problems, one longstanding issue persists—how to ensure code and data consistency between the RDBMS and the application.

This challenge of managing the state of code and data is a particularly thorny one during the software development and testing (dev/test) life cycle. As code is added and changed in the application, testers must have a known state for the code and data in the database. For example, if a test is designed to add 10 customer accounts to an existing database and then test for the total number of customers, the team must ensure that they are starting with same set of base customers every time. For larger applications with hundreds or thousands of tests, this activity becomes a challenge even for experienced teams.

Container technology enables development teams to quickly provision isolated applications without the traditional complexities. For many companies, to boost productivity and time to value, the use of containers starts with the departments that are focused on software development. The journey typically starts with installing, implementing, and using containers for applications that are based on the microservice architecture. In the past, integration between containerized applications and database services like Microsoft SQL Server were clumsy at best. Often, they would introduce delays in the agile development process.

Solution overview

This solution shows how the use of SQL Server containers, Kubernetes, and the Dell EMC XtremIO X2 Container Storage Interface (CSI) plug-in transforms the development process. Using orchestration and automation, developers can self-provision a SQL Server database, increasing productivity and saving substantial time.

We are choosing to focus on the software dev/test use case because many analysts agree that this market represents the most immediate opportunity to solve significant business challenges using SQL Server on containers. The current method for developing SQL Server powered applications consists of a hodge-podge of platforms and tools. The process is overly complex and prone to creating schedule delays and cost overruns. Any path forward that has advantages for IT professionals and provides a more heterogeneous and familiar environment for software developers will likely gain significant adoption with minimal friction or risk.

Document purpose

In this paper, we expand on information that is available from Microsoft and the SQL Server ecosystem, providing two use cases that highlight the test/dev benefits that SQL Server containers enable. In addition, we explore the intersection of SQL Server 2019 Docker containers, the Kubernetes implementation of the CSI specification, and products and services from Dell Technologies. The use cases that we present are designed to show how developers and others can easily use SQL Server containers with the XtremIO

X2 storage array. Using the XtremIO X2 CSI plug-in enables comprehensive automation and orchestration from server through storage.

Audience

This white paper is for IT professionals who are interested in learning about the benefits of implementing SQL Server containers in a dev/test environment.

Terminology

The following table defines some of the terms that are used in this white paper:

Table 1. Terminology

Term	Description
Container	An isolated object that includes an application and its dependencies. Programs running on Docker are packaged as Linux containers. Because containers are a widely accepted standard, many prebuilt container images are available for deployment on Docker.
Cluster	A Kubernetes cluster is a set of machines that are known as nodes. One node controls the cluster and is designated as the master node; the remaining nodes are worker nodes. The Kubernetes master is responsible for distributing work among the workers and for monitoring the health of the cluster.
Node	A node runs containerized applications. It can be either a physical machine or a virtual machine. A Kubernetes cluster can contain a mixture of physical machine and virtual machine nodes.
Pod	A pod is the minimum deployment unit of Kubernetes. It is a logical group of one or more containers and associated resources that are needed to run an application. Each pod runs on a node, which can run one or more pods. The Kubernetes master automatically assigns pods to nodes in the cluster.

We value your feedback

Dell Technologies and the authors of this document welcome your feedback on the solution and the solution documentation. Contact the Dell Technologies Solutions team by email or provide your comments by completing our documentation survey.

Author: Sam Lucido

Contributors: Phil Hummel, Anil Papisetty, Sanjeev Ranjan, Mahesh Reddy, Abhishek Sharma, Karen Johnson

Use case overview

Our use cases demonstrate the advantages of using Microsoft SQL Server 2019 containers for an application dev/test environment that is hosted on a Dell EMC infrastructure platform. The test environment for both use cases consisted of three Dell EMC PowerEdge R740 servers and an XtremIO X2 all-flash storage array that were hosted in our labs. For an architecture diagram and details about the solution configuration, see Appendix A: Solution architecture and component specifications.

The use cases demonstrate how Docker, Kubernetes, and the XtremIO X2 CSI plug-in accelerate the SQL Server development life cycle. With this solution, developers can easily provision SQL Server container databases without the complexities that are associated with installing the database and provisioning storage.

Use Case 1 overview

In the first use case, we start the way many companies begin to work with containers—by installing Docker and establishing a functioning development environment. Our goal is to quickly provision a SQL Server container and then attach a copy of a sample database—the popular AdventureWorks database from Microsoft—using a Dell EMC XtremIO X2 storage array. With the SQL AdventureWorks container running, we show how to access the database using a web browser to simulate a typical enterprise web application. Then we remove the container and clean up the environment to free resources for the next sprint.

Use Case 2 overview

The second use case continues the containerized application journey by using the XtremIO X2 CSI plug-in for Kubernetes to achieve a greater level of automation and ease of management for dev/test environments. Here we move beyond manually provisioning storage to automated provisioning. Using Kubernetes, our developer controls the provisioning of the SQL container from a local private registry and the database storage from the XtremIO X2 array. After working on the AdventureWorks database application, the developer protects the updated state of the database code and data by using Kubernetes to take an XtremIO Virtual Copies snapshot of the database. After a round of destructive testing, the developer then restores the database to the preserved state by using Kubernetes and XtremIO Virtual Copies. A technical writer provisions the modified database to document the code changes, and the developer removes the containers and cleans up the environment.

Use case comparison summary

The following table provides a high-level comparison of the two use cases:

Table 2. Use-case comparison

Action	Use Case 1: Docker only	Use Case 2: Kubernetes and XtremIO X2 CSI plug-in
Provisioning a SQL Server container	Manual, using script	Self-service (full automation)
Provisioning an AdventureWorks database	Storage and operating system administrator tasks	
Removing the container and persistent storage	Manual, using script	

Supporting software technology

This section summarizes the important technology components of this solution.

Container-based virtualization

Two primary methods of enabling software applications to run on virtual hardware are through the use of virtual machines (VMs) and a hypervisor, and through container-based virtualization—also known as operating system virtualization or containerization.

The older and more pervasive virtualization method, which was first developed by Burroughs Corporation in the 1950s, is through the use of VMs and a hypervisor. That method was replicated with the commercialization of IBM mainframes in the early 1960s. The primary virtualization method that is used by platforms such as IBM VM/CMS, VMware ESXi, and Microsoft Hyper-V starts with a hypervisor layer that abstracts the physical components of the computer. The abstraction enables sharing of the components by multiple VMs, each running a guest operating system. A more recent development is container-based virtualization, where a single host operating system supports multiple processes that are running as virtual applications.

The following figure contrasts VM-based virtualization with container-based virtualization. In container-based virtualization, the combination of the guest operating system components and any isolated software applications constitutes a container running on the host server, as indicated by the App 1, App 2, and App 3 boxes.

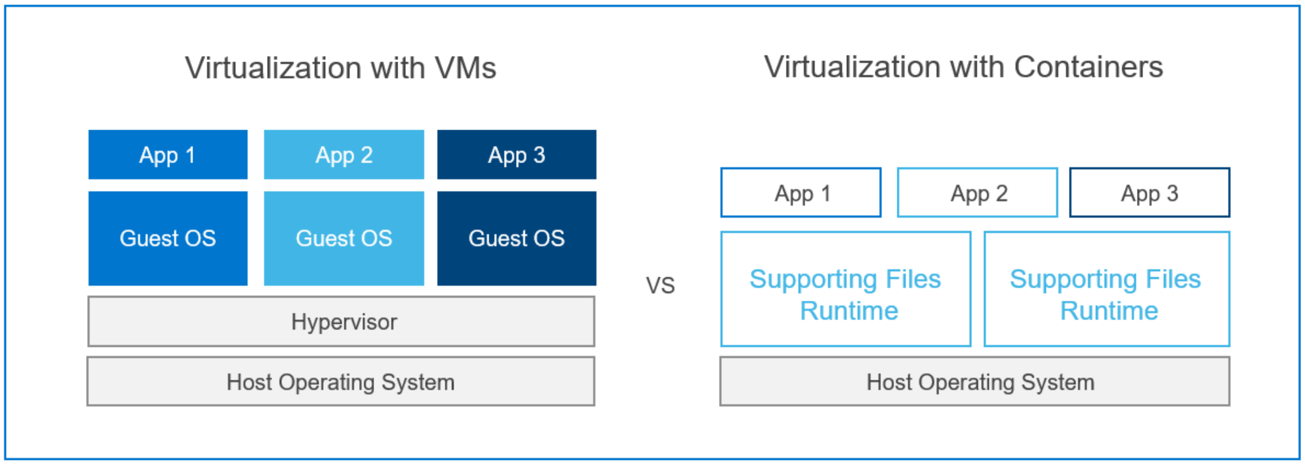


Figure 1. Primary virtualization methods

Both types of virtualization were developed to increase the efficiency of computer hardware investments by supporting multiple users and applications in parallel. Containerization further improves IT operations productivity by simplifying application portability. Application developers most often work outside the server environments that their programs will run in. To minimize conflicts in library versions, dependencies, and configuration settings, developers must re-create the production environment multiple times for development, testing, and preproduction integration. IT professionals have found containers easier to deploy consistently across multiple environments because the core operating system can be configured independently of the application container.

Docker containers

Concepts that led to the development of container-based virtualization began to emerge when the UNIX operating system became publicly available in the early 1970s. Container technology development expanded on many fronts until 2013 when Solomon Hykes released the Docker code base to the open-source community. The Docker ecosystem is made up of the container runtime environment along with tools to define and build application containers and to manage the interactions between the runtime environment and the host operating system.

Two Docker runtime environments—the Community Edition and the Enterprise Edition—are available. The Community Edition is free and comes with best-effort community support. For our use-case testing, we used the Enterprise Edition, which is fitting for most organizations that are using Docker in production or business-critical situations. The Enterprise Edition requires purchasing a license that is based on the number of cores in the environment. Organizations likely will have licensed and nonlicensed Docker runtimes and should implement safeguards to ensure that the correct version is deployed in environments where support is critical.

A Docker registry is supporting technology that is used for storing and delivering Docker images from a central repository. Registries can be public, such as Docker Hub, or private. Docker users install a local registry by downloading from Docker Hub a compressed image that contains all the necessary container components that are specific to the guest operating system and application. Depending on Internet connection speed and availability, a local registry can mitigate many of the challenges that are associated with using a public registry, including high latency during image downloading. Docker Hub does provide the option for users to upload private images. However, a local private registry might offer both better security and less latency for deployment.

Private registries can reside in the cloud or in the local data center. Provisioning speed and provisioning frequency are two factors to consider when determining where to locate a private registry. Private registries that are hosted in the data center where they will be used benefit from the speed and reliability of the LAN, which means images can be provisioned quickly in most cases. For our use cases, we implemented a local private registry to enable fast provisioning without the complexities and cost of hosting in the cloud.

Kubernetes

Modern applications—primarily microservices that are packaged with their dependencies and configurations—are increasingly being built using container technology. Kubernetes, also known as K8s, is an open-source platform for deploying and managing containerized applications at scale. The Kubernetes container orchestration system was open-sourced by Google in 2014.

