Oracle in Docker Containers Managed by Kubernetes

Software Development Use Cases using Dell EMC Infrastructure

October 2020

H18132.2

White Paper

Abstract

This white paper demonstrates the advantages of using Oracle with Docker containers managed by Kubernetes for an application development and testing environment that is hosted on a Dell EMC platform.

Dell Technologies Solutions

Copyright

Published in the USA July 2020 White Paper H18132.1.

Contents

Chapter 1 Executive Summary

This chapter presents the following topics:

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 long-standing issue persists—how to ensure code and data consistency between the RDBMS and the application during the software development and testing life cycle.

In the past, integration between containerized applications and database services like Oracle Database Server was challenging. Software developers often had to wait for Oracle DBAs, who were busy troubleshooting in the production database systems, to create production copies. This delay caused an interruption in the Agile development process.

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.

Solution overview

This solution shows how the use of Oracle Database in containers, Kubernetes, and the Container Storage Interface (CSI) Driver for the Dell EMC PowerFlex family (previously known as Dell EMC VxFlex family) transforms the development process. Using orchestration and automation, developers can self-provision an Oracle database, thereby increasing flexibility and productivity while saving substantial time in creating a production copy for development and testing environments.

We are focusing on the software development and testing use cases because many analysts agree that this market represents the most immediate opportunity to solve significant business challenges using Oracle databases on Docker containers. The current method for developing Oracle-powered applications consists of various platforms and tools. The process is overly complex and prone to creating schedule delays and cost overruns. Any path 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 an Oracle Database ecosystem. We provide two use cases that highlight the development and testing benefits that Oracle databases running on Docker containers enable. Also, we explore the intersection of Oracle databases, Docker containers, the Kubernetes implementation of the CSI specification, and Dell Technologies products and services. Using the CSI Driver for Dell EMC PowerFlex enables comprehensive automation and orchestration from Kubernetes through PowerFlex storage. Using the CSI Driver, customers can automate

storage provisioning of PowerFlex using Kubernetes to gain management efficiencies. The use cases that we present are designed to show how developers and others can easily use Oracle on Docker containers with the PowerFlex family of storage products.

Audience

This white paper is for IT professionals who are interested in learning about the benefits of implementing Oracle in Docker containers in a development and testing environment.

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: Indranil Chakrabarti

Contributors: Anurag A C, Ramamohan Reddy K, Phani MV, Robert Percy, Reed Tucker

Note: For links to additional documentation for this solution, see the Dell Technologies Solutions Info Hub for Oracle.

Chapter 2 Use Case Overview

This chapter presents the following topics:

Introduction to use cases

Our use cases demonstrate the advantages of using Oracle containers for an application development and testing environment that is hosted on a Dell EMC platform. The test environment for both use cases consisted of four Dell EMC PowerEdge R640 servers, which are an integral part of Dell EMC VxFlex Ready Nodes, and a CSI Driver for Dell EMC PowerFlex that were hosted in our Dell EMC 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 CSI Driver for PowerFlex accelerate the applications development life cycle. With this solution, developers can provision Oracle databases in containers without the complexities that are associated with installing the database and provisioning storage.

Containers are a lightweight, stand-alone, executable package of software that includes everything that is needed to run an application: code, runtime, system tools, system libraries, and settings. A container isolates software from its environment and ensures that it works uniformly despite any differences between development and staging. Containers share the machine's operating system kernel and do not require an operating system per application, driving higher server efficiencies and reducing server and licensing costs.

The following figure outlines some typical use cases of Docker containers:

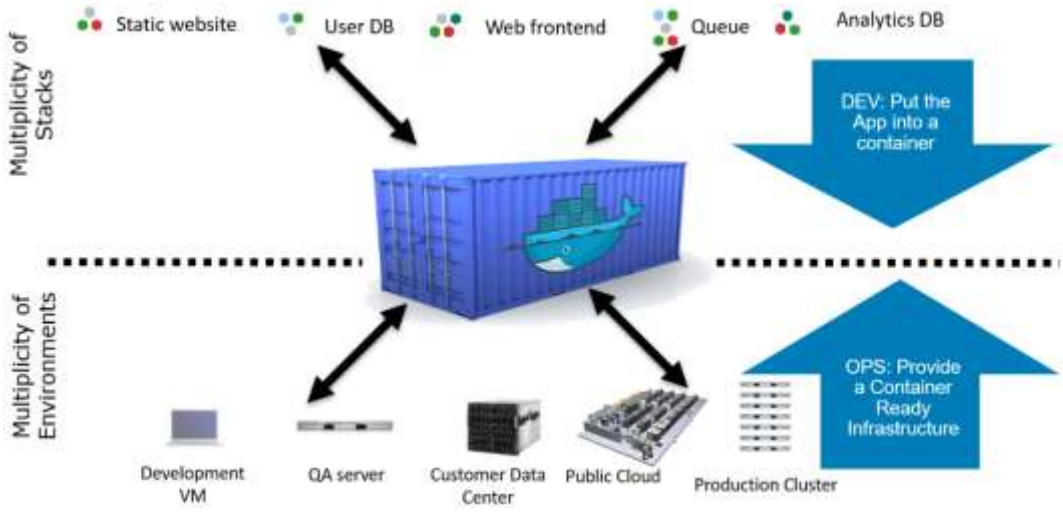

Figure 1. Docker containers – use cases

Recommended platforms and components

The following table lists the components that are required to build a virtualized container infrastructure for the two use cases that are described in this solution:

Table 1. Component specifications of VxFlex Ready Nodes

Component	Details	
Compute hosts	4 Dell EMC PowerEdge R640 servers	
	Processor	2 Intel Xeon Platinum 8268 CPU @ 2.70 GHz 24/48 Cores - 96 Logical Processors (HT)
	Memory	DRAM: 384 GB
	Storage	6 1788.5 GB SSDs
	Network	Server1: 10.230.79.120 Server2: 10.230.79.122 Server3: 10.230.79.124 Server4: 10.230.79.126
	VMs (guests)	Server1: PowerFlex, an SDS Server2: PowerFlex, an SDS Server3: PowerFlex, an SDS Server4: PowerFlex, an SDS
	Hypervisor ESXi 6.7	

The following table lists the software components:

Table 2. Software components

Name	Version/product
HCI	PowerFlex 3.0.x
VMware vCenter	6.7
Operating system	Oracle Linux 7.6
Docker	19.03.2
Kubernetes	1.14.9
Oracle	Oracle 12c, 19c
CNI Plugin	Flannel/Calico
CSI Plugin	PowerFlex CSI plug-in, version 1.0

The following are PowerFlex recommendations:

- At least four physical servers are required in a protection domain.
- The Meta Data Manager (MDM) and Storage Data Server (SDS) components are installed on a dedicated Storage Virtual Machine (SVM); the Storage Data Client (SDC) is installed directly on the ESXi host.

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

Table 3. Terms and definitions

Term	Description
Container	A software-defined form of virtualization that packages together an application and its dependencies. Docker is a widely used container format and is based on Linux container technology. Because Docker containers are a widely accepted standard, many prebuilt container images are available for deployment on systems that support the Docker format.
Kubernetes cluster	A highly available instance of an open-source container-orchestration system for automating application deployment, scaling, and management Some possible abstractions of a Kubernetes cluster are applications, data plane, control plane, cluster infrastructure, and cluster operations. A Kubernetes cluster consists of a set of machines that are known as nodes.
Kubernetes cluster node	A physical machine or a virtual machine (VM) that runs containerized applications. A Kubernetes cluster can contain a mixture of physical machine and VM nodes. One node of the cluster is designated as the master node, which is used to control the cluster. 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.
Kubernetes pod	One or more containers that are guaranteed to be co-located on a worker node and can share resources. The basic scheduling unit and the minimum deployment unit of Kubernetes is a pod. Kubernetes pods are assigned a unique IP address in the cluster, enabling applications in the pod to use ports without the risk of conflict. The Kubernetes master automatically assigns pods to nodes in the cluster.

For more information about these and other PowerFlex networking elements, see *Dell EMC PowerFlex Networking Best Practices and Design Considerations White Paper*.

The following table lists the VMware components of the use case architecture for Oracle in Docker containers:

Table 4. VMware components of use case architecture for Oracle in Docker containers

VMware component	Version
vCPU/VM	32
Memory/VM	320 GB
Operating system	Oracle Linux 7.6
Docker	19.03.2
Kubernetes	1.14.9

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 an Oracle database in containers and then attach a copy of a sample database schema, using VxFlex Ready Nodes. With the Oracle 12c and 19c databases running in containers, we show how to access the database using an Oracle SQL

Developer web interface 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 CSI Driver for Dell EMC PowerFlex for Kubernetes to achieve a greater level of automation and ease of management for development and testing environments. We move beyond manual provisioning of storage to automated provisioning. Using Kubernetes, our developer controls the provisioning of the Oracle database and containers from a local private registry and the database storage from the Dell EMC PowerFlex storage system. After pulling the Oracle database schema application from the Github site, the developer protects the updated state of the database code and data by using Kubernetes to take a snapshot persistent volume container (PVC) of the database. After a round of destructive testing, the developer then restores the database to the preserved state by using Kubernetes and snapshot PVC. 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 5. Use-case comparison

Action	Use Case 1: Docker only	Use Case 2: Kubernetes and CSI Driver for Dell EMC PowerFlex
Provisioning a container	Manual, using script	Self-service (full automation)
Provisioning an Oracle Schema application from Github.	Storage and operating system administrator tasks	
Removing the container and persistent storage	Manual, using script	

Chapter 3 Supporting Software Technology

This chapter presents the following topics:

Container-based virtualization

Two primary methods for enabling software applications to run on virtual hardware are:

- Using virtual machines (VMs) and a hypervisor
- Using container-based virtualization—also known as operating system virtualization or containerization

The older and more pervasive virtualization method, first developed by Burroughs Corporation in the 1950s, is the use of VMs and a hypervisor. This 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. This abstraction enables sharing of the components by multiple VMs, each running a guest operating system. A more recent development is container-based virtualization, in which 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.

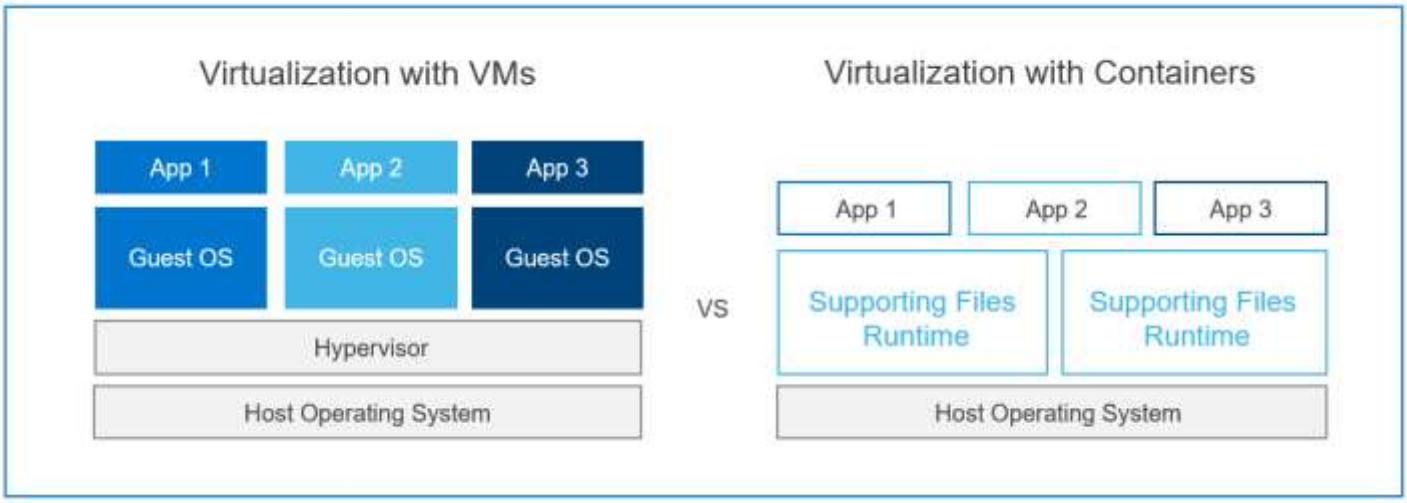

Figure 2. Primary virtualization methods

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 the productivity of IT operations by simplifying application portability. Application developers most often work outside the server environments in which their programs run. 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

The Docker ecosystem consists of the container runtime environment and the tools to define and build application containers. It also includes managing 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 because it is more appropriate for use 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 have licensed and nonlicensed Docker runtimes and must 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 a compressed image from Docker Hub. The compressed image 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, such as high latency during image downloads. Docker Hub does provide the option for users to upload private images to a public registry. 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.

Benefits of virtualization with containers

For data center architects who have standardized on VMware virtualization, a logical question is whether benefits can be gained from hosting containers on virtual machines. Our answer is yes—hosting containers on VMware vSphere VMs increases security and isolation, and it enables the use of multiple host operating systems on one server.

The VxFlex Ready Node infrastructure that we used for this testing hosted two parallel projects. The projects ran on the same software-defined storage but required isolation from each other. In our testing, we employed VMware vSphere VM security to prevent accidental access to resources by anyone outside the respective project teams.

Another key benefit of using VM virtualization for containers is the capability to use multiple host operating systems on the same server. A bare-metal implementation with one host operating system would have forced both projects to use the same stack: operating system, Docker, Kubernetes, and the PowerFlex CSI plug-in. Alternatively, the projects would use separate physical servers to isolate different container software stacks.

Using VMware for our projects gave us the isolation and consolidation benefits of running multiple container stacks within VMs on the same VxFlex Ready Node infrastructure. Performance testing was not part of this project, but testing performance for any production systems, including virtualized container infrastructure, is essential. Virtualization adds another layer to the application stack. Both the container and the VM must be optimized to gain the best performance. For example, the VM configuration (vCPU, vMem, and storage) must be aligned to the performance requirements of the containerized application. For more information, see *Best Practices for Storage Container Provisioning* in the VMware documents web site.

Note that this solution works on bare metal without the VMware virtualization layer/environment that is described in this section.

Kubernetes

Modern applications—primarily microservices that are bundled 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. Google open-sourced the Kubernetes container orchestration system in 2014.

The following figure shows the Kubernetes architecture:

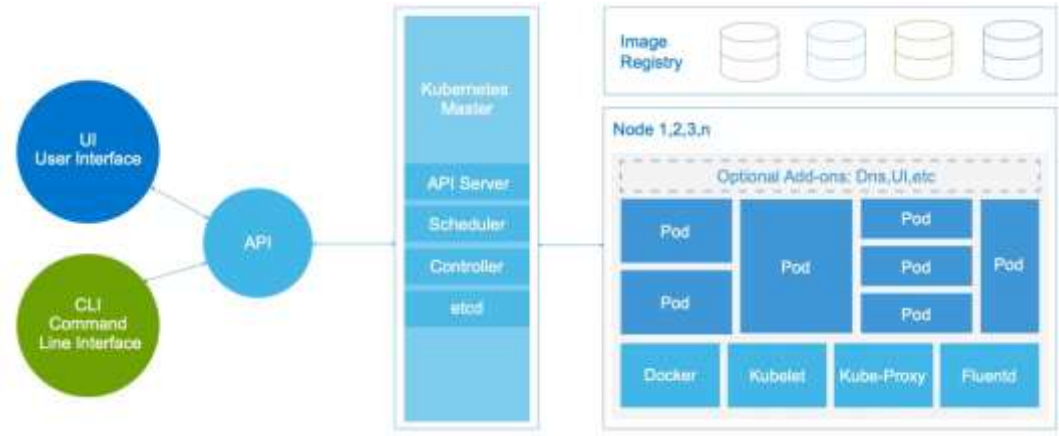

Figure 3. Kubernetes architecture

For additional information about Kubernetes components and concepts, see Kubernetes Components.

Kubernetes features for container orchestration at scale include:

- Autoscaling, replication, and recovery of containers
- Intracontainer communication, such as IP sharing
- A single entity—a pod—for creating and managing multiple containers
- A container resource usage and performance analysis agent, Container Advisor (cAdvisor). cAdvisor provides container users an understanding of the resource usage and performance characteristics of their running containers. It is a daemon

that collects, aggregates, processes, and exports information about running containers.

- Network pluggable architecture
- Load balancing
- Health check service

In a simulated development and testing scenario in Use Case 2, we used the Kubernetes container orchestration system to deploy two Docker containers in a pod.

Kubernetes clusters

A Kubernetes cluster consists of at least one cluster master and multiple worker machines called nodes. These master and node machines run the Kubernetes orchestration system. A cluster is the foundation of the Kubernetes objects including the containerized Oracle database application, all running within a cluster (Figure 12). The Kubernetes cluster has the following components:

- **Load balancer**—One NGINX load balancer runs on a dedicated Red Hat Enterprise Linux VM (along with a Docker registry), with VMware Fault Tolerance (VMware FT) enabled for the VM.
- **Local Docker registry**—One Docker registry container is deployed on the load balancer VM for simplicity. For better control and security, deploy the local Docker registry on its own dedicated VM or VMs that are configured with HA.
- **Kubernetes master node**—One dedicated Oracle Red Hat Enterprise Linux VM provides HA for Kubernetes master nodes if there is a failure. The etcd is deployed on the master node. You can also deploy separate etcd cluster nodes in their own dedicated VM. Figure 12 depicts the architecture.
- **Kubernetes worker nodes**—One dedicated Oracle Linux VM works as workload driver nodes. We put the databases and the customer Data Cluster pods on these VMs. Figure 10 depicts the details.

Kubernetes Container Storage Interface specification

The Kubernetes CSI plug-in implements the Container Storage Interface protocol, which enables containerized applications in Kubernetes clusters to use block storage. To address the challenges of persistent storage, PowerFlex provides its unique CSI plug-in. The CSI plug-in for PowerFlex enables our customers to deliver persistent storage for container-based applications on premises, for both development and production scale.

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 who wanted to add support for their storage system to Kubernetes, or even to fix a problem in an existing volume plug-in, were forced to align with the Kubernetes release process. Also, third-party storage code can 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 CSI Driver for Dell EMC PowerFlex to show the benefits of Kubernetes storage automation.

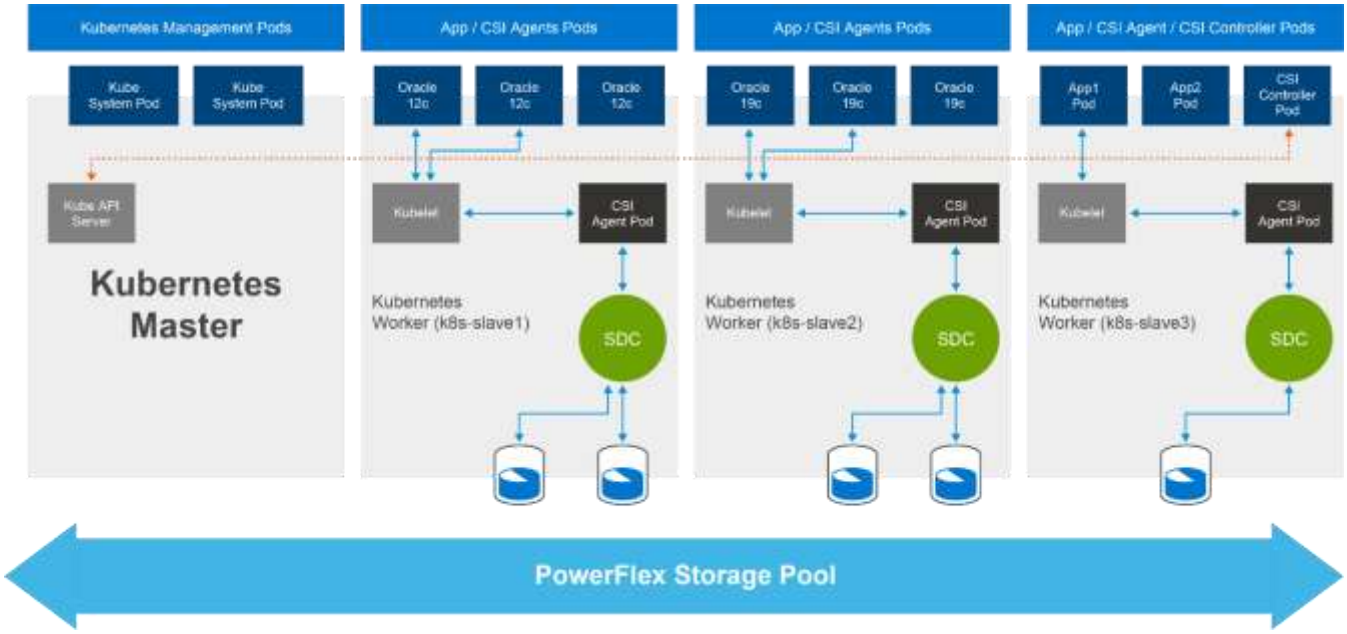

Figure 4. PowerFlex and Kubernetes architecture with the CSI driver

Kubernetes automation with the Dell EMC PowerFlex CSI driver

The PowerFlex CSI driver enables customers to automate storage activities while using Kubernetes. Capabilities include:

- **Persistent volume (PV) actions**—Create, list, delete, and create from a snapshot
- **Dynamic volume provisioning**—Create persistent volumes on demand without any manual steps
- **Snapshot capabilities**—Create, delete, and list

Volume prefixes enable LUN identification. For persistent volumes, the CSI plug-in supports both the ext4 and xfs file systems on worker nodes. The GitHub dell/csi-vxflexos page provides installation details and a download link for the latest PowerFlex CSI driver. You can also download the driver product guide from GitHub.

Note: Dell Technologies also offers CSI drivers on GitHub for the following systems:
• Dell EMC XtremIO
• Dell EMC PowerScale
• Dell EMC PowerMax
• Dell EMC Unity

Kubernetes implementations

Kubernetes is an open-source container orchestration system. Dell Technologies is a platinum member of the Cloud Native Computing Foundation (CNCF), which supports ongoing Kubernetes development. Companies such as VMware, Red Hat, and Canonical have created their own supported Kubernetes versions that are based on the common

open-source version. In the use cases that we describe in this white paper, we used open-source Kubernetes because of its capability to run anywhere, to cover the broadest number of designs. For example, key supported platforms include most versions of Linux and clouds like Google GCP, Amazon AWS, and Microsoft Azure. There is no support cost for open-source Kubernetes, which is supported by the Kubernetes community; however, customers needing enterprise support must explore other versions.

VMware Enterprise PKS, which was jointly developed by VMware and Pivotal, is an enterprise implementation of Kubernetes with deep NSX-T integration and a built-in private registry. The commitment of VMware and Pivotal to support upstream Kubernetes means that customers can get a new version of PKS within weeks of a new Kubernetes release. For customers that have standardized on VMware vSphere, PKS is a natural extension of the platform.