The following figure shows the Kubernetes architecture:

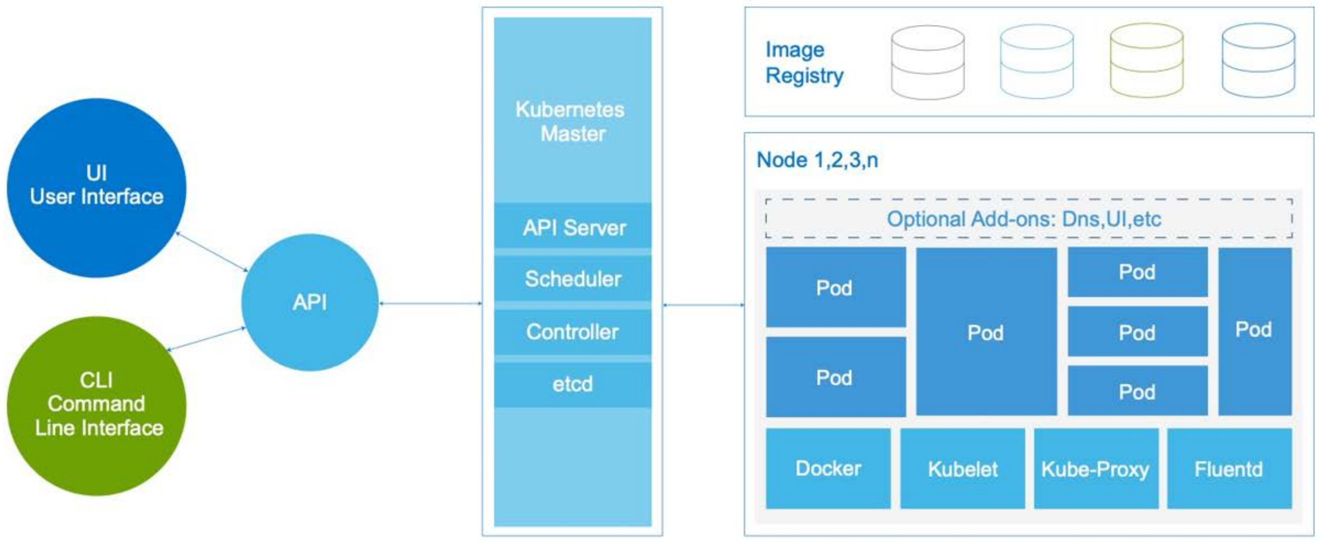


Figure 2. Kubernetes architecture

Kubernetes features for container orchestration at scale include:

- Auto-scaling, replication, and recovery of containers
- Intra-container communication, such as IP sharing
- A single entity—a pod—for creating and managing multiple containers
- A container resource usage and performance analysis agent, cAdvisor
- Network pluggable architecture
- Load balancing
- Health check service

In a simulated dev/test scenario in Use Case 2, we used the Kubernetes container orchestration system to deploy two Docker containers in a pod.

Kubernetes Container Storage Interface specification

The Kubernetes CSI specification was developed as a standard for exposing arbitrary block and file storage systems to containerized workloads through an orchestration layer. Kubernetes previously provided a powerful volume plug-in that was part of the core Kubernetes code and shipped with the core Kubernetes binaries. Before the adoption of CSI, however, adding support for new volume plug-ins to Kubernetes when the code was “in-tree” was challenging. Vendors wanting to add support for their storage system to Kubernetes, or even fix a bug in an existing volume plug-in, were forced to align with the Kubernetes release process. In addition, third-party storage code could cause reliability and security issues in core Kubernetes binaries. The code was often difficult—or sometimes impossible—for Kubernetes maintainers to test and maintain.

The adoption of the CSI specification makes the Kubernetes volume layer truly extensible. Using CSI, third-party storage providers can write and deploy plug-ins to expose new storage systems in Kubernetes without ever having to touch the core Kubernetes code. This capability gives Kubernetes users more storage options and makes the system more secure and reliable. Our Use Case 2 highlights these advantages by using the Dell EMC XtremIO X2 CSI plug-in to show the benefits of Kubernetes storage automation.

Kubernetes storage classes

We do not directly use Kubernetes storage classes in either of the use cases that we describe in this paper; however, the Kubernetes storage classes are closely related to CSI and the XtremIO X2 CSI plug-in. Kubernetes provides administrators an option to describe various levels of storage features and differentiate them by quality-of-service (QoS) levels, backup policies, or other storage-specific services. Kubernetes itself is unopinionated about what these classes represent. In other management systems, this concept is sometimes referred to as storage profiles.

The XtremIO X2 CSI plug-in creates three storage classes in Kubernetes during installation. The XtremIO X2 storage classes, which can be viewed from the Kubernetes dashboard, are predefined. These storage classes enable users to specify the amount of bandwidth to be made available to persistent storage that is created on the array. The following table shows the predefined storage classes:

Table 3. XtremIO X2 CSI predefined storage classes

Storage class	MB/s per GB
High	15
Medium	5
Low	1

The size of the requested storage volume and the storage class define the amount of bandwidth to be specified. For example, bandwidth for a 1,000 gibi (Gi) storage volume configured with the medium storage class is computed as follows:

Storage size (1,000 Gi) x storage class (medium at 5 MB/s per GB) = Total bandwidth (5,000 MB/s)

Note: Gi indicates power-of-two equivalents—1024^3 in this case.

Using the XtremIO X2 predefined storage classes helps to efficiently scale an environment by defining performance limits. For example, a storage class of low for a pool of 100 containers limits containerized applications so that they consume no more than their allocated bandwidth. Such limitations help to maintain more reliable storage performance across the entire environment.

Using QoS-based storage classes helps balance the resources that are consumed by containerized applications and the total amount of storage bandwidth. For scenarios that require a more customized set of storage classes than the one that is created by the XtremIO X2 CSI plug-in, you can configure XtremIO X2 QoS in Kubernetes. In creating a custom QoS policy, you can define maximum bandwidth per gigabyte or, alternatively, maximum IOPS. You could also define a burst percentage, which is the amount of bandwidth or IOPS above the maximum limit that the container can use for temporary performance.

The benefits of using predefined storage classes and customized QoS policies include:

- Guaranteed service for critical applications
- Eliminating “noisy neighbor” problems by placing performance limits on nonproduction containers

SQL Server and Docker containers on Linux

In recent years, Microsoft has been expanding its portfolio of offerings that are either compatible with or ported to the Linux operating system. For example, Microsoft released the first version of its SQL Server RDBMS that was commercially available on Linux in November 2016. More recently, with its SQL Server 2017 release, Microsoft delivered SQL Server on Docker containers. The next generation of SQL Server for Linux containers is in development, as part of SQL Server 2019, with release scheduled for the fall of 2019.

Microsoft is currently developing SQL Server implementations of Linux containers for both Linux and Window hosts as well as Windows containers for Windows. The supported features and road maps for these implementations vary, so carefully verify whether a product will meet your requirements. For this white paper, we worked exclusively with

SQL Server containers for Linux. We recommend that you check with Dell Technologies to ensure that the latest certified CSI plug-ins are used in your Kubernetes environment.

Microsoft first introduced support for containerized Linux images in SQL Server 2017. According to Microsoft, one of the primary use cases for customers who are adopting SQL Server containers is for local dev/test in DevOps pipelines, with deployment handled by Kubernetes. SQL Server in containers offers many advantages for DevOps because of its consistent, isolated, and reliable behavior across environments, ease of use, and ease of starting and stopping. Applications can be built on top of SQL Server containers and run without being affected by the rest of the environment. This isolation makes SQL Server in containers ideal for test deployment scenarios as well as DevOps processes.

Dell EMC servers and storage

PowerEdge servers

Dell EMC PowerEdge servers provide a scalable business architecture, intelligent automation, and integrated security for high-value data-management and analytics workloads. The PowerEdge portfolio of rack, tower, and modular server infrastructure, based on open-standard x86 technology, can help you quickly scale from the data center to the cloud. PowerEdge servers deliver the same user experience and the same integrated management experience across all our product options; thus, you have one set of runbooks to patch, manage, update, refresh, and retire all your assets.

For our use cases, we chose the PowerEdge R740 server. The R740 is a 2U form factor that houses up to two Intel Xeon Scalable processors, each with up to 28 compute cores. It has support for the most popular enterprise-deployed versions of Linux—Canonical Ubuntu, Red Hat Enterprise Linux, and SUSE Linux Enterprise Server. The R740 supports a range of memory configurations to satisfy the most demanding database and analytic workloads. It includes 24 slots for registered ECC DDR4 load-reduced DIMMS (LRDIMMs) with speeds up to 2,933 MT/s and has expandable memory up to 3 TB. On-board storage can be configured with front drive bays holding up to 16 x 2.5 in. SAS/SATA SSDs, for a maximum of 122.88 TB, or up to 8 x 3.5 in. SAS/SATA drives, for a maximum of 112 TB.

For details about the PowerEdge server configuration that we used for our use cases, see Appendix A: Solution architecture and component specifications.

XtremIO X2 storage

The Dell EMC XtremIO X2 all-flash array is an ideal storage platform for running online transaction processing (OLTP), online analytical processing (OLAP), or mixed workloads. It delivers high IOPS, ultrawide bandwidth, and consistent submillisecond latency for databases of all sizes.

Note: For details about designing a SQL Server solution using XtremIO X2 all-flash storage with PowerEdge servers, see *Dell EMC Ready Solutions for Microsoft SQL: Design for Dell EMC XtremIO*. The guide provides recommended design principles, configuration best practices, and validation with both Windows Server 2016 and Red Hat Enterprise Linux 7.6 running instances of SQL Server 2017. In the solution testing, the XtremIO X2 array delivered sub-500-microsecond latencies while supporting 275,000-plus IOPS with 72 flash drives, compared to a rated 220,000 achievable IOPS per the XtremIO X2 specification sheet. The test engineers found no noticeable increase in latency even when the XtremIO X2 array exceeded the total expected IOPS.

We chose the XtremIO X2 storage array for our use cases to highlight its innovative management capability in Kubernetes/Docker environments. The use cases show how DBAs and system administrators can use the XtremIO/Kubernetes CSI 1.0 plug-in to deploy a SQL Server 2019 Linux container with persistent storage hosted by the XtremIO X2 array.

Note: For details about using the CSI plug-in to provision XtremIO storage for Kubernetes clusters, see *Dell EMC XtremIO X2 Container Storage Interface Plugin Integration with Kubernetes*.

In Use Case 2, we show the use of XtremIO Virtual Copies. For provisioning, the developer uses a protected copy, or "gold master," of the AdventureWorks database, which is maintained on the XtremIO X2 array. XtremIO Virtual Copies makes provisioning copies of gold master databases easy. In our use case, we use Kubernetes to create a copy of our AdventureWorks database through the XtremIO Virtual Copies capability. This approach enables easy provisioning of multiple consistent copies of the database. After destructive testing, the developer restores the database from XtremIO Virtual Copies.

Use Case 1: Manual provisioning of a containerized dev/test environment

In Use Case 1, we manually provision a container-based dev/test environment as follows:

1. Install the Docker runtime on a PowerEdge R740 server that is running Red Hat Enterprise Linux.
2. Install and configure a local private Docker registry.
3. Pull the most recent SQL Server 2019 for Linux container image from Docker Hub.
4. Push the SQL Server image to the local private registry.
5. Pull a lightweight data access layer container image from Docker Hub and push the image to the local private registry.
6. Configure a storage volume containing the Microsoft AdventureWorks database files and connect the volume to the SQL Server container by using Linux bind mounts.
7. Verify access to the AdventureWorks database through the data access layer application, using a web browser.
8. Clean up the environment by removing the containers.

Step 1: Install Docker

The first step in creating a developer environment using Docker containers is licensing and installing the Docker runtime.

The Docker administrator installs the Enterprise Edition by running the following installation command:

```
$ yum -y install docker-ee docker-ee-cli containerd.io
```

By default, Docker containers access all host memory. Also by default, containers can access all the host's CPU cycles. For details about how to place memory and CPU restraints on a Docker container, see Appendix B: Container resource configuration.

Step 2: Install the Docker registry

Docker registry placement is a key consideration when building a dev/test environment. The factors that influence Docker registry placement include:

- **Variety**—Number of images in the registry
- **Velocity**—Frequency of application provisioning
- **Security**—Protection of application images

For our use cases, we use a local private registry, which addresses our key variety, velocity, and security requirements.

To create a local registry from Docker Hub, the developer runs the following commands:

```
$ docker pull registry
$ mkdir -p /registry/private
$ docker run -d -p 5000:5000 -name registry -v /var/lib/registry -
-restart
```

Step 3: Pull the SQL Server 2019 image

The Docker administrator downloads the latest Microsoft SQL Server image from Docker Hub to our local private registry. The Docker administrator ensures that all stated requirements for this SQL Server image are met, including Docker Engine 1.8+ on Docker-supported platforms and a minimum of 2 GB of RAM.

Note: Having access to high-quality container images from a trusted source can save many hours of labor that are typically required to create and manage images that are built locally from Docker files. Always check requirements before attempting to deploy a container image.

To download the image, the Docker administrator runs the following Docker `pull` command:

```
$ docker pull mcr.microsoft.com/mssql/server
```

Step 4: Push the SQL Server image to the local private registry

To populate the local private Docker registry with the SQL Server, the Docker administrator runs the following Docker `push` command:

```
$ docker push localhost:5000/sqlserver2019
```

Note: If you customize the SQL Server container image, save both the base image and any customization to the local private registry with appropriate annotations, if required, for your business use case.

Step 5: Pull the REST API and push it to the local private registry

As part of this simulated development environment, the developer uses a lightweight data access layer container that implements a simple ASP.NET Core 2.0 REST API web application prototype. Using SQL Server Management Objects (SMO), the prototype provides a RESTful interface for SQL Server running on-premises, Azure SQL Database, and Azure SQL Data Warehouse. This lightweight data access layer for HTTP clients, which simulates a typical three-tier database application design, enables the developer to create web-based business reports.

The developer obtains the container image from a Github repository by running the following Docker `pull` command:

```
$ docker pull sanagama/mssql-restapi
```

The developer then saves the image to the local private registry by running the following Docker `push` command:

```
$ docker push localhost:5000/mssql-restapi
```

Step 6: Configure a storage volume and connect it to the container

Containers that are used for microservices can often function without the need for persistent storage. Database servers such as SQL Server most often require persistent storage for critical data that must be preserved after the container has been shut down or deleted.

Configuring the storage volume

Three steps are required for creating and mounting a persistent volume from XtremIO X2 to the PowerEdge server:

1. Create one or more volumes.
2. Create one or more initiator groups.
3. Map the volumes to the initiator groups.

When creating storage volumes on the XtremIO X2 array, administrators never have to make choices that are typically associated with traditional storage arrays, such as defining the number and type of drives for a storage pool. The administrators also do not have to choose a RAID protection type. For databases that span multiple XtremIO X2 volumes, consider creating a consistency group on the array. A consistency group is a set of volumes that are grouped to ensure write consistency when a snapshot is requested or the disks in the group are replicated.

Note: Although provisioning XtremIO X2 storage is fast and easy, the process requires the coordination between the storage administrator and the developer. Experience shows that delays often occur in this coordination of provisioning containers with external storage and, thus, affect development. In the second use case, we explore how teams can use Kubernetes and the CSI plug-in for XtremIO X2 to solve these challenges.

Connecting persistent storage to the container

For this first use case, the storage administrator manages the storage without the aid of Kubernetes and the CSI plug-in. Therefore, after configuring and connecting the XtremIO X2 storage to the PowerEdge server, the storage administrator must manually connect the storage volumes to the SQL Server container. The administrator can do so by using Docker volumes or Linux bind mounts.

Docker volumes (in container storage)

Docker volumes provide the ability to define storage to be managed by a Docker container. The storage is maintained under the Docker directory structure (for example, `/var/lib/docker/volumes`) and can be managed from the Docker CLI or through the Docker API. The volumes are managed by the Docker engine and are isolated from direct access by the host, as shown in the following figure:

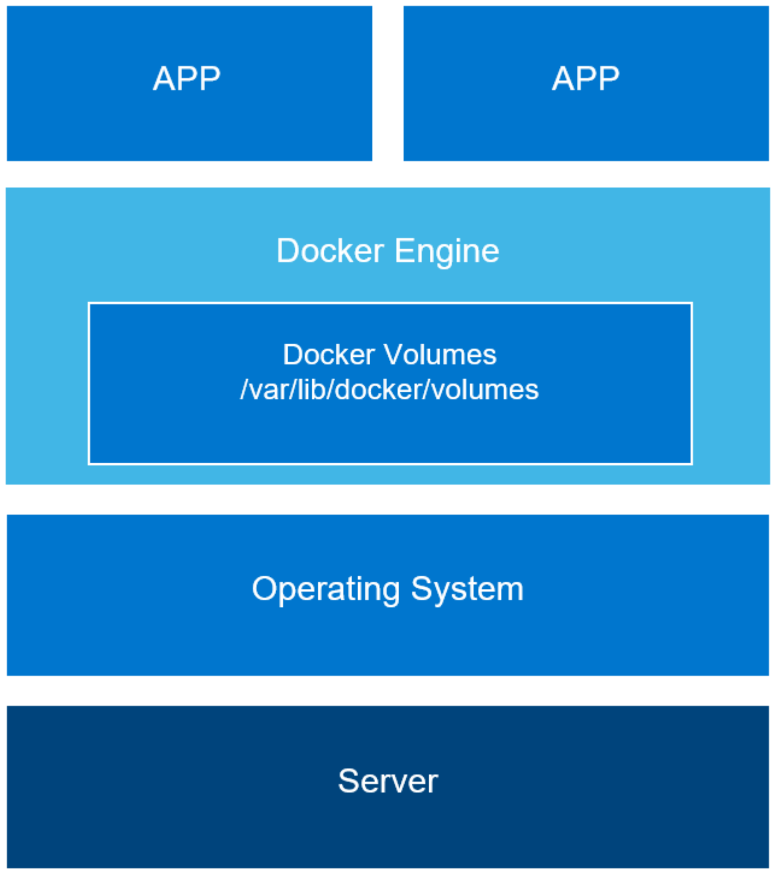


Figure 3. Docker volumes

Advantages of Docker data volumes include:

- Volumes work on both Linux and Windows containers.
- Volumes can be safely shared across containers.
- New volumes can have their content prepopulated by any container.

For a full list of benefits, see Volumes in the Docker documentation.

Linux bind mounts (in host storage)

In this case, the Linux administrator uses Linux bind mounts to connect a SQL Server container to XtremIO X2 storage that has already been provisioned to the server. As mentioned in the Docker guide, bind mounts are fast, which makes this method ideal for attaching storage to a container. As shown in the following figure, bind mounts can be anywhere in the host operating system and are not managed by Docker:

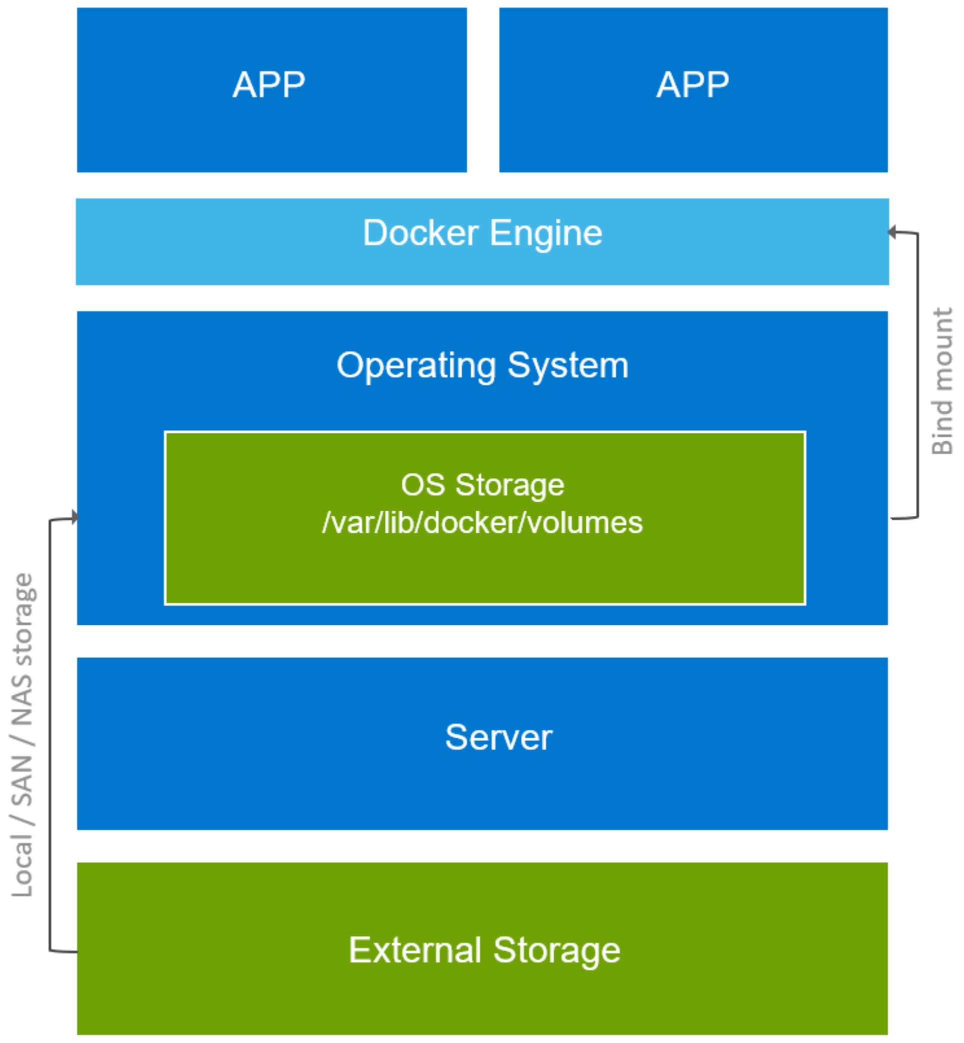


Figure 4. Bind mount

The Linux administrator runs the following commands to mount the XtremIO X2 `sdbl` directory to the SQL Server container:

```
$ cd /sdb1
```

```
$ mkdir mssql-data-dir
$ mount -o bind /sdb1/mssql-data-dir /var/opt/
```

For persistence after reboots, the Linux administrator updates the `/etc/fstab` file with the bind mount as follows:

```
/var/opt          /sdb1/mssql-data-dir    none  bind  0 0
```

Then the Kubernetes administrator creates the SQL volume with the `-v` parameter:

```
$ docker run --name sqlservername -e “ACCEPT_EULA” -e
“MSSQL_SA_PASSWORD=password” -v volumename:/var/opt/mssql-data-dir
-p 1433:1433 -d localhost:5000/sql2019
```

Because bind mounts are external to Docker, an external process or person can modify the files. For SQL Server on Linux, the Linux administrator secures the directory and its files as follows:

1. Change the owner of the directory and its files to `mssql`:

   ```
   chown -R mssql /mssql-data-dir
   ```

2. Change the group of the directory and its files to `mssql`:

   ```
   chgrp -R mssql /mssql-data-dir
   ```

3. Change permissions for the owner, group, and others:

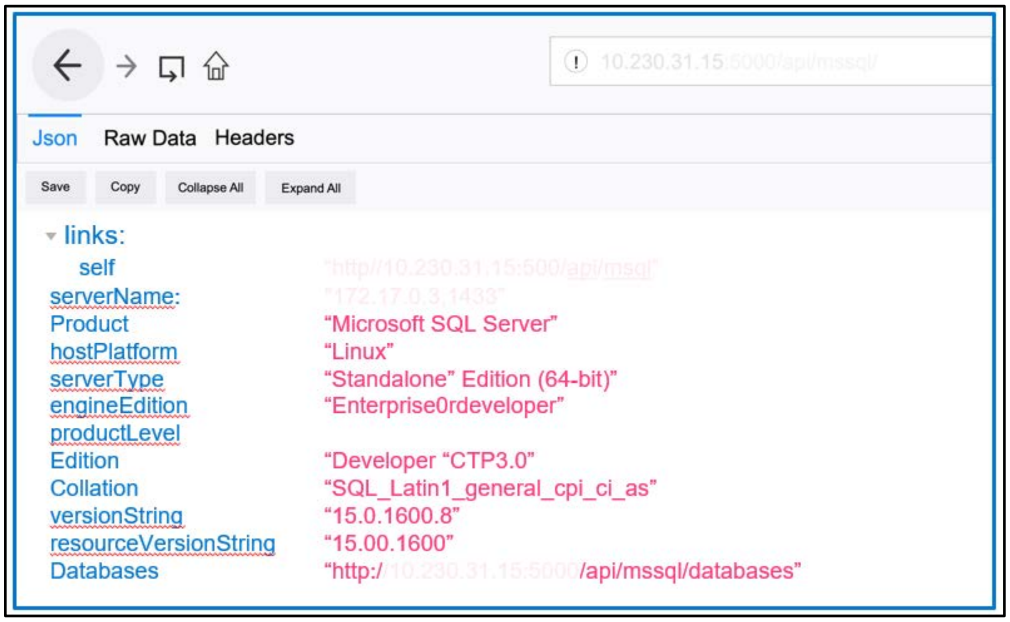
   ```
   chmod -R 750 mssql /mssql-data-dir
   ```

To show how the XtremIO X2 array can be used to quickly provide storage to a SQL Server container, we use the AdventureWorks database. AdventureWorks, created by Microsoft, is a sample database that is based on a fictitious multinational manufacturing company called AdventureWorks Cycles. The AdventureWorks database, which can be installed by restoring a database backup or by attaching a datafile, simulates an OLTP application and can be used for learning how SQL Server works.