Red Hat OpenShift is a platform for managing containers across on-premises data centers and clouds such as Azure Red Hat OpenShift. Red Hat OpenShift is part of the CNCF Certified Kubernetes program, ensuring compatibility for your container workloads. Ease of installation, a focus on security, and enterprise support make OpenShift a popular choice. The Dell Technologies Solutions Info Hub for the Red Hat OpenShift Container Platform has a library of related technical guides and papers.

Canonical offers a pure upstream Kubernetes platform for managing containers across a wide range of clouds, including all major public clouds, and in private data centers for both bare-metal and virtualized infrastructure. Canonical also offers enterprise support for Kubernetes on Ubuntu for public clouds, VMware, OpenStack, and bare metal.

Kubernetes storage classes

We do not directly use Kubernetes storage classes in either of the use cases that we describe in this white paper; however, the Kubernetes storage classes are closely related to the CSI Driver for Dell EMC PowerFlex plug-in. PowerFlex uses a Container Storage Interface (CSI)-compatible driver with Kubernetes, supporting the broadest set of features for block storage integration. Using storage classes, persistent applications dynamically provision PowerFlex volumes directly for any persistent volume requirements. Kubernetes provides administrators with 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 is agnostic about these class representations. In other management systems, this concept is sometimes referred to as storage profiles. The unique features of PowerFlex make it an excellent complement to Kubernetes for stateful applications. The storage classes objects are required during Dynamic Volume Provisioning, but we are performing Static Volume Provisioning in our use cases that are described in the following sections.

The CSI Driver for Dell EMC PowerFlex creates three storage classes in Kubernetes during installation. The PowerFlex 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.

Using PowerFlex predefined storage classes efficiently scales 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 containerized applications consume and the total amount of storage bandwidth. For scenarios that require a more customized set of storage classes than those that the CSI Driver for Dell EMC PowerFlex creates, you can configure PowerFlex storage system QoS in Kubernetes. In creating a custom QoS policy, you can define maximum bandwidth per GB or, alternatively, maximum IOPS. You can 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

Oracle and Docker containers on Linux

At the DockerCon US event in April 2017, Oracle announced that its Oracle 12c database software application would be available alongside other Oracle products on Docker Store, the standard for dev-ops developers. Dev-ops developers have pulled images from the Docker Store over four billion times, and are increasingly turning to the Docker Store as the canonical source for high-quality curated content. The other benefit to using a containerized Oracle database is that administrators, developers, and customers need not worry about patching and upgrading their Oracle database applications. Docker containers make it easy to deploy applications that are packaged with all their runtime dependencies. The basic aim is to build the capability for building microservices-based container applications without changing code or infrastructure. This approach enables portability between data centers and obviates the need for changes in traditional applications by treating them agilely for faster deployment.

Eventually, deploying these containerized applications at a scale of thousands surpasses human ability. In this task, Kubernetes pods (a group of containers) help in open-source container orchestration. In the future, it is likely that Oracle Docker containers will run the microservices while Kubernetes will be used for container orchestration. Also, the microservices running within Docker containers will communicate with the Oracle databases by using messaging services.

For this white paper, we worked exclusively with Oracle containers for Oracle Linux. We recommend that you check with Dell Technologies to ensure that the latest certified CSI plug-ins are used in your Kubernetes environment.

The following figure shows a prototypical converged Oracle database architecture featuring Docker and Kubernetes:

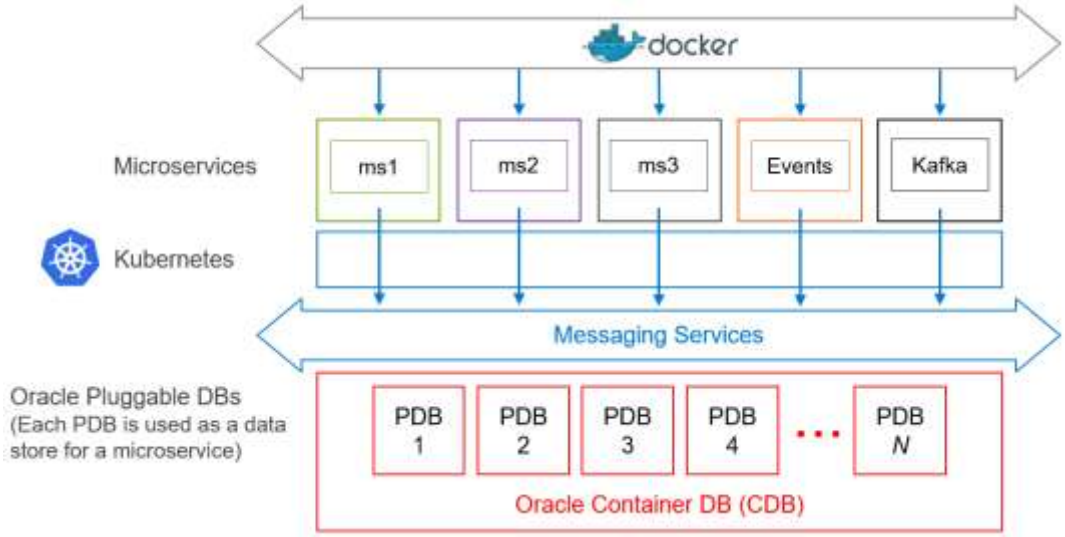

Figure 5. Architecture for Oracle database featuring Docker and Kubernetes

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 Docker engine manages the volumes, which are isolated from direct access by the host, as shown in the following figure:

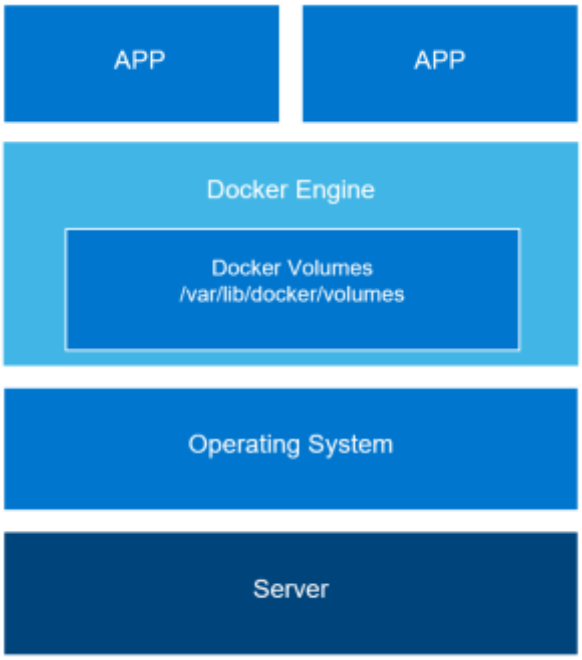

Figure 6. Docker volumes

Advantages of Docker data volumes include:

- Volumes work on both Linux and Windows containers.
- Volumes can be safely shared across containers.
- Any container can prepopulate content into new volumes.

For a full list of benefits, see Volumes in the Docker documentation. For this solution, the Linux administrator used Docker Data Volumes for Use Case 1, which is described in the next section.

Linux bind mounts (in host storage)

In Use Case 1, the Linux administrator can also use Linux bind mounts to connect an Oracle database container to PowerFlex storage that has already been provisioned to the server. The Docker guide indicates that 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:

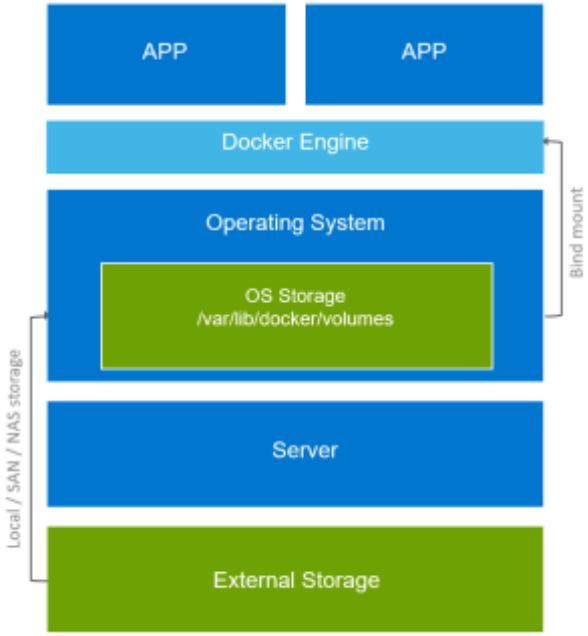

Figure 7. Linux bind mounts

Chapter 4 Dell EMC Flex Nodes and Storage

This chapter presents the following topics:

PowerEdge family servers

Dell EMC PowerEdge family 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 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.

The R640 server is a 1U 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 R640 server 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. Onboard storage can be configured with:

- Front drive bays holding up to 10 x 2.5 in. SAS/SATA SSDs, for a maximum of 76.8 TB
- Up to 10 NVMe drives for a maximum of 64 TB
- Up to 4 x 3.5 in. SAS/SATA drives, for a maximum of 56 TB

For our use cases, we chose the PowerEdge R640 server. Three R640 servers are used as PowerFlex controller nodes and four R640 servers are used as PowerFlex customer nodes. In summary, the PowerEdge R640 is a 1U rack server that supports up to:

- Two Intel Xeon Scalable processors
- 24 DIMM slots supporting up to 1,536 GB of memory
- Two AC or DC power supply units
- 10 + 2 SAS, SATA, or near-line SAS hard drives or SSDs

For details about the PowerEdge server configuration of R640 that the Dell Technologies engineers used for the use cases, see Appendix A Solution Architecture and Component Specifications.

For more details about PowerFlex cluster controller node setup and configuration, see the *Dell EMC PowerFlex: Networking Best Practices and Design Considerations White Paper*.

Dell Technologies hyperconverged infrastructure - PowerFlex Family overview

PowerFlex (previously called VxFlex OS) is the software foundation for the PowerFlex family. It is a scale-out, software-defined, block storage service designed to deliver flexibility, elasticity, and simplicity with predictable high performance and resiliency at scale. The PowerFlex family, which includes the PowerFlex appliance and PowerFlex rack, are fully supported and configured to customer specifications. VxFlex Ready Nodes are validated server building blocks configured for use with PowerFlex. They are available

with thousands of configuration options and are available for customers who prefer to build their own environments.

Customers have several configuration options—from solid-state drives (SSDs) to newer storage technologies such as Non-Volatile Memory Express (NVMe) or Peripheral Component Interconnect Express (PCIe) flash. With these options, customers can create storage tiers that match their capacity and performance requirements. Complementary to storage tiering is the ability to use Quality of Service (QoS) settings. With QoS, customers can define maximum IOPS, maximum IOPS per GB, maximum bandwidth, and maximum bandwidth per GB.

PowerFlex virtualization software supports data compression, which saves valuable storage space on SSDs. Compression is not enabled by default; rather, it must be specified when a volume is created. If a volume does not support compression, then thin provisioning is used by default. Thin provisioning is a technology that reserves storage space by allocating only space that is used, enabling more efficient use of storage.

The PowerFlex rack is an engineered system that provides the ultimate performance, reliability, scalability, agility, and flexibility for modern data center workloads, IaaS, and PaaS cloud infrastructure initiatives. The system is powered by PowerFlex software-defined storage and based on industry-leading enterprise-class PowerEdge servers. It is a rack scale hyperconverged system that comes with a proprietary intelligent physical infrastructure (IPI) cabinet and offers integrated networking and a dedicated system management control plane.

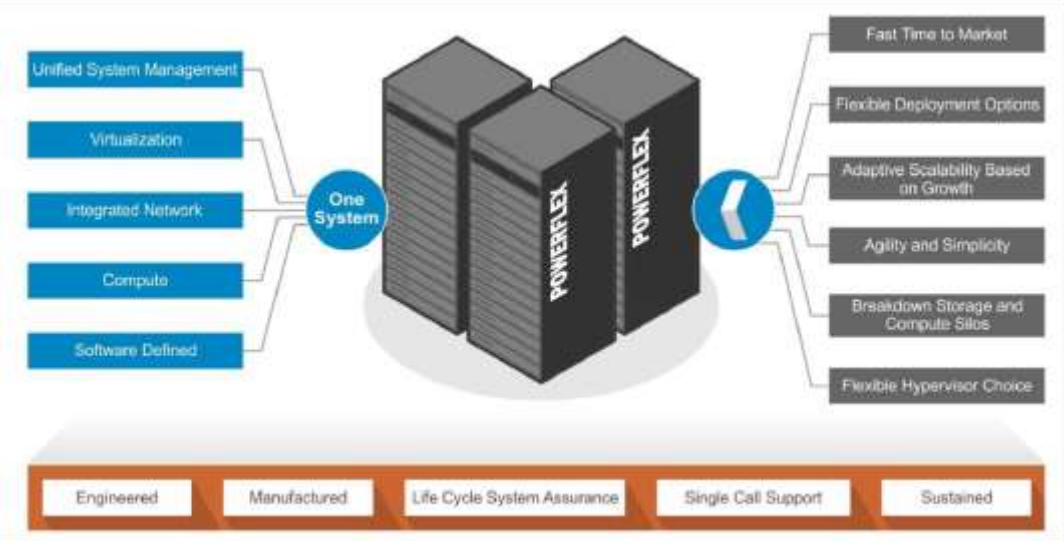

Figure 8. PowerFlex integrated rack benefits

The modular design of the PowerFlex integrated rack enables you to add standardized units of infrastructure to the environment. With this scalable model, you can expand the infrastructure in small increments, to help to eliminate the overprovisioning that is experienced with other approaches. The following figure shows the overall PowerFlex architecture:

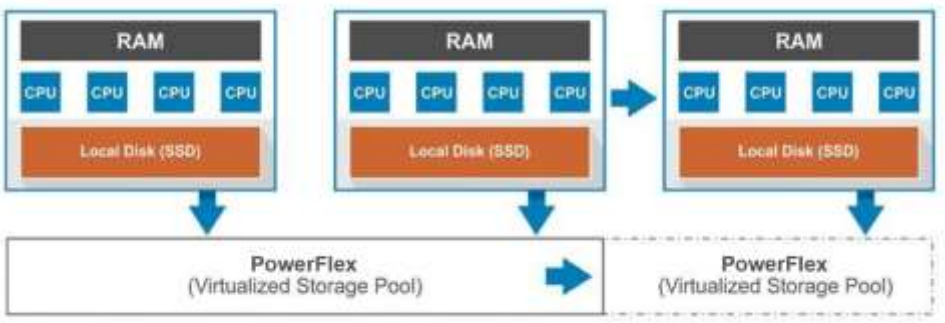

Figure 9. PowerFlex integrated rack scalability

The entire system is built and configured at the Dell Technologies factory according to proven and tested best practices. In addition to the unmatched performance, scalability, and performance, customers benefit from one-call support for all components and end to end life cycle management through a proven automated Release Certification Matrix (RCM) for all components including software and firmware.

PowerFlex Components

PowerFlex consists of three primary components:

- **Meta Data Manager (MDM)—**The MDMs are responsible for managing metadata and core functions such as automated rebuild and rebalance, which ensure data access if media and servers fail. A PowerFlex cluster has multiple MDMs deployed as master, slaves, standby, and tiebreakers to ensure high availability. At any given point, a PowerFlex cluster has one master, one or two slaves, and one or two tiebreaker MDMs. Optionally, the cluster can have up to 10 standby MDMs. Resiliency improves as you add these standby MDMs, and six-nines of availability can be expected with three standby MDMs and two tie-breakers.
- **Storage Data Client (SDC)—**The SDC runs in a server's kernel and acts like a virtual HBA providing highly available connectivity to the storage cluster, consuming the storage as required by the application workload. The SDCs are installed on the same server that is running the application workload, and present PowerFlex volumes to the operating system as if they were local disks.
- **Storage Data Server (SDS)—**The SDSs are daemons that contribute storage to the storage cluster. SDCs communicate directly with the SDSs. When an SDC gets an I/O request from the application, it detects the cached metadata map and sends the request directly to the SDS, which contains the requested data. For read operations, the SDC sees the volume metadata map and sends the request to the SDS using the network that has the data. The volume metadata map has SDS data block mapping for the volume. If the I/O is a write, then the SDC detects the volume metadata map and sends the write request to the SDS that has available block storage. The SDS concurrently writes a secondary copy of the data to another SDS in the protection domain. The other SDS becomes the secondary SDS for that data block. When the secondary copy is written, the primary SDS sends an acknowledgment to the SDC, completing the I/O request.

PowerFlex has an efficient decentralized block I/O flow that is combined with a distributed, sliced volume layout. This design results in a massively parallel I/O system that can scale

up to several hundred nodes. PowerFlex offers multiple deployment options, taking the flexibility of an HCI and an engineered system to the next level.

VMware storage virtualization

In the VMware environment, the PowerFlex SDS is installed as a vSphere Installation Bundle (VIB) in a special VM called Storage VM (SVM). In other words, the MDM and SDS components are installed on a dedicated SVM, whereas the SDC is installed directly on the ESXi host. Storage VMs (SVMs) must have a management IP address and another address for the data network. The data network is where traffic flows between SDSs and SDCs (for read/writes) and between SDSs (for rebuild and rebalance).

The PowerFlex volumes that are defined over the Storage Pools are mapped to the ESXi host and then can be formatted as VMFS datastores, or can be used as RDM devices. When an SDC is mapped to a volume, it immediately gets access to the volume and exposes it locally to the applications as a standard block device. DirectPath Device management is performed using the SVM, yielding the best high availability and performance using two data networks.

PowerFlex GUI

From the PowerFlex GUI, you can perform standard configuration and maintenance activities, as well as monitor the storage system's health and performance. You can use the PowerFlex GUI to retrieve overall PowerFlex performance metrics, and to examine various elements.

From the PowerFlex GUI, select **Backend > Storage**.

The Dashboard displays the following performance metrics:

- Overall system IOPS
- Overall system bandwidth
- Read/Write statistics
- Average I/O size

To retrieve volume-specific performance metrics like Read/Write size, Read/Write IOPS, and Read/Write Bandwidth, select **Frontend > Volumes > Volume Monitor**.

PowerFlex Manager

PowerFlex Manager is a PowerFlex integrated rack management and orchestration (M&O) tool that provides a simple interface for provisioning, managing, monitoring, alerting, life cycle management, and reporting. It increases efficiency by reducing time-consuming manual operations that are otherwise required to implement, provision, and manage operations for your PowerFlex integrated rack. Through automation, you can deploy and manage operations for your PowerFlex integrated rack.

Using PowerFlex Manager, you can:

- Quickly discover and inventory nodes in your PowerFlex integrated rack deployment
- Grow or shrink the PowerFlex integrated rack environment by adding or removing nodes
- Run your PowerFlex integrated rack that is aligned to IT operations management practices

- Monitor, alert, report, and troubleshoot technical issues
- Provide support for the two-layer architecture
- Add or remove volumes within a service
- Store configurations as service templates for easy and consistent deployments

Chapter 5 Manual Provisioning

This chapter presents the following topics:

Use Case 1: Manual provisioning of a containerized development and testing environment

Use Case 1 includes both Oracle 12c and 19c containers running on development and testing environments. As part of this use case, we created four virtual machines:

- Two VMs for running Oracle 12c and 19c containers respectively, each with a 300 GB hard disk, 2 vCPUs, and 20 GB of memory. These VMs are named UC1_12C and UC1_19C.
- The third VM is used for the Oracle SQL Developer client application, which provides the GUI interface between both the 12c and 19c Oracle databases residing inside the containers to the external network.
- The fourth VM is used for storing the Container registry so that Oracle DB images are available locally. The following diagram shows the Use Case 1 architecture:

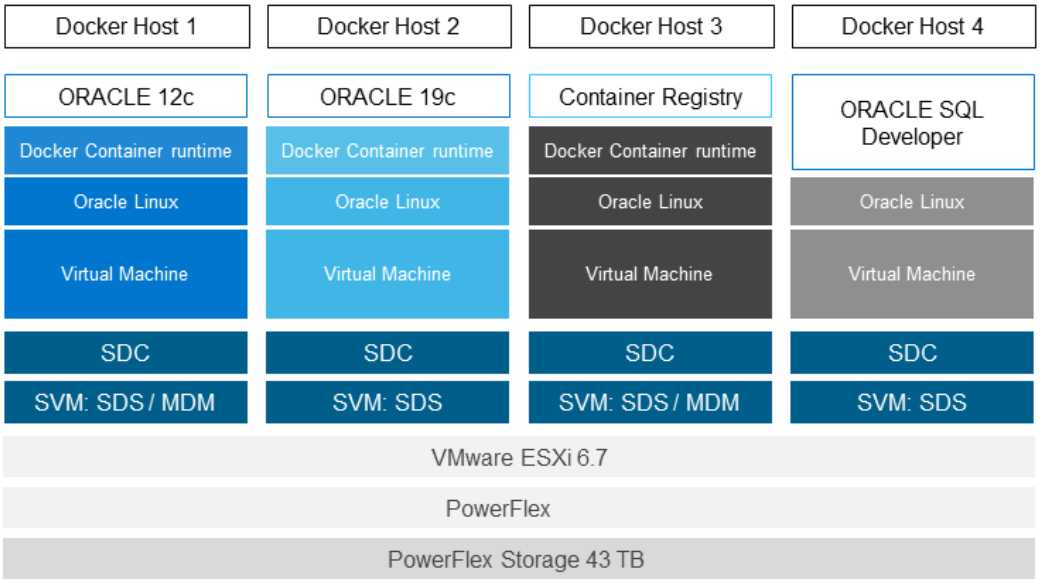

Figure 10. Use Case 1 - Architecture

In Use Case 1, we manually provision the container-based development and testing environment shown above as follows:

1. Install Docker.
2. Activate the Docker EE-License.
3. Run the Oracle 12c database within the Docker container.
4. Build and run the Oracle 19c database in the Docker container.
5. Import the sample Oracle schemas that are pulled from GitHub into the Oracle 12c and 19c database.
6. Install Oracle SQL Developer and query tables from the container.