In our case, we put a copy of the AdventureWorks database on the XtremIO X2 array. Integrated Copy Data Management (iCDM) tracks copies of data on XtremIO X2 and enables deduplication. Thus, by using XtremIO Virtual Copies to create an associated copy of our AdventureWorks database, the copy takes a fraction of space on the array. Using XtremIO Virtual Copies enables us to create copies quickly and provision multiple copies from one primary AdventureWorks database.

Step 7: Verify database access

To complete Use Case 1, the developer verifies that the development environment is working by accessing data in the AdventureWorks database through a web browser, as shown in the following figure:



```
▾ links:
    self                "http//10.230.31.15:500/api/msql"
  serverName:           "172.17.0.3,1433"
  Product               "Microsoft SQL Server"
  hostPlatform          "Linux"
  serverType            "Standalone" Edition (64-bit)"
  engineEdition         "Enterprise0rdeveloper"
  productLevel
  Edition               "Developer "CTP3.0"
  Collation             "SQL_Latin1_general_cpi_ci_as"
  versionString         "15.0.1600.8"
  resourceVersionString "15.00.1600"
  Databases             "http:/10.230.31.15:5000/api/mssql/databases"
```

Figure 5. Accessing AdventureWorks data through a web browser

Step 8: Clean up the environment

Docker containers are designed to be ephemeral—that is, they can be stopped and destroyed easily, and new containers can be built from the same image and put in place with minimal set-up and configuration. The ephemeral nature of Docker containers is ideal for dev/test scenarios, where most developers and other team members use the containers for only a short time.

The final part of our use case is to clean up the resources from the first round of our dev/test by removing the containers. We do this quickly and easily, as follows:

```
$ docker rm /SQLServer2019
$ docker rm /mssql-restapi
```

Use Case 1 review

The key benefit in our first use case was the transformation from the traditional installation and configuration of a SQL Server environment to using Docker containers. The traditional build process is complex and involves a great deal of time and planning. With Docker containers, the traditional build process is transformed into a self-service on-demand experience that enables developers and others to rapidly deploy applications. Using Docker containers offers many advantages. The primary benefit in this first use case is the capability of having a SQL Server container and persistent storage running in a matter of minutes.

While setting up the Docker container environment for this use case, we learned valuable lessons regarding server environment configuration and its impact on the cost of Docker licensing. Use Case 1 planning also demonstrated the importance of selecting the Docker registry location and storage provisioning options that are most appropriate for the requirements of a particular dev/test environment.

Server environment configuration

Dell Technologies offers a broad selection of servers, enabling customers to configure their compute to match business requirements. The vast PowerEdge server configuration choices means that you can optimize the Docker Enterprise Edition per-core licensing. We recommend investing time into designing a PowerEdge server environment that maximizes your Docker licensing investment. The key to getting the greatest return on your Docker environment is consolidation that maximizes the efficiency of CPU utilization.

Docker registry location

When selecting the location for a Docker registry, consider ease of use and support, speed of container provisioning, and frequency of container provisioning. Container provisioning speed and frequency requirements help in determining where the registry resides. For example, for low-speed and low-frequency provisioning, a cloud-based registry approach might be ideal. High-speed provisioning coupled with high-frequency provisioning can mean that a local private registry using a LAN is best.

Storage provisioning

Provisioning storage from the XtremIO X2 array is fast and easy but is also a manual process that requires coordination between the developer and the storage administrator. In this use case, we demonstrated the manual provisioning of storage, which can work well for small development environments. For larger development environments, or for customers who are interested in automation regardless of the environment size, Use

Case 2 shows how Kubernetes combined with the XtremIO X2 CSI plug-in accelerates storage provisioning.

In Use Case 1, we used bind mounts, enabling Docker to use in-host storage and providing fast performance. When bind mounts are used, any server processor or person can access the directory. However, administrators can manage this access by securing database files at the owner and group levels and by using directory and file permissions.

Business decision summary

The following table provides a high-level summary of the decisions we made when implementing containerized SQL Server in Use Case 1.

Table 4. Summary of business decisions in Use Case 1

Choice	Decision	Explanation
Docker Community Edition or Enterprise Edition	Enterprise Edition	Provides certified images and business support
Cloud-based private registry or local private registry	Local private registry	Offers the fastest provisioning of containers, although increases complexity and support requirements
XtremIO X2 manual or automated provisioning	Manual storage provisioning	Is appropriate for the limited requirements of this use case
Docker volumes or bind mounts	Bind mounts	Provides host-based high performance

Use Case 2: Automated provisioning of a containerized dev/test environment

In the preceding use case, we showed the manual provisioning of a container with persistent storage from the XtremIO X2 array. The next step in our journey is automating the provisioning of containers and storage, further accelerating the provisioning of the software development environment. Container orchestration becomes essential when hundreds or thousands of containers must be managed. In this use case, a developer provisions a SQL Server container along with database storage by using Kubernetes with the CSI plug-in as a platform as a service (PaaS).

In Use Case 2:

1. The Kubernetes administrator performs a custom installation of Kubernetes.
2. The storage administrator works with the Kubernetes administrator to install the XtremIO X2 CSI plug-in.
3. The developer provisions a SQL Server and REST API container, using Kubernetes.
4. The developer creates a copy of the AdventureWorks database, using the CSI plug-in.

5. The developer modifies the database data and protects the database state, using XtremIO Virtual Copies.
6. The tester performs destructive testing.
7. The developer recovers the modified SQL Server database.
8. The technical writer provisions the modified SQL database.
9. The developer removes the containers and cleans up the environments.

Step 1: Install Kubernetes

Many Kubernetes solutions are available today. For example, turnkey managed Kubernetes offerings from cloud providers give IT organizations a zero-data-center-footprint solution that requires no installation. An on-premises private cloud Kubernetes implementation offers greater control and flexibility but requires investment in infrastructure and training. For this use case, we show a basic Kubernetes installation to demonstrate how having the container orchestration system on our LAN provides greater performance and control as well as the ability to customize the configuration.

Note: For complete Kubernetes installation instructions, see the Kubernetes documentation.

We install Kubernetes as follows:

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-
el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kube*
EOF
# Set SELinux in permissive mode (effectively disabling it)
setenforce 0
sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/'
/etc/selinux/config

yum install -y kubelet kubeadm kubectl --
disableexcludes=kubernetes

systemctl enable --now kubelet
```

Step 2: Install the XtremIO X2 CSI plug-in

In addition to our Kubernetes environment, we also need a CSI plug-in to complete our automation journey. CSI plug-ins are a Kubernetes defined standard that Dell Technologies and others use to expose block and file storage to container orchestration systems. CSI plug-ins unify storage management across many container orchestration systems, including Mesos, Docker Swarm, and Kubernetes.

The XtremIO X2 CSI plug-in for Kubernetes provides the following orchestration capabilities:

- Dynamic provisioning and decommissioning of volumes
- Attaching and detaching volumes from a host node
- Mounting and unmounting a volume from a host node

The Kubernetes administrator works with the storage administrator to download, modify, and install the XtremIO X2 CSI plug-in as follows.

Note: Detailed installation instructions are available on YouTube at Dell EMC XtremIO CSI Plugin Installation for Kubernetes and in the following white paper: *Dell EMC XtremIO X2 Container Storage Interface Plugin Integration with Kubernetes*.

1. Download the plug-in from GitHub:

   ```
   $ git clone https://github.com/dell/csi-xtremio-deploy.git
   ```

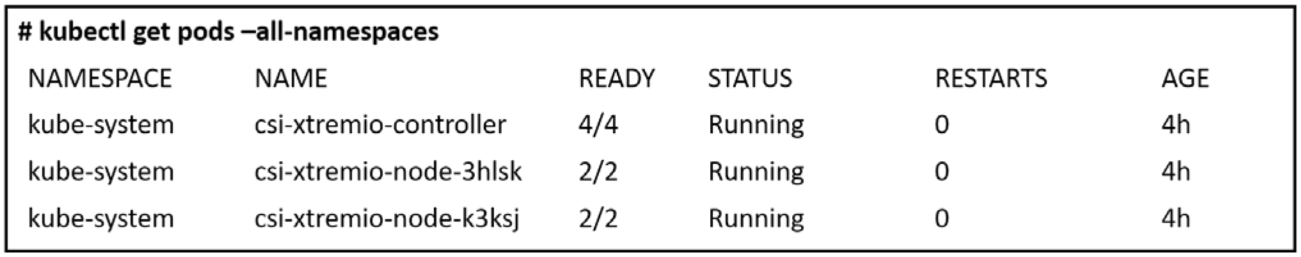
2. Modify the CSI.ini file to connect to the XtremIO X2 array.
3. Install the CSI plug-in on the first node run:

   ```
   $ ./install -csi-plugin.sh -c
   ```

   and on all other nodes in the cluster:

   ```
   $ ./install -csi-plugin.sh -n
   ```

The XtremIO X2 storage array is then available in Kubernetes. The following figure shows the `kubectl get pods –all-namespaces` output when the CSI plug-in is running on all Kubernetes worker nodes:

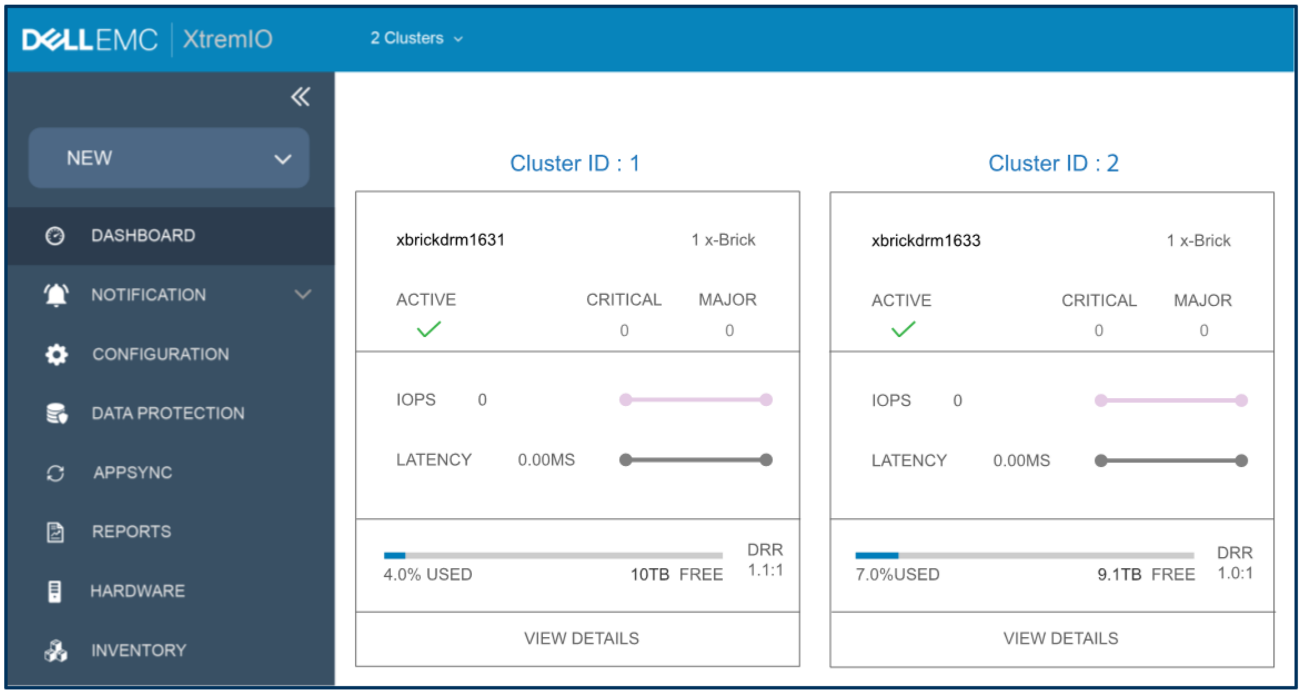