Step 1: Install Docker

In this step, we install Docker on Oracle Enterprise Linux 7.6 on both VMs (UC1_12C and UC1_19C). Before installing Docker, ensure that:

- Oracle Enterprise Linux 7.6 is already installed on both VMs
- The host operating system has an Internet connection to Yum repositories for Oracle Enterprise Linux

When these prerequisites have been met, perform these steps to install the Community version of Docker:

1. Enable kernel UEKR5 on Oracle Linux 7.6.

```
[root@docker ~] #yum-config-manager --enable ol7_UEKR5
```

2. Enable the add-ons.

```
[root@docker ~] # yum-config-manager --enable *addons
```

3. Update the repository using Yum.

```
[root@docker ~] # yum update
```

4. Install the docker engine.

```
[root@docker ~] #yum install docker-engine
```

5. Enable the docker service.

```
[root@docker ~] #systemctl enable docker
```

6. Start the docker service.

```
[root@docker ~] #systemctl start docker
```

7. Check the docker version.

```
[root@docker ~] #docker version
```

Step 2: Activate the Docker Enterprise Edition license

To activate the Docker Enterprise Edition (EE) license, do the following:

1. Download and copy the Docker license key (`docker_subscription.lic`) to your local Docker host.
2. Activate the license key.

```
[root@12c-docker ~] # sudo docker engine activate –license
docker_subscription.lic
```

3. Verify that the license key has been successfully applied.

```
[root@12c-docker ~] #docker info
```

4. Pull a sample “Hello-World” image to validate Docker.

```
[root@12c-docker ~] #docker pull hello-world
```

5. Run “hello-world”.

```
[root@12c-docker ~] #docker run hello-world
```

Step 3: Run the Oracle 12c container on Docker

To pull the Oracle image from the Oracle registry, run the Oracle 12c container from the Oracle Container Registry (OCR). Follow these steps:

1. From your local host, log in to Oracle Container Registry and provide the credentials.

   ```
   [root@12c-docker ~] # Docker login container-registry.oracle.com
   ```

2. Open a web browser, log in to the OCR, provide your Oracle support ID and password, and select a database.
3. Select the Oracle database 12.2.0.1 Docker image.
4. Pull the 12c Docker image from the OCR.

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.

```
[root@12c-docker ~] # docker pull container-registry.oracle.com/database/enterprise:12.2.0.1
```

5. Display the Docker images.

   ```
   [root@12c-docker ~] #docker images
   ```

6. To save bandwidth, we recommend using the option to set up a private Docker registry for running the containers. Follow these steps to set up a local private registry:

   a. `[root@12c-docker ~] # mkdir -p /opt/registry/data`

   b. `[root@12c-docker ~] # mkdir -p /var/lib/registry`

   c. `[root@12c-docker ~] #docker run -d -p 5000:5000 --name registry -v /opt/registry/data:/var/lib/registry/ --restart always registry`

7. Tag the Oracle 12c image with the local host.

   ```
   [root@12c-docker ~] #docker tag container-registry.oracle.com/database/enterprise:12.2.0.1 localhost:5000/ora12C
   ```

8. Push the Oracle 12c image to the local repository.

   ```
   [root@12c-docker ~] # docker push localhost:5000/ora12c
   ```

Note: If you customize the Oracle 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.

9. Examine the contents of the local repository.

   ```
   [root@12c-docker ~] # ls -ll /opt/registry/data/docker/registry/v2/repositories/
   ```

10. On the host OS, create a Docker volume named “test” and mount it on /home/oracle. Oracle Linux administrators can also use bind mounts to ensure local persistence and avoid data loss. (Add a bind mount to the File Systems table in your server’s File Systems Table (**fstab**)). Though bind mounts are better suited for the Oracle database usage within a container environment, volumes are the preferred mechanism for persisting data generated by and used by Docker containers. In other words, volumes have several advantages over bind mounts; these advantages are described in Docker docs. For Use Case 1, Dell Technologies used Docker volumes. To create Docker volumes, run these commands:

    a. `[root@12c-docker ~] #docker volume create --driver local --opt type=none --opt device=/home/oracle/ --opt o=bind test`

    b. `[root@12c-docker ~] #docker volume create -d local -o Mountpoint=/home/oracle --name=test`

11. Create any necessary groups and users. To implement the security and access mechanism within Oracle database, create groups and valid permissions and add them to the Oracle users as described in the steps below.

    a. `[root@12c-docker oracle] # groupadd -g 54321 oinstall`

    b. `[root@12c-docker oracle] # groupadd -g 54322 dba`

    c. `[root@12c-docker oracle] # useradd -u 54321 -g oinstall -G dba oracle`

12. Because we are hosting database files in the `/home/oracle` directory on the local host, avoid permissions issues from subsequent commands by changing the permissions of this directory:

    a. `[root@12c-docker oracle] # chown -R oracle: dba /home/oracle`

    b. `[root@12c-docker oracle] # chmod -R 777 /home/oracle`

13. Create the Oracle database within the container with the parameters shown below:

```
[root@12c-docker ~] #docker run -d --name database12c -p
1521:1521 -p 5500:5500 -e ORACLE_SID=ORCLCDB -e
ORACLE_PDB=orclpdb1 -e ORACLE_PWD=oracle -v test:/ORCL
localhost:5000/ora12c
```

14. Log into the 12c container. In this step we execute an interactive `bash` shell on the container to run the Oracle 12c database services that are running inside the Docker container. Subsequently we will log in to the Oracle 12c database residing within the container.

```
[root@12c-docker ~] # docker exec -it 3804167e17da
bash[oracle@3804167e17da ~] $ sqlplus "/ as sysdba"
```

Step 4: Build and run the Oracle 19c container on Docker

This step is similar to Step 3 except that the Oracle 19c database image (unlike the Oracle 12c database image) is pulled from the Github repository.

1. Download the build files (images) for Oracle 19c from GitHub.

   ```
   [root@19c-docker ~] # git clone https://github.com/marcelo-ocha/docker-images.git
   ```

2. Download and copy the Oracle 19c database binaries to the location shown below.

   ```
   [root@19c-docker ~] # pwd
   /root/dockerimages/OracleDatabase/SingleInstance/dockerfiles/19.3.0
   ```

3. Build the Docker images.

   ```
   [root@19c-docker ~] #./buildDockerImage.sh -v 19.3.0 -e –I
   ```

4. Push the 19c Docker images to the local private registry.

   a. `[root@19c-docker ~] # docker run -d -p 5000:5000 --name registry -v /opt/registry/data:/var/lib/registry/ --restart always registry`

   b. `[root@19c-docker ~] # docker tag oracle/database:19.3.0-ee localhost:5000/ora19`

   c. `[root@19c-docker ~] # docker push localhost:5000/ora19`

5. On the host operating system, to ensure persistence, create named Docker volumes.

   ```
   [root@19c-docker ~] # docker volume create --driver local --opt type=none --opt device=/home/oracle/ --opt o=bind test
   [root@19c-docker ~] # docker volume create -d local -o Mountpoint=/home/oracle --name=test
   ```

6. Create any necessary groups and users.

   a. `[root@19c-docker oracle] # groupadd -g 54321 oinstall`

   b. `[root@19c-docker oracle] # groupadd -g 54322 dba`

   c. `[root@19c-docker oracle] # useradd -u 54321 -g oinstall -G dba oracle`

7. On the local host, we are hosting database files in the /home/oracle directory. To avoid permissions issues from subsequent commands, change the permissions of this directory.

   a. `[root@19c-docker oracle] # chown -R oracle: dba /home/oracle`

   b. `[root@19c-docker oracle] # chmod -R 777 /home/oracle`

8. Run the container.

   ```
   [root@19c-docker ~] #docker run -d --name database19c -p 1521:1521 -p 5500:5500 -e ORACLE_SID=ORCLCDB -e
   ```

```
ORACLE_PDB=orclpdb1 -e ORACLE_PWD=oracle -v test:/ORCL
localhost:5000/ora12c
```

9. Log in to the 19c container.

```
[root@19c-docker ~] # docker exec -it d9a40ee6ca10 bash
[oracle@d9a40ee6ca10 ~] $ sqlplus "/ as sysdba"
```

Step 5: Import sample schemas from GitHub

In this step we pull the ready-made Oracle 19c schema repository from the Github location and dump it into the local volume. This repository contains a copy of the Oracle Database sample schemas that are installed with Oracle Database Enterprise Edition. From the repository, we import the HR schema into the Oracle 19c database by running the script `@mksample`.

To pull the sample schemas from GitHub, do the following:

1. Import sample schemas.

```
[root@12c-docker ~] # git clone
https://github.com/oracle/db-sample-schemas.git
```

2. Copy these sample schemas to the Docker volume that is accessible to the container.

```
[root@12c-docker oracle_sample_schema] # cp -R db-sample-
schemas/ /home/oracle/
```

3. Modify the folder path inside the scripts to what it is inside the container. For example:

```
[root@12c-docker db-sample-schemas] # perl -p -i.bak -e
's#__SUB__CWD__#'/opt/oracle/oradata/db-sample-schemas'#g'
*.sql */*.sql */*.dat
```

4. Run the script “mksample” in the container to create sample schemas inside the database. Provide the credentials for sys, system users, and all other schemas when prompted.

```
sql>alter session set container=ORCLPDB1;
sql>@mksample
```

Step 6: Install Oracle SQL Developer and query tables from the container

In this step we download Oracle SQL Developer from the Oracle download site and install Oracle SQL Developer into the fourth VM as demonstrated in Figure 10. Oracle SQL Developer is a client software that can be used to connect to any of the Oracle 12c and 19c databases for data manipulation within the Oracle databases. By running different SQL queries from the Oracle SQL Developer (Figure 11) to Oracle databases, we establish that the Oracle databases are operational

To install Oracle SQL Developer, do the following.

1. Download Oracle SQL Developer 19.2.1 Downloads for Linux 64 bit rpm with Oracle jdk 13.0.1 from https://www.oracle.com/tools/downloads/sqldev-v192-downloads.html.
2. Install the Oracle SQL Developer rpm package.

```
[root@12c-docker ~ #rpm -ivh sqldeveloper-
19.2.12.247.2212.noarch.rpm
```

3. Open sqldeveloper.

```
[root@12c-docker ~] #sqldeveloper
```

4. Set up the connection user:hr password service name: ORCLPDB1, as shown in the following figure.
5. You can now query the tables in as shown in the following figure:

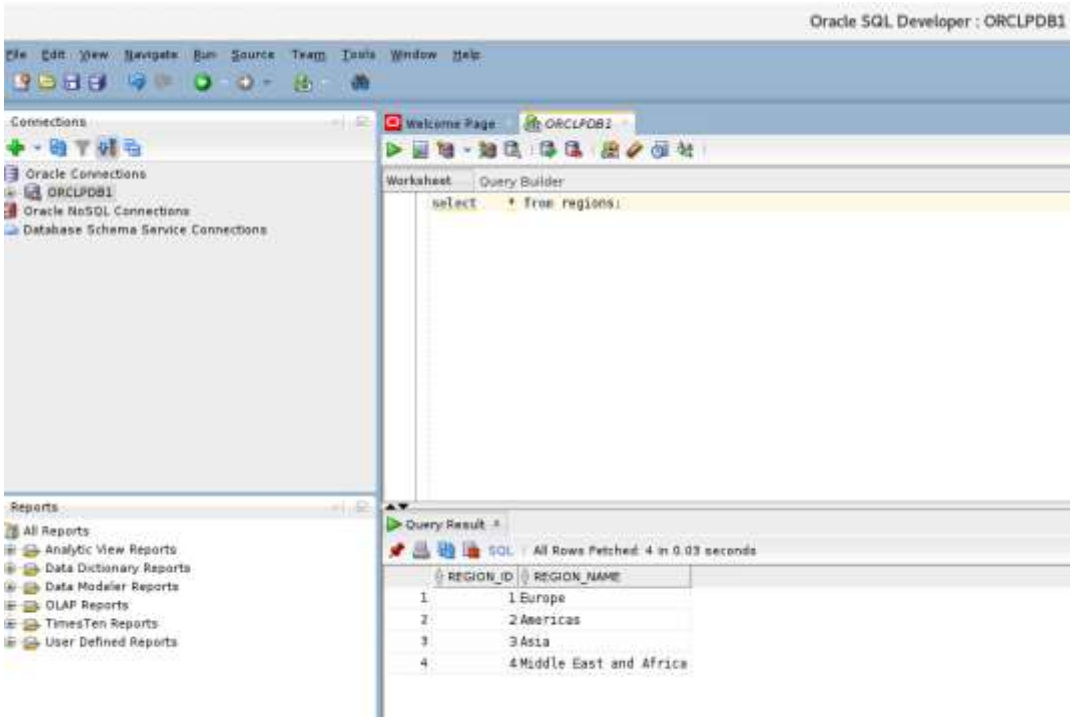

Figure 11. Query the tables

Use Case 1 review

The key benefit in our first use case was the time that we saved by using Docker containers instead of the traditional manual installation and configuration method of building a typical Oracle database environment. The traditional build process is complex and involves much 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 an Oracle database container 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 typical development and testing environment.

Server environment configuration

Dell Technologies offers a broad selection of servers, enabling customers to configure their compute resources to match business requirements. PowerEdge R640 servers 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 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 determine where the registry resides. For example, for low-speed and low-frequency provisioning, a cloud-based registry approach might be ideal. High-speed provisioning that is coupled with high-frequency provisioning can mean that a local private registry using a LAN is best.

Storage provisioning

Provisioning storage from the PowerFlex storage array is fast and easy, but it is a manual process that requires coordination between the developer and the storage administrator. In this use case, we demonstrated manual provisioning of storage, which works 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 CSI Driver for PowerFlex, accelerates storage provisioning.

In Use Case 1, we used Docker volumes, but we could have used bind mounts which would allow Docker to implement in-host storage for fast performance. This method is ideal for attaching storage to a container and can be anywhere in the host operating system. 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 Oracle database Server in Use Case 1.

Table 6. 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

Choice	Decision	Explanation
CSI Driver for PowerFlex product guide or automated provisioning	Manual storage provisioning	Is appropriate for the limited requirements of this use case
Docker volumes or bind mounts	We used Docker volumes in Use Case 1. But customers could use bind mounts for this use case instead.	Provides speed, flexibility and ease of deployment of Oracle database in the Docker containers.

Chapter 6 Automated Provisioning

This chapter presents the following topics:

Use Case 2: Automated provisioning of a containerized development and testing environment

In Use Case 1, we showed the manual provisioning of a container on an Oracle Linux OS, PowerFlex storage system, and VMware vSphere virtualization stack. The next step will effectively lead to automated provisioning of containers and storage via Container orchestration (Kubernetes), further accelerating the provisioning of the software development environment. This is even more important when hundreds or thousands of containers must be managed. Implementing Oracle databases in Docker containers ensures consistent, isolated, and reliable behavior across environments. In this use case, a developer provisions the Oracle database in containers on the existing infrastructure described in Use Case 1 by using Kubernetes with the CSI Driver for Dell EMC PowerFlex.

Use Case 2 demonstrates the value of CSI plug-in integration with Kubernetes and PowerFlex storage. Kubernetes orchestration with PowerFlex provides a container strategy on persistent storage. This strategy demonstrates the ease, simplicity, and speed in scaling out a development and testing environment from production Oracle databases.

Use Case 2 includes both Oracle 12c and 19c database containers running on the development and testing environment within the Kubernetes cluster as shown below.

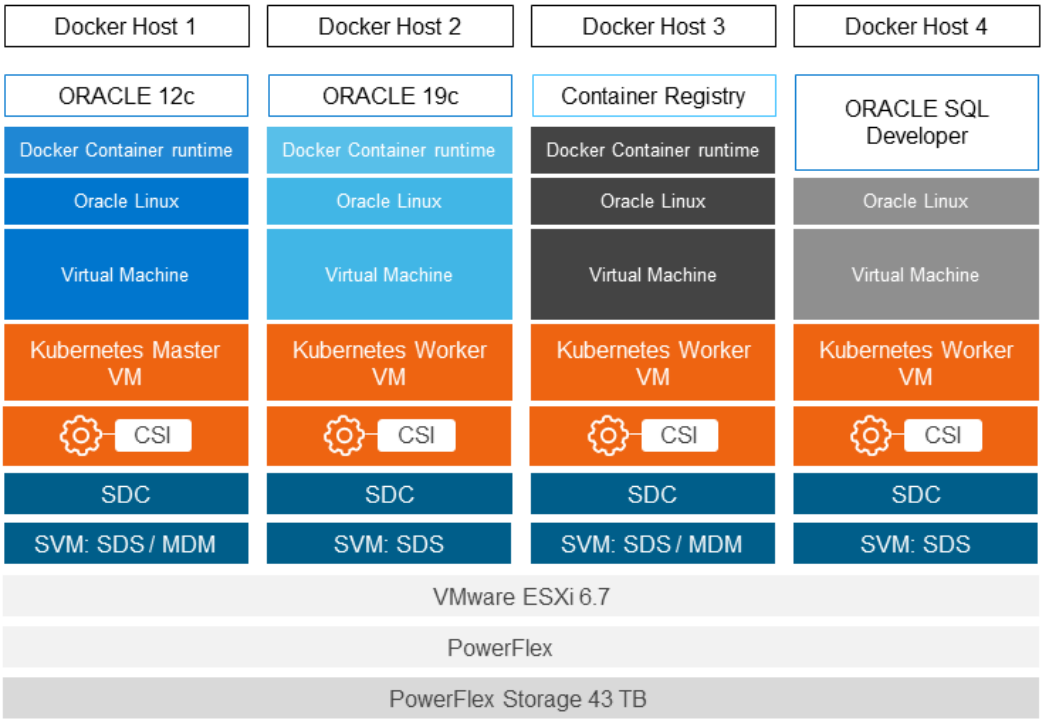

Figure 12. Use Case 2 – Architecture

Use Case 2 architecture description

As shown in the architecture diagram of Use Case 2 (above), one master node and three worker nodes are running within the Kubernetes cluster. The **Kubernetes Master** is a collection of processes that run on a single node in the cluster, which is designated as the

master node. Those processes are: kube-apiserver, kube-controller-manager and kube-scheduler. Each individual worker node in your cluster runs two processes:

- kubelet, which communicates with the Kubernetes Master.
- kube-proxy, a network proxy which reflects Kubernetes networking services on each node.

Kubernetes in an ESXi environment, the PowerFlex SDS, and MDM (explained earlier) are deployed in a special VM called Storage VM (SVM). A Storage VM (SVM) must have a management IP address and another address for the data network where traffic flows between SDSs and SDCs (explained earlier) for read/writes. Inside the Kubernetes cluster, we install the CSI Driver for Dell EMC PowerFlex which is a plug-in that provides persistent storage, using PowerFlex. The driver has two components: CSI controller and CSI agent. While CSI controller facilitates control, coordination, and communication among various worker nodes, the CSI agent sends the status of the target node/server to the CSI controller for management purposes. The distribution of various components of Use Case 2 objects in different Kubernetes worker nodes is depicted in the following table:

Table 7. Distribution of application/Objects across different Kubernetes worker nodes

Kubernetes	Virtual Machine	Application/ Object	POD
Kubernetes Worker 1	VM1	CSI Controller	POD1
		CSI Agent	POD2
	VM2	Oracle Container registry	
Kubernetes Worker 2	VM1	Oracle 12c DB	POD1
		CSI Agent	POD2
Kubernetes Worker 3	VM1	Oracle 19c DB	POD1
		CSI Agent	POD2
	VM2	Oracle SQL Developer	

Use Case 2 runs on VxFlex Ready Nodes which combine PowerEdge servers with PowerFlex software in scalable, reliable, and easy-to-deploy building blocks for hyper-converged or server SAN architecture, heterogeneous virtualized environments, and high performance databases. VxFlex Ready Nodes provide flexibility in deployment options, quick and easy deployments, enterprise-grade resilience, and massive scalability.

Provisioning the environment

In Use Case 2, we manually perform the following tasks to provision a container-based development and testing environment:

1. Setting up the Kubernetes cluster
2. Setting up the Kubernetes dashboard
3. Setting up the Kubernetes load-balancer
4. Configuring the CSI driver for PowerFlex
5. Creating Persistent Volume Claim (PVC) and Oracle Pods on PowerFlex

6. Creating snapshots and restoring persistent volumes
7. Verifying Data persistence and snapshot restore

Step 1: Set up the Kubernetes cluster

For Use Case 2, 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. The Kubernetes administrator performs a custom installation of Kubernetes before performing prerequisite tasks as described below. The Kubernetes cluster will facilitate the automation of the manual tasks of the Docker containers that were already described in Use Case 1.

Setting up the Kubernetes includes the following tasks:

- Fulfilling prerequisites
- Installing Kubernetes
- Initializing the Kubernetes cluster
- Adding worker nodes to the Kubernetes cluster

Prerequisites

Before setting up the Kubernetes cluster, complete the following prerequisite tasks:

- Set SELinux to permissive mode.
- Configure the firewall.
- Ensure that the br_netfilter module is loaded.
- Disable swap for all nodes.

The following sections provide the details for performing these tasks.

Set SELinux to permissive mode

Setting SELinux to permissive mode enables containers to access the host file system, which is required by pod networks.

```
# /usr/sbin/setenforce 0
# sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/'
/etc/selinux/config
```

Configure the firewall

To configure the firewall, select one of the following options:

- If you have a requirement to run a firewall directly on the nodes on which Kubernetes is deployed, be sure to comply with these requirements:
  a. The firewall must support masquerading.

     ```
     # firewall-cmd --add-masquerade --permanent
     ```

  b. All nodes must be able to accept connections from the master node on TCP port 10250.

     ```
     # firewall-cmd --add-port=10250/tcp --permanent
     ```

c. Traffic must be allowed on the UDP port 8472.

```
# firewall-cmd --add-port=8472/udp --permanent
```

d. Ensure that all ports required by Kubernetes are available. For instance, TCP port 6443 must be accessible on the master node to allow other nodes to access the API Server. Run the following command on the master node:

```
# firewall-cmd --add-port=6443/tcp –permanent
```

e. Restart the firewall for these rules to take effect.

All nodes must be able to receive traffic from all other nodes on every port on the network fabric that is used for the Kubernetes pods.

- If you have a requirement ***NOT*** to run a firewall directly on the nodes on which Kubernetes is deployed, enter the following commands:

```
# systemctl disable firewalld
# systemctl stop firewalld
```

Ensure that the br_netfilter module is loaded

Ensure that the br_netfilter module exists and is loaded. This module is usually loaded, and it is unlikely that you would need to load this module manually.

1. Check whether the br_netfilter module is loaded with this command:

```
# lsmod|grep br_netfilter
```

2. (Optional) If necessary, load the br_netfilter module manually by entering these commands:

```
# modprobe br_netfilter

# echo "br_netfilter" > /etc/modules-
load.d/br_netfilter.conf
```

3. Kubernetes requires that packets traversing a network bridge are processed by iptables for filtering and for port forwarding. Ensure that `net.bridge.bridge-nf-call-iptables` is set to 1 in the sysctl configuration file on all nodes.

```
# cat <<EOF >  /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
# /sbin/sysctl -p /etc/sysctl.d/k8s.conf
```

Disable swap for all nodes

Enter these commands to check for performance degradation.

```
# sed -i '/swap/d' /etc/fstab
# swapoff -a
```

Installing Kubernetes

In Use Case 2, we are using one master node and three worker nodes.