```
# kubectl get pods –all-namespaces
NAMESPACE     NAME                        READY   STATUS    RESTARTS   AGE
kube-system   csi-xtremio-controller      4/4     Running   0          4h
kube-system   csi-xtremio-node-3hlsk      2/2     Running   0          4h
kube-system   csi-xtremio-node-k3ksj      2/2     Running   0          4h
```

Figure 6. XtremIO X2 CSI plug-in running on Kubernetes worker nodes

The XtremIO X2 array uses intelligent placement for volumes that are provisioned by the XtremIO X2 CSI plug-in. An XtremIO X2 storage environment can support up to four clusters with a single management interface, the XtremIO Management Service (XMS). By default, persistent volumes that are created with the CSI plug-in are placed on the cluster that has the freest capacity as determined by the XMS. This intelligent placement of volumes by the XMS server means that the array is automatically load-balanced. Alternatively, you can explicitly select the cluster for a Kubernetes storage class by specifying the cluster ID.

The following figure shows a multicluster configuration in the XMS interface:



Cluster ID : 1

xbrickdrm1631 1 x-Brick
ACTIVE ✓ CRITICAL 0 MAJOR 0
IOPS 0
LATENCY 0.00MS
4.0% USED 10TB FREE DRR 1.1:1

Cluster ID : 2

xbrickdrm1633 1 x-Brick
ACTIVE ✓ CRITICAL 0 MAJOR 0
IOPS 0
LATENCY 0.00MS
7.0%USED 9.1TB FREE DRR 1.0:1

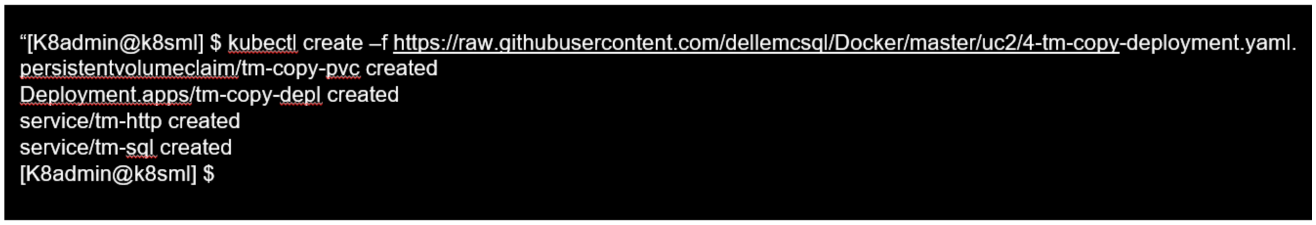
Figure 7. XtremIO X2 multicluster configuration

To use XtremIO Virtual Copies, the Kubernetes administrator must specify a Snapshot Class as a provisioner and then create a `VolumeSnapshot` YAML file. This process enables the developer to take a snapshot of the database to a persistent volume.

The Docker administrator creates a basic `VolumeSnapshot` YAML file as follows:

```
apiVersion: storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
      name: mssql-snapshot
      namespace: default
spec:
snapshotClassName: csi-xtremio-xvc
      kind: PersistentVolumeClaim
```

We have uploaded the following YAML file to GitHub as an example for customers:

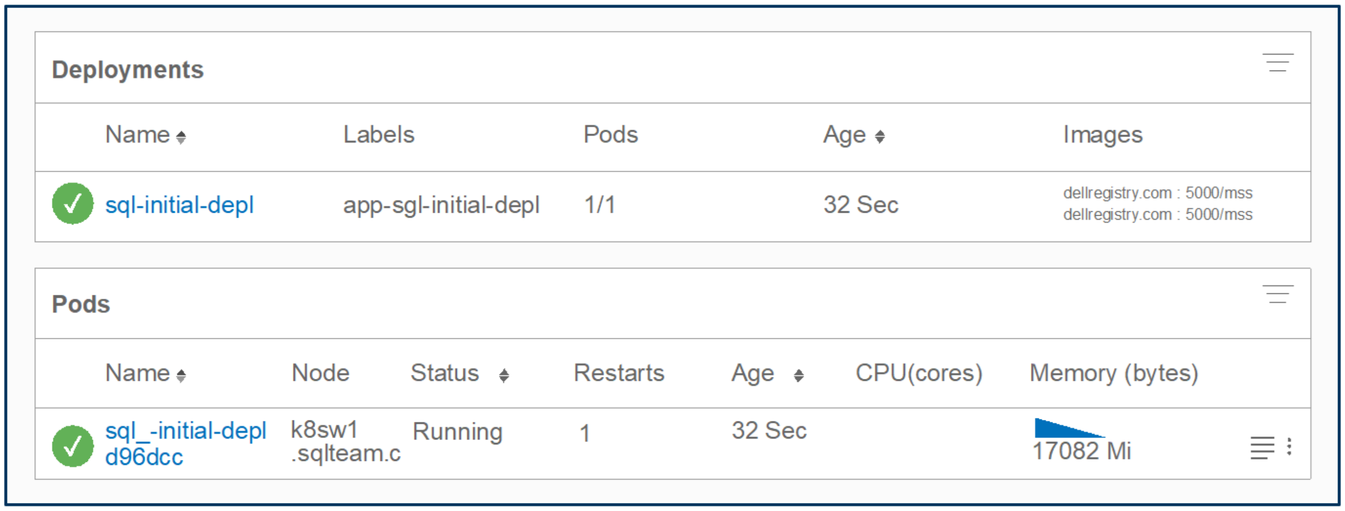


```
"[K8admin@k8sml] $ kubectl create –f https://raw.githubusercontent.com/dellemcsql/Docker/master/uc2/4-tm-copy-deployment.yaml.
persistentvolumeclaim/tm-copy-pvc created
Deployment.apps/tm-copy-depl created
service/tm-http created
service/tm-sql created
[K8admin@k8sml] $
```

Figure 8. Kubernetes YAML file used to create snapshot

Step 3: Provision SQL Server and REST API containers

Using Kubernetes, the developer provisions the SQL Server container together with the REST API container (in the same middle tier as described in Use Case 1) in a pod. A pod is a group of one or more containers using shared networking, storage, and configurations. In our pod, the REST API has been configured to connect to the SQL Server, as shown in the following figure, tightly integrating the two applications. The value in using pods is the capability of quickly provisioning shared application services, minimizing human configuration errors and post-configuration efforts. Kubernetes transforms the manual provisioning of our containers from Use Case 1 into an easy-to-use service.



Deployments

	Name	Labels	Pods	Age	Images
✓	sql-initial-depl	app-sgl-initial-depl	1/1	32 Sec	dellregistry.com : 5000/mss dellregistry.com : 5000/mss

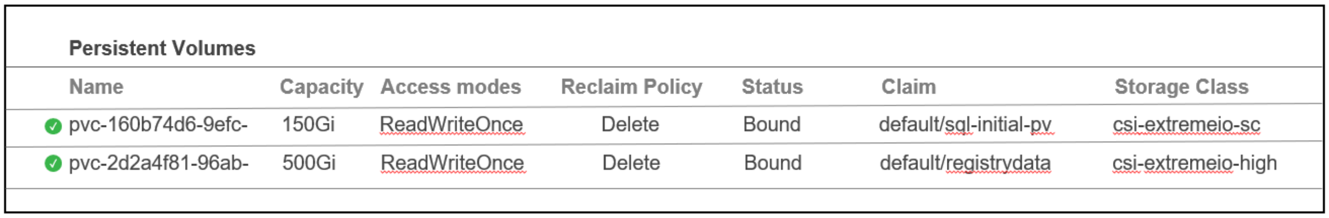
Pods

	Name	Node	Status	Restarts	Age	CPU(cores)	Memory (bytes)
✓	sql_-initial-depl d96dcc	k8sw1 .sqlteam.c	Running	1	32 Sec		17082 Mi

Figure 9. Provisioning the pod of SQL Server and REST API containers

Step 4: Provision a database copy

In this step, the developer provisions a copy of the AdventureWorks database by using Kubernetes and XtremIO Virtual Copies, as shown in the following figure:



Persistent Volumes

	Name	Capacity	Access modes	Reclaim Policy	Status	Claim	Storage Class
✓	pvc-160b74d6-9efc-	150Gi	ReadWriteOnce	Delete	Bound	default/sql-initial-pv	csi-extremeio-sc
✓	pvc-2d2a4f81-96ab-	500Gi	ReadWriteOnce	Delete	Bound	default/registrydata	csi-extremeio-high

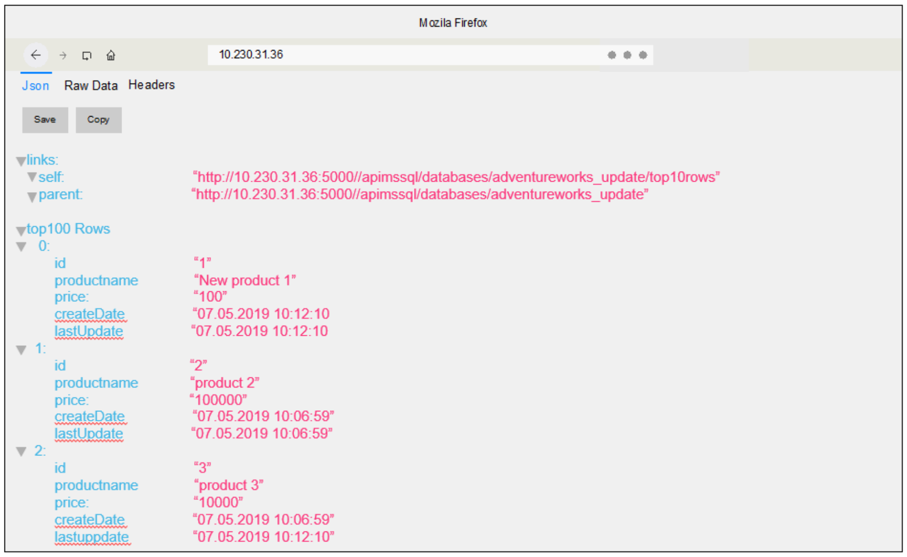
Figure 10. Provisioning an AdventureWorks database by using Kubernetes

This quick and easy approach to provisioning the database easily scales for additional development environments due to the flexibility and lightweight resource consumption of Docker containers. Businesses benefit from simultaneous time savings and reduced complexity.

Step 5: Modify database data and protect the database state

After provisioning the AdventureWorks database, the developer modifies the database data.

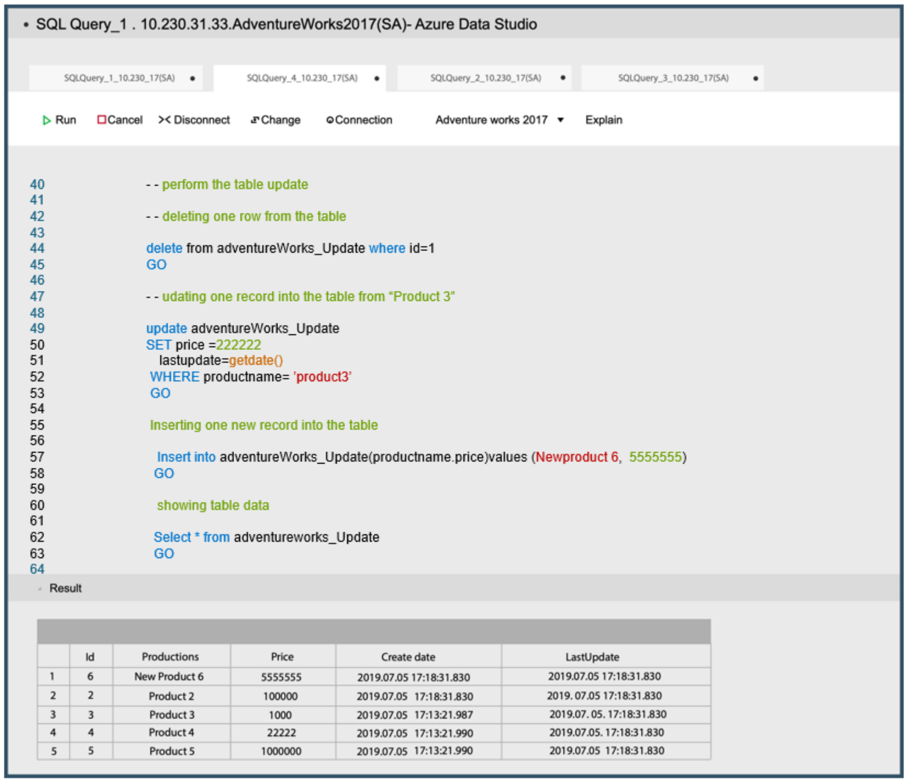
The following figure shows the data before it is modified:



```
▼links:
  ▼self:          "http://10.230.31.36:5000//apimssql/databases/adventureworks_update/top10rows"
  ▼parent:        "http://10.230.31.36:5000//apimssql/databases/adventureworks_update"

▼top100 Rows
 ▼ 0:
      id            "1"
      productname   "New product 1"
      price:        "100"
      createDate    "07.05.2019 10:12:10
      lastUpdate    "07.05.2019 10:12:10
 ▼ 1:
      id            "2"
      productname   "product 2"
      price:        "100000"
      createDate    "07.05.2019 10:06:59"
      lastUpdate    "07.05.2019 10:06:59"
 ▼ 2:
      id            "3"
      productname   "product 3"
      price:        "10000"
      createDate    "07.05.2019 10:06:59"
      lastuppdate   "07.05.2019 10:12:10"
```

Figure 11. Data before modification

Here is the T-SQL that the developer uses to modify the table data:

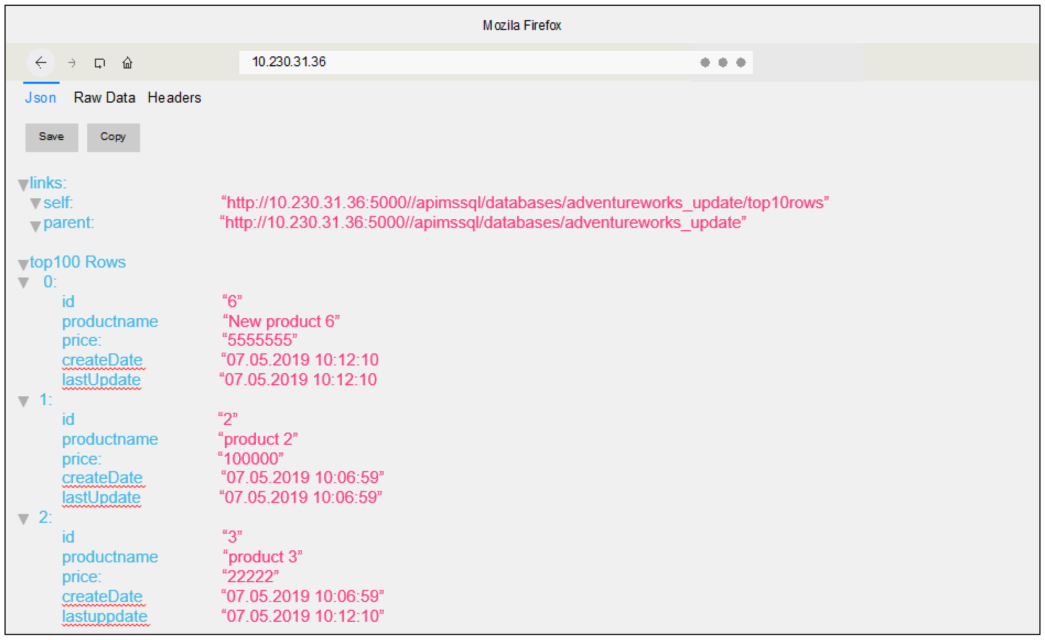


```
- - perform the table update

- - deleting one row from the table

delete from adventureWorks_Update where id=1
GO

- - udating one record into the table from "Product 3"

update adventureWorks_Update
SET price =222222
   lastupdate=getdate()
 WHERE productname= 'product3'
 GO

 Inserting one new record into the table

  Insert into adventureWorks_Update(productname.price)values (Newproduct 6,  5555555)
 GO

  showing table data

 Select * from adventureworks_Update
 GO
```

	Id	Productions	Price	Create date	LastUpdate
1	6	New Product 6	5555555	2019.07.05 17:18:31.830	2019.07.05 17:18:31.830
2	2	Product 2	100000	2019.07.05 17:18:31.830	2019. 07.05 17:18:31.830
3	3	Product 3	1000	2019.07.05 17:13:21.987	2019.07. 05. 17:18:31.830
4	4	Product 4	22222	2019.07.05 17:13:21.990	2019.07.05. 17:18:31.830
5	5	Product 5	1000000	2019.07.05 17:13:21.990	2019.07.05 17:18:31.830

Figure 12. T-SQL used to modify data

The following figure shows the data after it is modified:

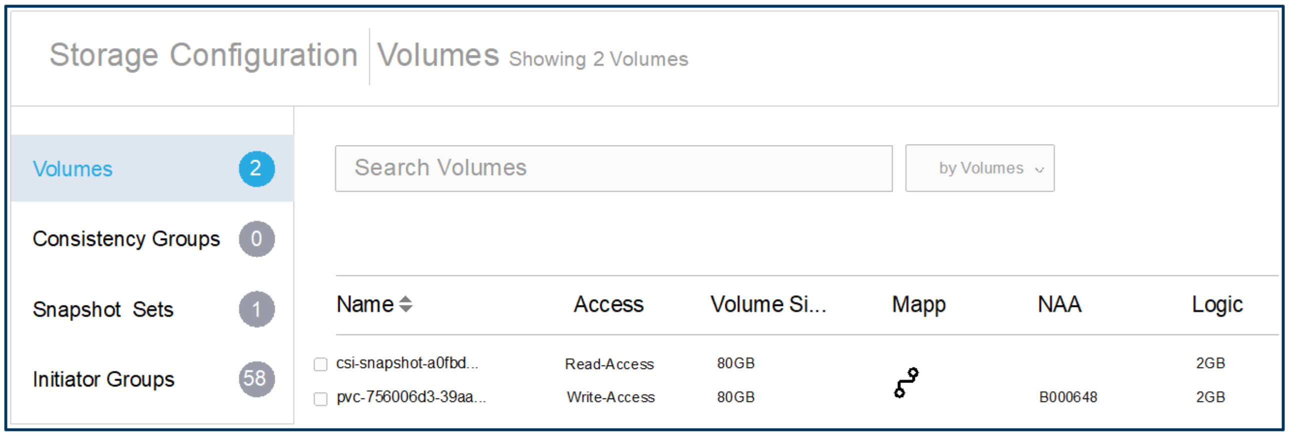


```
▼links:
  ▼self:        "http://10.230.31.36:5000//apimssql/databases/adventureworks_update/top10rows"
  ▼parent:      "http://10.230.31.36:5000//apimssql/databases/adventureworks_update"

▼top100 Rows
▼  0:
      id            "6"
      productname   "New product 6"
      price:        "5555555"
      createDate    "07.05.2019 10:12:10
      lastUpdate    "07.05.2019 10:12:10
▼  1:
      id            "2"
      productname   "product 2"
      price:        "100000"
      createDate    "07.05.2019 10:06:59"
      lastUpdate    "07.05.2019 10:06:59"
▼  2:
      id            "3"
      productname   "product 3"
      price:        "22222"
      createDate    "07.05.2019 10:06:59"
      lastuppdate   "07.05.2019 10:12:10"
```

Figure 13. Modified data

Before releasing the modified environment to the test team, the developer uses XtremIO Virtual Copies to protect the state of the database. XtremIO X2 data reduction services ensure that only unique data is written to the database copy after the snapshot is taken and saved to the array. Also, the unique data that is managed by the snapshot feature is compressed, which further reduces the capacity footprint. Thus, under most circumstances, the XtremIO X2 Virtual Copies snapshot of the database takes little additional storage space and can be both created and destroyed quickly.

The developer reviews the following XtremIO X2 screen to verify the existence of the XtremIO Virtual Copies snapshot of the AdventureWorks database:



Name	Access	Volume Si...	Mapp	NAA	Logic
csi-snapshot-a0fbd...	Read-Access	80GB			2GB
pvc-756006d3-39aa...	Write-Access	80GB		B000648	2GB

Figure 14. Snapshot of the AdventureWorks database shown in XtremIO

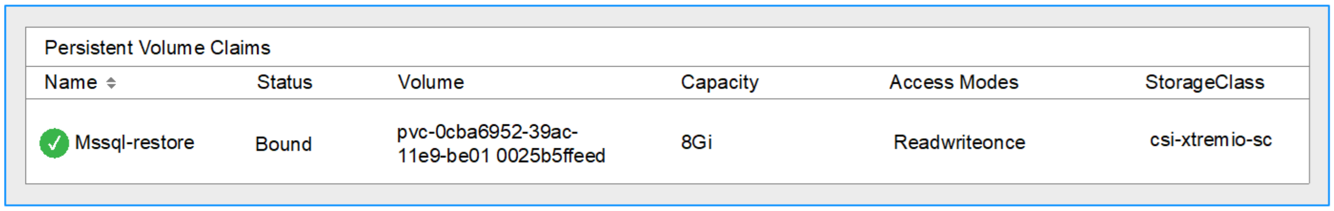
Step 6: Run a destructive test and recover the modified database

In this use case scenario, the quality assurance teams perform automated destructive testing on the most recent version of the AdventureWorks database application, and the tests fail. Therefore, the developer returns to the state that preceded the destructive testing by removing the attached storage and restoring the AdventureWorks database from the XtremIO Virtual Copies snapshot.

The developer uses the following YAML file to restore the database to the former version:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
      name: mssql-restore
spec:
snapshotClassName: csi-xtremio-sc
      dataSource:
            name: mssql-snapshot
            kind: VolumeSnapshot
            apiGroup: snapshot.storage.k8s.io
      accessModes:
ReadWriteOnce
Resources:
      Requests:
            Storage: 8Gi
```

The following figure shows the database restore in Kubernetes:



Persistent Volume Claims

Name	Status	Volume	Capacity	Access Modes	StorageClass
✓ Mssql-restore	Bound	pvc-0cba6952-39ac-11e9-be01 0025b5ffeed	8Gi	Readwriteonce	csi-xtremio-sc

Figure 15. Restoring the database in Kubernetes

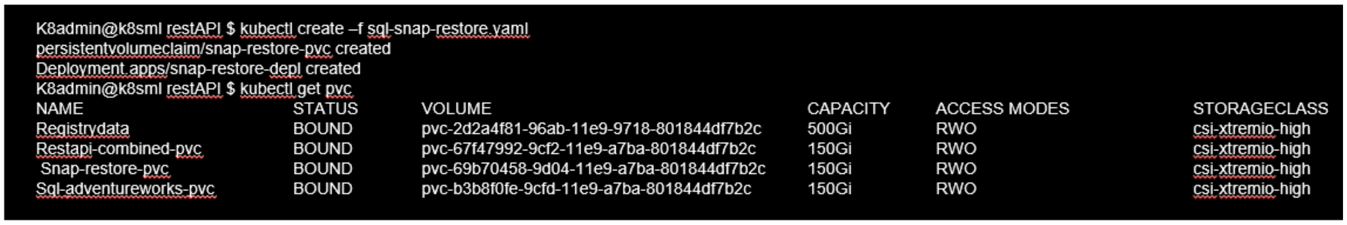
Restoring the database is fast, saving our developer valuable time. The capability of incrementally saving work with the XtremIO Virtual Copies feature provides for more iterative development by:

- Protecting work at logical checkpoints
- Enabling destructive testing with quick recovery to a previous XtremIO Virtual Copies snapshot

Step 7: Technical writer provisions modified SQL database

The use of containers combined with the gains of using the XtremIO X2 CSI plug-in and Kubernetes has enabled our developer to be more productive. It has also freed up valuable storage administrator time to work on production and business-critical needs. The same benefits can apply to other roles as well. Technical writers, for example, often must work with the software to document updates and new features.