To install Kubernetes, follow these steps:

1. Ensure that network configuration is complete on all Kubernetes nodes and that all the nodes are communicating with each other and the Internet. Place the hostname and IP address in the `/etc/hosts` file on all the nodes. All references of IP addresses for the Kubernetes master and worker nodes are stored in this hosts file which is used by different Kubernetes processes.
2. Ensure that Docker Enterprise Edition is installed on all the Kubernetes nodes. To check if Docker service is running, enter the following command:

   ```
   [root@docker ~] # systemctl status docker
   ```

   To check the Docker version, enter this command:

   ```
   [root@docker ~] #docker version
   ```

3. If the Docker Enterprise Edition is not already installed, install it by following the procedure described in Step 2: Activate the Docker Enterprise Edition license.
4. Add the Kubernetes repository, which is basically the creation of the etcd repository that is the primary key-value datastore of Kubernetes cluster state as depicted in Figure 12.

   ```
   # cat <<EOF > /etc/yum.repos.d/kubernetes.repo
   [kubernetes]
   name=Kubernetes
   baseurl=https://packages.cloud.google.com/yum/repos/kubernet
   es-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
   EOF
   ```

5. Install the Kubernetes packages kubeadm, kubelet, and kubectl. If you are installing a specific version of Kubernetes (such as 1.14.9), specify the version now.

   ```
   # yum install kubelet-1.14.9 kubectl-1.14.9 kubeadm-1.14.9
   # systemctl enable kubelet
   # systemctl start kubelet
   ```

Note: All these Kubernetes processes are described in earlier sections and depicted in Figure 12. Kubernetes is now loaded on all nodes and ready to be configured.

Initializing the Kubernetes cluster

These steps help you initialize Kubernetes, set up a cluster, and test your Oracle 12c and 19c applications. The steps in this section verify the operability of the Kubernetes cluster and test the networking communications between the master and worker Kubernetes nodes.

1. To initialize the Kubernetes cluster, run the following command on the master node.

```
# kubeadm init --pod-network-cidr=192.168.0.0/16 --
kubernetes-version=1.14.9  --ignore-preflight-
errors=Swap,FileContent--proc-sys-net-bridge-bridge-nf-call-
iptables,SystemVerification
```

   Where:

   - pod-network-cidr =192.168.0.0/16 is the range of IP addresses for the pod network. (We are using the 'calico' virtual network. If you want to use another pod network such as weave-net or flannel, change the IP address range.) There will be different IP addresses for different pod networks. For example, for flannel the address can be 10.244.0.0/16.
   - kubernetes-version=1.14.9 is the Kubernetes version that you installed on the Kubernetes nodes

2. After the initialization completes, perform the following steps on the master node to copy the node joining command and save it to use in the next section when adding worker nodes to the cluster:

```
# mkdir -p $HOME/.kube
# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

3. Deploy the pod (calico) network to the Kubernetes cluster.

```
# kubectl apply -f
https://docs.projectcalico.org/v3.10/manifests/calico.yaml
```

4. Check the Kubernetes system pods.

```
#kubectl get pods --all-namespaces
```

Adding worker nodes to the Kubernetes cluster

1. Connect to each worker node and run the kubeadm join command that we copied in the previous procedure. The addition of Kubernetes worker nodes to the Kubernetes master completes the cluster creation process.

```
# kubeadm join 10.230.87.241:6443 --token
sntfta.wjsndor3q8zqrpjz --discovery-token-ca-cert-hash
sha256:2e46cf8ffb2838bfee7d419d6bc27b27e0713f98741b84c8cb673
bc34f49e017
```

Note: Synchronize the system time on the master node and worker nodes.

2. Connect to the master node and check the nodes' status.

```
# kubectl get nodes
```

Step 2: Set up the Kubernetes dashboard

Kubernetes dashboard is a web-based Kubernetes user interface that you can use to deploy containerized applications like Oracle database to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster resources. You can use the dashboard to get an overview of applications running on your cluster, as well as to create or modify individual Kubernetes resources (such as deployments, jobs, daemon sets, and so on).

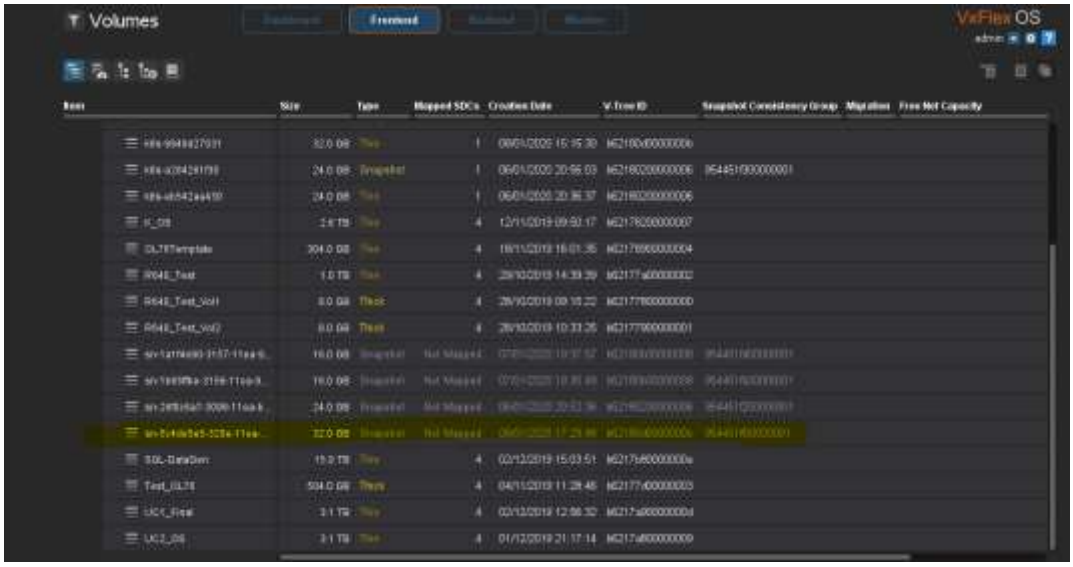

Figure 13. Kubernetes Dashboard

The steps to set up the Kubernetes dashboard are as follows:

1. To create a dashboard for the Kubernetes cluster, connect to the git repository and download the following required yaml files:
   - `influxdb.yaml`
   - `heapster.yaml`
   - `dashbord.yaml`
   - `sa_cluster_admin.yaml`
2. Create the following dashboard configuration files:

```
#kubectl create -f influxdb.yaml
#kubectl create -f heapster.yaml
#kubectl create -f dashbord.yaml
#kubectl create -f sa_cluster_admin.yaml
```

3. Display the token for the service account.

```
#kubectl -n kube-system describe sa dashboard-admin
```

4. Using the service account token displayed in the previous step, get the token for the dashboard login and copy it.

```
#kubectl -n kube-system describe secret <sa token copied
above>
```

5. In a web browser, access the Kubernetes dashboard using this URL format:

   `https://<ip address of master node>:32323`

   For example: `https://10.230.87.241:32323`

6. Choose the token option and specify the same token value that you received from the command line in Step 3 and copied in Step 4.

7. The Kubernetes dashboard displays various navigation options, as shown in the following figure:

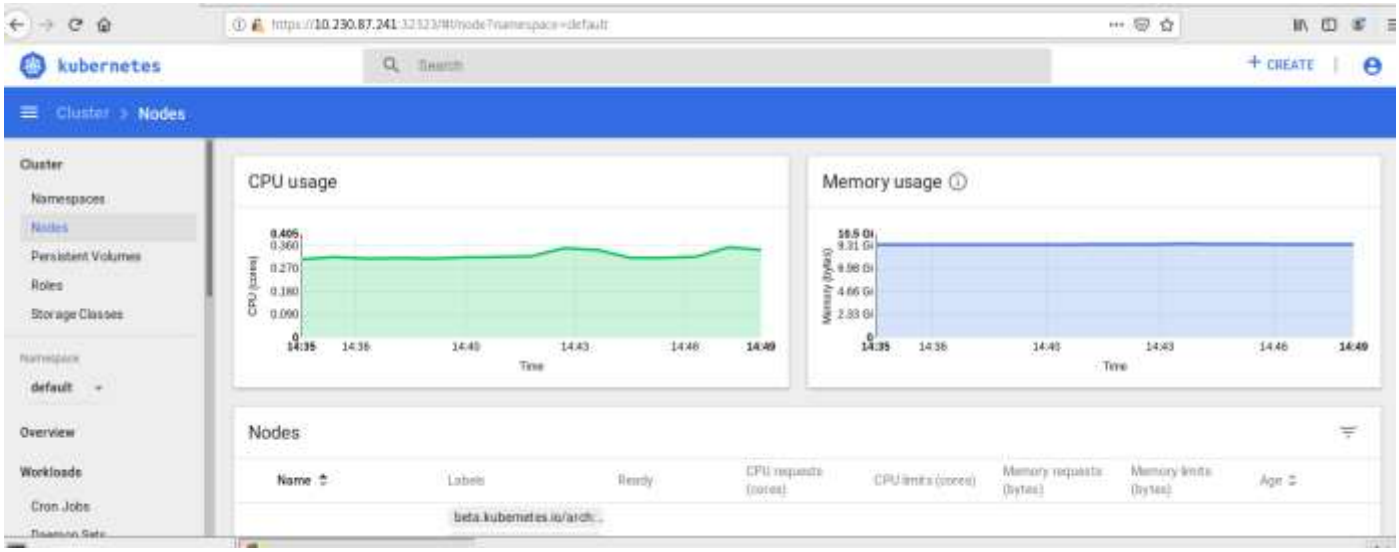

Figure 14. The Kubernetes dashboard

Step 3: Set up the Kubernetes load-balancer

Load balancing in Kubernetes is used to optimally balance the load distribution emanating out of the networking traffic from external sources. Load balancing is accomplished through a feature called kube-proxy, which manages the virtual IPs used by services. To access deployments in the Kubernetes cluster from the external network, we must set up a load-balancer. In this setup, we are using MetalLB as a load-balancer.

1. To install MetalLB, apply the manifest on the master node.

   ```
   #kubectl apply -f
   https://raw.githubusercontent.com/google/metallb/v0.8.3/mani
   fests/metallb.yaml
   ```

   This command deploys MetalLB in the Kubernetes cluster, under the metallb-system namespace.

2. To configure MetalLB, create the config yaml file (`metallb-config.yaml`) by adding the following content to the file:

   ```
   apiVersion: v1
   ```

```
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 10.230.87.180-10.230.87.190
```

3. Run the create command.