The following figure shows how the technical writer provisions an updated pod from Kubernetes:



```
K8admin@k8sml restAPI $ kubectl create –f sql-snap-restore.yaml
persistentvolumeclaim/snap-restore-pvc created
Deployment.apps/snap-restore-depl created
K8admin@k8sml restAPI $ kubectl get pvc
NAME                    STATUS      VOLUME                                      CAPACITY    ACCESS MODES              STORAGECLASS
Registrydata            BOUND       pvc-2d2a4f81-96ab-11e9-9718-801844df7b2c    500Gi       RWO                       csi-xtremio-high
Restapi-combined-pvc    BOUND       pvc-67f47992-9cf2-11e9-a7ba-801844df7b2c    150Gi       RWO                       csi-xtremio-high
 Snap-restore-pvc       BOUND       pvc-69b70458-9d04-11e9-a7ba-801844df7b2c    150Gi       RWO                       csi-xtremio-high
Sql-adventureworks-pvc  BOUND       pvc-b3b8f0fe-9cfd-11e9-a7ba-801844df7b2c    150Gi       RWO                       csi-xtremio-high
```

Figure 16. Technical writer provisioning SQL Server pod

Step 8: Clean up the environment

The final step in our journey is to show the ease of removing the containers and storage. Two XtremIO X2 options—the retain policy and the delete policy—are available for users who have completed working with their volumes.

Retain policy

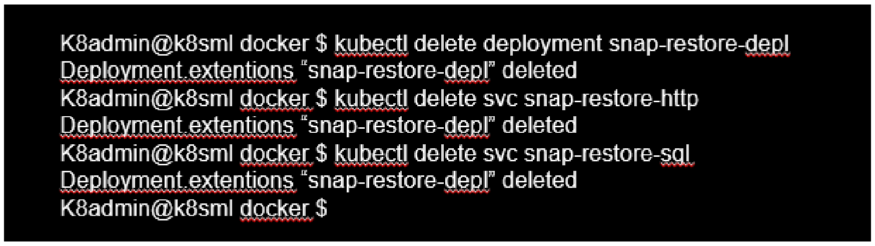
The retain policy enables the user to detach the container from storage but retain the persistent volume so that the storage can be attached to other containers. To enable this functionality, which is not the default behavior, the cluster administrator must configure `StorageClass`, as shown in the following example:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
      name: csi-xtremio-qos-sc-high
provisioner: csi-xtremio.dellemc.com
      reclaimPolicy: Retain
parameters
      qos_policy_id: “QoS_High”
```

If the storage is not required but `StorageClass` is configured so that `reclaimPolicy` is set to `Retain`, the XtremIO X2 LUNs can be manually removed.

Delete policy

The delete policy enables the container to be detached from storage and the associated storage LUN to be removed from the XtremIO X2 array. The delete policy is the default behavior and removes all associated storage. For databases and other applications that require persistent storage, developers and others should work with the cluster administrator to understand if the associated storage is going to be retained or deleted and make any needed changes. In this case, our technical writer does not have to retain a copy of the AdventureWorks database and so uses the delete policy, as shown in the following figure:



```
K8admin@k8sml docker $ kubectl delete deployment snap-restore-depl
Deployment.extentions “snap-restore-depl” deleted
K8admin@k8sml docker $ kubectl delete svc snap-restore-http
Deployment.extentions “snap-restore-depl” deleted
K8admin@k8sml docker $ kubectl delete svc snap-restore-sql
Deployment.extentions “snap-restore-depl” deleted
K8admin@k8sml docker $
```

Figure 17. Technical writer removing SQL Server pod in Kubernetes

Use Case 2 review

In our second use case, Kubernetes combined with the XtremIO X2 plug-in simplified and automated the provisioning and removal of containers and storage.

This use case demonstrated how we can easily shift away from the complexities of scripting and using the command line to a self-service model that accelerates container management. The move to a self-service model, which increases developer productivity by removing bottlenecks, becomes increasingly important as the Docker container environment grows.

Using a container orchestration system such as Kubernetes is the next step in the container journey for database developers. Automation becomes a necessity with the growth of containerized applications. In this case, it enabled our developer to bypass the complexities that are associated with scripting and uses the Kubernetes application to accomplish the developer’s objectives. The CSI plug-in integrates with Kubernetes and exposes the capabilities of the XtremIO X2 array, enabling the developer to:

- Provision an XtremIO Virtual Copies snapshot of the AdventureWorks database from a gold master on XtremIO X2
- Protect work in the AdventureWorks database by creating an XtremIO Virtual Copies snapshot
- Restore a database from the saved copy of the AdventureWorks database
- Remove the containers and attached storage

The powerful part of this enablement is that steps such as these have traditionally required multiple roles—developers and others working with the storage administrator, for example—and more time. Kubernetes with the CSI plug-in enables the developer and others to do more in less time. The time savings means that projects can be completed faster, benefiting both the developer and the business. Overall, the key benefit of our second use case was the transformation from a manually managed container environment to an orchestrated system with more storage capabilities.

CSI plug-ins: Additional options from Dell Technologies

Dell Technologies also offers customers CSI plug-ins for the Dell EMC PowerFlex family (formerly referred to as the VxFlex family) to provide a broad range of features in Kubernetes. The PowerFlex rack and PowerFlex appliance create a server-based SAN by using PowerFlex (the software foundation formerly known as VxFlex OS) with PowerEdge servers. Local server and storage resources are combined to create virtual pools of block storage with varying performance tiers. For more information, see *PowerFlex and VxFlex Ready Nodes for Kubernetes*. The CSI plug-in for PowerFlex rack and PowerFlex appliance is available from Docker Hub.

For more information about CSI plug-ins from Dell Technologies, see Dell EMC Storage Automation and Developer Resources.

Conclusion

Innovation drives transformation. In the case of Docker containers and Kubernetes, the key benefit is a shift to rapid application deployment services. Microsoft and many others have embraced containers and provide images of applications such as SQL Server that can be integrated in days and instantiated in seconds. Installations and other repetitive tasks are replaced with packaged applications that have the developer working quickly in the database. The ease of using Docker and Kubernetes combined with rapid provisioning of persistent storage transforms development by removing wait time and enabling the developer to move closer to the speed of thought.

While the shift to Docker containers in Use Case 1 benefited our developers, provisioning and attaching storage was a manual process that slowed the overall speed of application development. The challenges with manual storage provisioning are twofold: variety and velocity. As the IT organization adds more application images, variety increases administration and support complexity. Velocity, the frequency of provisioning applications, tends to increase with greater selection. Increased velocity is a growth indicator but also places pressure on the IT organization to address automation.

The second transformation in our journey was the addition of the Kubernetes orchestration system and the XtremIO X2 CSI plug-in. Kubernetes brings a rich user interface that simplifies provisioning containers and persistent storage. In our testing, we found that Kubernetes plus the XtremIO X2 CSI plug-in enabled a self-service, on-demand capability. This capability enabled developers to provision containerized applications with persistent storage through point-and-click simplicity and freed valuable storage administrator time to focus on business-critical tasks.

Kubernetes, enhanced with the XtremIO X2 CSI plug-in, provides the capability to attach and manage all-flash XtremIO X2 volumes to containerized applications. Our developer worked with a familiar Kubernetes interface to create a copy of the AdventureWorks database and connect it to the SQL Server container. After modifying the database, the developer could protect progress by using XtremIO Virtual Copies to save a copy of the database.

Moving to a Docker plus Kubernetes infrastructure provides a faster and more consistent way to package and deploy SQL Server. Microsoft and the open-source community that supports Docker and Kubernetes have done much of the foundational work. Manual or scripted installation procedures are not necessary, leaving only customization to the business. Dell Technologies adds storage value through XtremIO X2 enterprise storage technology and the CSI plug-in, streamlining the delivery of applications. The goal of this white paper is to jump-start your application development transformation to enable you to achieve all these benefits. Dell can show you how and can provide an infrastructure that optimizes your containerized applications.

The road ahead

General availability of SQL Server 2019 will enable additional production uses cases such as Production Always On groups, where the Always On Listener construct becomes a load balancer for the primary application. Rolling upgrades will become the norm for production environments, greatly simplifying the SQL Server patch and update process. Also, in the near future, XtremIO Virtual Copies will seed larger Always On replicas, as it currently does with virtual machine nodes.

Appendix A: Solution architecture and component specifications

Architecture diagram

The following figure shows the design architecture:

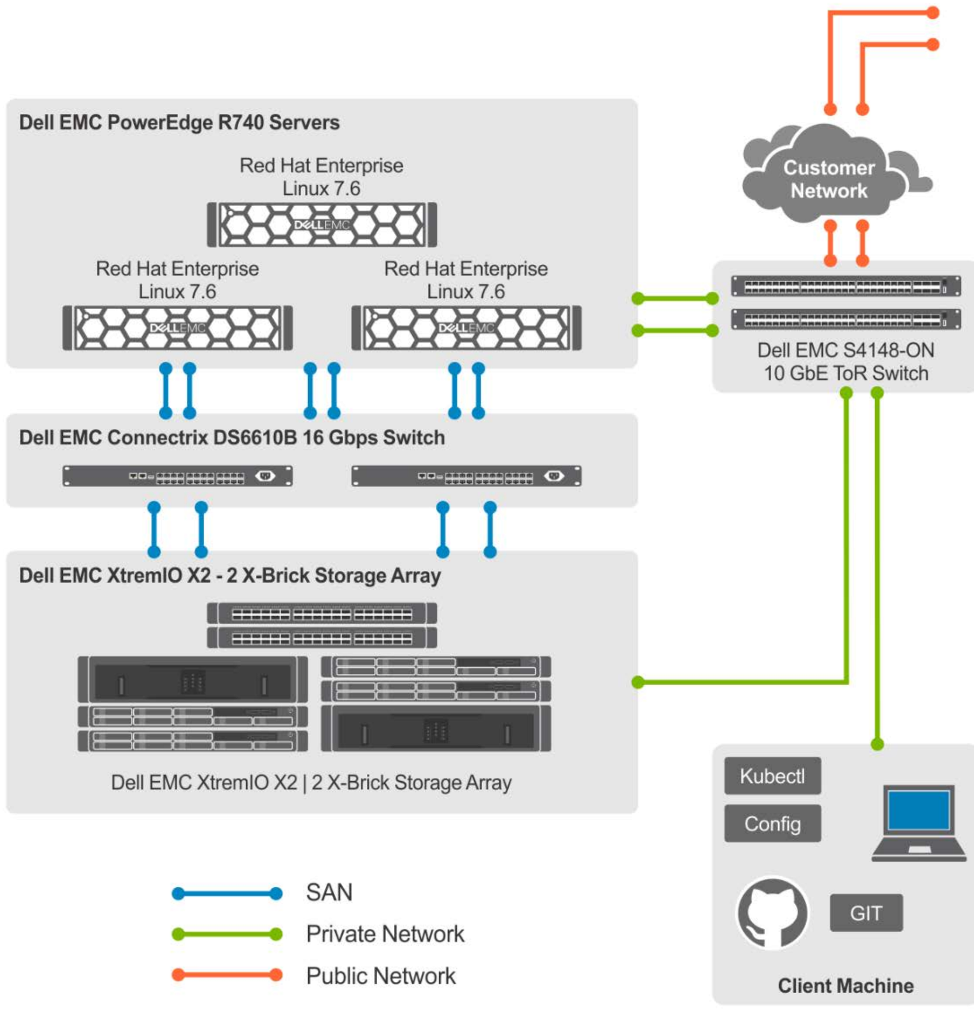

Dell EMC PowerEdge R740 Servers
Red Hat Enterprise Linux 7.6
Red Hat Enterprise Linux 7.6
Red Hat Enterprise Linux 7.6
Customer Network
Dell EMC S4148-ON 10 GbE ToR Switch
Dell EMC Connectrix DS6610B 16 Gbps Switch
Dell EMC XtremIO X2 - 2 X-Brick Storage Array
Dell EMC XtremIO X2 | 2 X-Brick Storage Array
SAN
Private Network
Public Network
Kubectl
Config
GIT
Client Machine

Figure 18. Solution architecture

Server layer

The server layer consists of three PowerEdge R740 servers. The PowerEdge R740 is a 2U rack server that supports up to:

- Two Intel Xeon Processor Scalable processors
- 24 DIMM slots supporting up to 1,536 GB of memory
- Two AC or DC power supply units
- 16 SAS, SATA, or near-line SAS hard drives or SSDs

We configured three bare-metal servers as the three-node Kubernetes cluster. Each node hosted Red Hat Enterprise Linux 7.6.

The following table lists the PowerEdge R740 configuration details. While the upper limits of resources for the R740 server are significantly higher, this configuration is sufficient for supporting a functional design showing the advantages of containerized SQL Server instances for use in dev/test environments.

Table 5. PowerEdge R740 server configuration

Component	Details
Chassis	2-CPU configuration
Memory	12 DDR4 Dual Rank 32 GB @ 2,400 MHz
Processors	2 Intel Xeon Silver 4110 CPU @ 2.10 GHz with 8 cores
Host bus adapters	2 Emulex LightPulse LPe31002-M6-D 2-Port 16 Gb Fibre Channel
rNDC	BRCM GbE 4P 5720-t rNDC
Add-on NIC	Intel 10 GbE 2P X710
Power supplies	2 x Dell 750 W, RDNT, DELTA
RAID controller	Dell H740P
iDRAC	iDRAC9 Enterprise
Physical disks	• 2 x 900 GB SSDs • 4 x 1.92 TB HDDs

Network layer

The network layer consists of the following switches:

- **Two 10 GbE network switches**—Connect to two 10 Gb ports on the Kubernetes node to route the network traffic
- **Two 16 Gb/s Fibre Channel (FC) fabric switches**—Route SAN traffic between the R740 servers and the XtremIO X2 storage array

The following figure shows the recommended FC connectivity between the host bus adapters (HBAs) and the FC switches. It also shows the connectivity between the FC switches and the XtremIO X2 storage array. As shown, each server HBA port connects to two separate FC switches, and the two front-end ports on each XtremIO X2 array controller connect to those same FC switches.

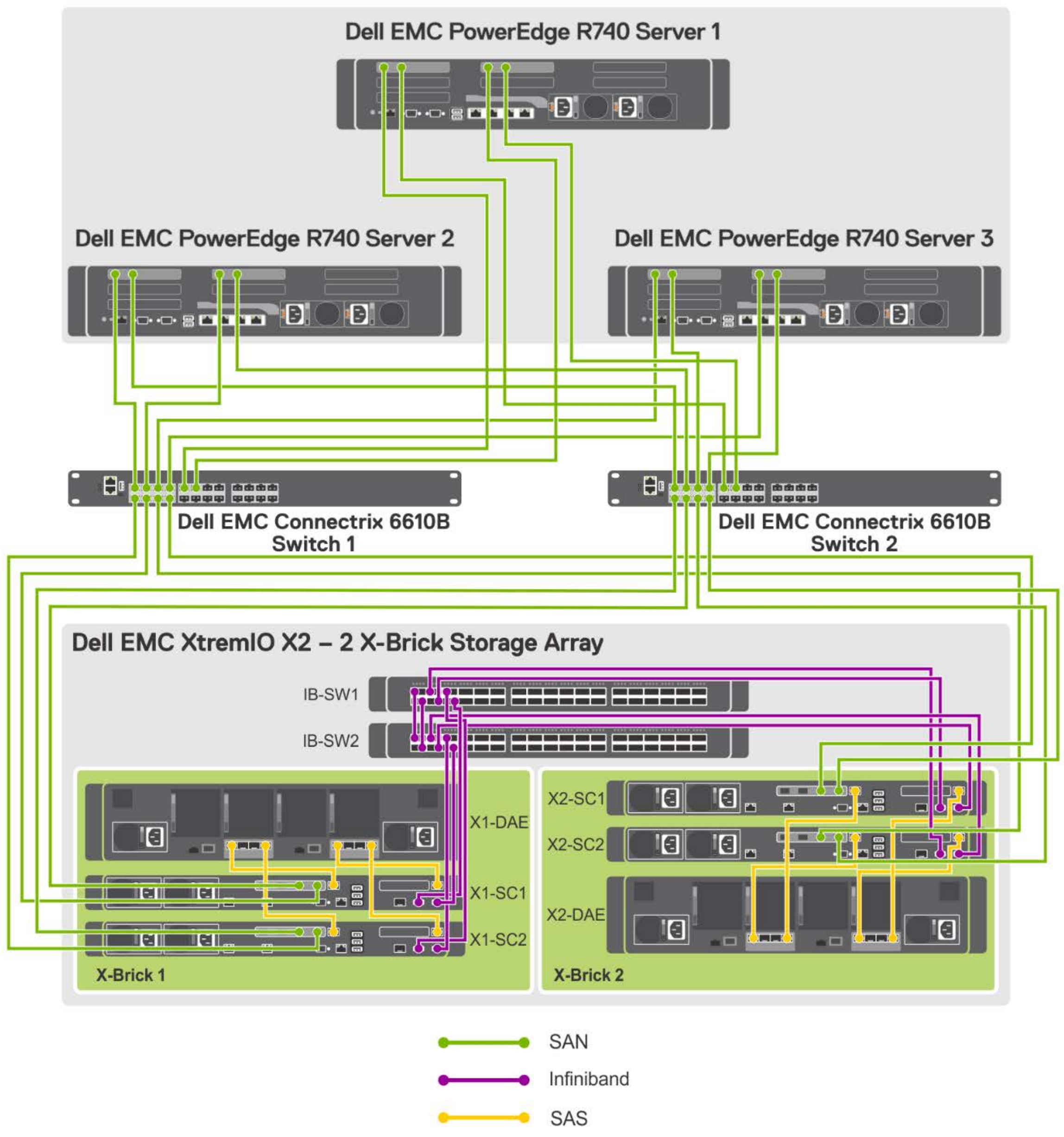

Dell EMC PowerEdge R740 Server 1
Dell EMC PowerEdge R740 Server 2
Dell EMC PowerEdge R740 Server 3
Dell EMC Connectrix 6610B Switch 1
Dell EMC Connectrix 6610B Switch 2
Dell EMC XtremIO X2 – 2 X-Brick Storage Array
IB-SW1
IB-SW2
X1-DAE
X1-SC1
X1-SC2
X-Brick 1
X2-SC1
X2-SC2
X2-DAE
X-Brick 2
SAN
Infiniband
SAS

Figure 19. FC fabric connectivity design

Storage layer

We used one XtremIO X2 storage array as FC SAN storage to test SQL Server 2019 on Kubernetes using the CSI plug-in.

The storage layer consists of the following components:

- An XtremIO X2 cluster with two X-Brick modules, with 72 x 2 TB flash-based SAS SSDs
- Two controllers and one disk array enclosure (DAE) on each X-Brick module
- Four 16 Gb/s front-end FC ports
- Two InfiniBand switches for two X-Brick connections

Redundancy

The LAN and SAN design includes redundant components and connectivity at every level to ensure that no single point of failure exists. This design ensures that the application server can reach the database server and the database server can reach the storage array if a component fails. The design provides protection even if a failure occurs in one or more NICs or HBA ports, one LAN or FC switch, one or more XtremIO X2 front-end ports, or one XtremIO X2 X-Brick controller.

Multipathing

The XtremIO X2 array supports native multipathing technology. Multipathing increases the efficiency of sending data over redundant hardware paths that connect PowerEdge servers to XtremIO X2 storage. Benefits include alternating I/O by using round-robin to optimize use of the hardware paths and to distribute the data evenly. Also, if any component along the storage path fails, then NMP resets the connection and passes I/O using an alternate path.

Software components

The following table provides details about the solution software components:

Table 6. Software components

Component	Details
Red Hat Enterprise Linux Server	7.6 (3.10.0-957.el7.x86_64) (Maipo)
Docker Enterprise Edition	18.09.7
Kubernetes	1.15.0
Flannel (pod network orchestration)	0.11.0-amd64
MetalLB (load balancer)	0.7.3
Microsoft SQL Server	2019 CTP 3.0

Appendix B: Container resource configuration

This appendix provides supplementary information about configuring container resources. As previously noted, by default, Docker container access to the host's CPU cycles is unlimited. Access to the host memory, except for any limitations that are imposed by the host's kernel scheduler, is also unlimited. This appendix provides information about how to override these default behaviors by limiting a container's CPU resources and allocating memory to a container.

CPUs

Of the many ways to limit CPU resources per container, one of the easiest is to specify the `–cpus` flag when starting the container. The flag communicates the CPU resource configuration to the Completely Fair Scheduler (CFS), which is the Linux kernel CPU scheduler for normal Linux processes. The `–cpus` flag specifies the maximum available CPU resources that the container can use. In the following example, `–cpus="2"` limits the container to a maximum of two cores on the server:

```
$ docker run –cpus="2" --name sqlservername -e "ACCEPT_EULA" -e
"MSSQL_SA_PASSWORD=password" -v volumename:/var/opt/mssql-data-dir
-p 1433:1433 -d localhost:5000/sql2019
```

For processor-intensive applications such as databases, CPU settings assist with managing SLAs and enable greater consolidation.

Memory

You can allocate memory to a container in either of two ways. The first is the `–m` or `––memory` flag that specifies the maximum amount of memory that the container can use. For example, the minimum amount of required memory for the SQL Server image is 2 GB of RAM. Because databases benefit from more memory, we used 16 GB, as shown in the following command:

```
$ docker run –cpus="2" ––memory=16g --name sqlservername -e
"ACCEPT_EULA" -e "MSSQL_SA_PASSWORD=password" -v
volumename:/var/opt/mssql-data-dir -p 1433:1433 -d
localhost:5000/sql2019
```

The second way to configure memory is by using the `–memory-reservation` flag. With this flag, you can specify a soft limit that is smaller than the `--memory` configuration, as shown in the following command. If Docker detects memory contention on the server, the setting on the `–memory-reservation` flag specifies the minimum amount of memory for the container.

```
$ docker run –cpus="2" ––memory=16g –memory-reservation=4g --name
sqlservername -e "ACCEPT_EULA" -e "MSSQL_SA_PASSWORD=password" -v
volumename:/var/opt/mssql-data-dir -p 1433:1433 -d
localhost:5000/sql2019
```

By using the `--memory` and `–memory-reservation` flags, you can enforce hard memory limits to prevent out-of-memory conditions on the server.