```
#kubectl create -f metallb-config.yaml
```

4. After configuring the load-balancer, create a service type called ‘LoadBalancer’ for deployment. This service generates an IP address that helps to connect pods with the external network.

Step 4: Configure the CSI driver for Dell EMC PowerFlex

The CSI Driver for Dell EMC PowerFlex is a plug-in that is installed in Kubernetes to provide persistent storage, using PowerFlex. In addition to our Kubernetes environment, we also need a CSI Driver for Dell EMC PowerFlex to complete the automation process. 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.

Helm charts initiate the installation of the CSI driver. The Helm chart uses a shell script to install the CSI Driver for Dell EMC PowerFlex. This script installs the CSI driver container image and the required Kubernetes sidecar containers.

To configure the CSI Driver for Dell EMC PowerFlex:

- Install Kubernetes (see the previous section for Installing Kubernetes).
- Verify that zero padding is enabled on the PowerFlex storage pools that will be used. Use PowerFlex GUI or the PowerFlex CLI to check this setting.

Then complete the following tasks (defined in detail in the following sections) for configuring the CSI Driver:

- Enable the Kubernetes feature gates
- Configure the Docker service
- Install the Helm package manager
- Install the PowerFlex Storage Data Client (SDC)
- Install the CSI Driver for Dell EMC PowerFlex

Enable the Kubernetes feature gates

Enable the required Kubernetes feature gates before installing the CSI Driver for Dell EMC PowerFlex. In this case, we enable the VolumeSnapshotDataSource feature gate as shown in the steps below:

1. On each Kubernetes master and worker node, to set feature gate settings for the kubelets, edit the `/var/lib/kubelet/config.yaml` file by adding the following line at the end:

   ```
   VolumeSnapshotDataSource: true
   ```

2. On the master node, set the feature gate settings of the `kube-apiserver.yaml` file:

   ```
   vi /etc/kubernetes/manifests/kube-apiserver.yaml
   ```

   Append the following line to the `kube-apiserver.yaml` file

   ```
   - --feature-gates=VolumeSnapshotDataSource=true
   ```

3. On the master node, set the feature gate settings of the `kube-controller-manager.yaml` file as follows:

   ```
   vi /etc/kubernetes/manifests/kube-controller-manager.yaml
   ```

   Append following line to the `kube-controller-manager.yaml` file:

   ```
   - --feature-gates=VolumeSnapshotDataSource=true
   ```

4. On the master node, set the feature gate settings of the `kube-scheduler.yaml` file as follows:

   ```
   vi /etc/kubernetes/manifests/kube-scheduler.yaml
   ```

   Append the following line to the `kube-scheduler.yaml` file:

   ```
   - --feature-gates=VolumeSnapshotDataSource=true
   ```

5. On each node, edit the variable KUBELET_KUBECONFIG_ARGS of `/usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf` file as follows:

   ```
   Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-
   kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --
   kubeconfig=/etc/kubernetes/kubelet.conf --feature-
   gates=VolumeSnapshotDataSource=true"
   ```

6. Use the following commands to restart the kubelet on all nodes:

   ```
   systemctl daemon-reload
   systemctl restart kubelet
   ```

Configure the Docker service

Configure the mount propagation in Docker on all Kubernetes nodes before installing the CSI driver. Mount propagation volumes mounted by a Container to be shared with other Containers in the same Pod, or even to other Pods on the same node.

1. Edit the service section of `/etc/systemd/system/multi-user.target.wants/docker.service` file as follows:

```
Docker.service
[Service]
…
MountFlags=shared
```

2. Restart the Docker service.

```
systemctl daemon-reload
systemctl restart docker
```

Install the Helm package manager

Helm is a package manager. A companion server component called Tiller runs on your Kubernetes cluster, listens for commands from Helm, and handles the configuration and deployment of software releases on the cluster. Find details on Helm and Tiller in Getting Started with Helm/Tiller in Kubernetes. Once you have successfully installed the Helm Client and Tiller, you can use Helm to manage the charts described in Step 4: Configure the CSI driver for Dell EMC .

The curl tool fetches a given URL from the command line in order to save a web file to the local client, or pipe it to another program. Use curl as shown below to install the Helm and Tiller package managers on the master node:

1. Type this command:

```
curl
https://raw.githubusercontent.com/helm/helm/master/scripts/g
et > get_helm.sh
```

2. Change permissions on the script.

```
chmod 700 get_helm.sh
```

3. Run the script.

```
./get_helm.sh
```

4. Initialize Helm.

```
helm init
```

5. Test the Helm installation.

```
helm version
```

6. Set up a service account for Tiller.

   a. Create a `rbac-config.yaml` file and add the following content to the file:

```
apiVersion: v1
kind: ServiceAccount
metadata:
name: tiller
namespace: kube-system
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
```

```
name: tiller-clusterrolebinding
subjects:
- kind: ServiceAccount
name: tiller
namespace: kube-system
roleRef:
kind: ClusterRole
name: cluster-admin
apiGroup: ""
```

b. Create the service account.

```
kubectl create -f rbac-config.yaml
```

c. Apply the service account to Tiller.

```
# helm init --upgrade --service-account tiller
```

Install the PowerFlex Storage Data Client

To install the PowerFlex Storage Data Client (SDC) on all Kubernetes nodes, follow these steps:

1. Download the PowerFlex SDS from Dell EMC Online support. The filename is `EMC-ScaleIO-sdc-3.0-1000.208.el7.x86_64`.
2. Export the MDM_IP in a comma-separated list. This list contains the IP addresses of the MDMs.

   ```
   export MDM_IP=192.168.152.25,192.168.160.25
   ```

3. On each Kubernetes node, use the following command to install the SDC.

   ```
   #rpm -iv ./EMC-ScaleIO-sdc-3.0-1000.208.el7.x86_64.rpm
   ```

Install the CSI Driver for Dell EMC PowerFlex

The CSI Driver for Dell EMC PowerFlex facilitates Use Case 2 by providing the following features:

- Persistent volume (PV) capabilities - create, list, delete, and create-from-snapshot
- Dynamic and static PV provisioning
- Snapshot capabilities - create, delete, and list
- Supports the following access modes:
  - single-node-writer
  - single-node-reader-only
  - multi-node-reader-only
  - multi-node-single-writer
- Supports HELM charts installer

To install CSI the Driver for Dell EMC PowerFlex, follow these steps:

1. Download the installation source files from github.com/dell/csi-vxflexos.

   ```
   # git clone https://github.com/dell/csi-vxflexos
   ```

2. Namespaces provide a way to divide cluster resources among multiple users. This step creates the vxflexos namespace within the Kubernetes cluster:

```
#kubectl create namespace vxflexos
```

3. Create a Kubernetes secret with PowerFlex username and password. Use the `secret.yaml` file to create the secret with the following values to match the default installation parameters:

```
Name: vxflexos-creds
Namespace: vxflexos
apiVersion: v1
kind: Secret
metadata:
  name: vxflexos-creds
  namespace: vxflexos
type: Opaque
data:
  # set username to the base64 encoded username
  username: YWRtaW4=
  # set password to the base64 encoded password
  password: QFZhbnRhZ2U0
```

4. Collect information from the PowerFlex Storage Data Client (SDC) by running the `get_vxflexos_info.sh` script located in the top-level helm directory. This script displays the PowerFlex system ID and MDM IP addresses.

5. To customize settings for installation, copy the `csi-vxflexos/values.yaml` into a file in the same directory as the install.vxflexos named `myvalues.yaml`.

6. Edit the `myvalues.yaml` file to set the following parameters for your installation.

   a. Set the `systemName` string variable to the PowerFlex system name or system ID. This value was obtained by running the `get_vxflexos_info.sh` script in Step 4 of this procedure.

   ```
   systemName: 31846a6a738a010f
   ```

   b. Set the `restGateway` string variable to the URL of your system's REST API Gateway.

   ```
   restGateway: https://10.230.87.31
   ```

   c. Set the `storagePool` string variable to a default (already existing) storage pool name.

   ```
   storagePool: R640_SP2
   ```

   d. Set the `mdmIP` string variable to a comma-separated list of MDM IP addresses.

   ```
   mdmIP: 192.168.152.25,192.168.160.25
   ```

   e. Set the `volumeNamePrefix` string variable so that volumes that are created by the driver have a default prefix. If one PowerFlex system is servicing several different Kubernetes installations or users, these prefixes help you distinguish them.

```
    volumeNamePrefix: k8s
```

7. Run the `sh install.vxflexos` command to proceed with the installation. When the script finishes running, it displays the status of the pods (by calling `kubectl get pods -n vxflexos`).

Step 5: Create Persistent Volume Claim (PVC) and Oracle Pods on PowerFlex

A Persistent Volume Claim (PVC) is a request for storage, similar to how a pod requests a compute resource. A PVC provides an abstraction layer to underlying storage. For example, an administrator could create static persistent volumes (PVs) that can later be bound to one or more persistent volume claims.

After creating the Kubernetes cluster and installing the CSI driver for PowerFlex, create persistent volumes using yaml files. Next, create pods on these volumes by following these steps:

1. The following are yaml files for creating the Persistent Volume Claim (PVC) for deploying an Oracle 12c pod and the Oracle 12c load balancer service.

   The PVC claim file is as follows:

```
#1-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    volume.beta.kubernetes.io/storage-provisioner: csi-
vxflexos
  finalizers:
  - kubernetes.io/pvc-protection
  name: pvc-1
  labels:
    app: oracledb12c
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
  storageClassName: vxflexos
```

   The Oracle 12c pod deployment file is as follows:

```
#2-deployment-Oracle12c.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orcldb12c
  labels:
    app: oracledb12c
```

```
spec:
  selector:
    matchLabels:
      app: oracledb12c
  template:
    metadata:
      labels:
        app: oracledb12c
    spec:
      containers:
      - image: regsrv.brora.com:5000/ora12c
        name: orcldb
        ports:
        - containerPort: 1521
          name: orcldb
        volumeMounts:
        - name: data
          mountPath: /ORCL
      imagePullSecrets:
        - name: oradocreg
      securityContext:
        runAsNonRoot: true
        runAsUser: 54321
        fsGroup: 54321
      volumes:
        - name: data
          persistentVolumeClaim:
            claimName: pvc-1
```

The Oracle 12c load balancer service file is as follows:

```
#3-service-Oracle12c.yaml
apiVersion: v1
kind: Service
metadata:
 name: oracledb12c
spec:
 type: LoadBalancer
 selector:
   app: oracledb12c
 ports:
   - name: client
     protocol: TCP
     port: 1521
selector:
   app: oracledb12c
```

2. Create yaml files for Persistent Volume Claim, Oracle pod deployment, and load-balancing service.

```
# kubectl create -f 1-pvc.yaml
```

```
# kubectl create -f 2-deployment-Oracle12c.yaml
# kubectl create -f 3-service-Oracle12c.yaml
```

3. Use these commands to check the created persistent volume and pod with Oracle 12c container database:

```
# kubectl get pv
# kubectl get pvc
# kubectl get pods -o wide
# kubectl get deployments
# kubectl get svc
```

Step 6: Create snapshots and restore persistent volume

After creating the PVC in step 5, use these yaml files to create a snapshot and to perform a volume restore after installation of a new pod. The objective behind this step is to create a "point in time database" with the help of PowerFlex snapshot and CSI Driver for Dell EMC PowerFlex. The example yaml file that creates a persistent volume for the backed up snapshot is as follows:

```
#
apiVersion: snapshot.storage.k8s.io/v1alpha1
kind: VolumeSnapshot
metadata:
  name: pvc-1-snap
  namespace: default
spec:
  snapshotClassName: vxflexos-snapclass
  source:
    name: pvc-1
    kind: PersistentVolumeClaim
```

The example yaml file for creating a persistent volume claim for restoring the backed up snapshot is as follows:

```
#
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-1-restore
  namespace: default
spec:
  storageClassName: vxflexos
  dataSource:
    name: pvc-1-snap
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
```

Step 7: Verify data persistency and restore snapshot

In this step we create an Oracle 19c database backup using the PowerFlex snapshot mechanism. We also use snapshot restore to verify data persistence in the newly created Oracle 19c database. The objective is to show that the 19c database in the newly created pod can be easily restored.

Follow the steps in this section to create the yaml files to create, snapshot, and restore a volume, and to deploy Oracle 19c. In this use case, to show persistence and snapshot restoration, we are using the yaml files that are shown in the following table:

Table 8. yaml files

yaml File	Description
`1-initial-19c-deployment.yaml`	Creates persistence volume and Oracle 19c deployment and service.
`2-snap1-pvc19c.yaml`	Creates a snapshot of the PVC that was created in Step 1.
`3-restore1-19c-deployment.yaml`	Creates a PVC from the snapshot that was created in Step 2, then creates Oracle 19c deployment and service.
`4-snap2-pvc19c.yaml`	Creates another snapshot of the PVC that was created in Step 1.
`5-restore2-19c-deployment.yaml`	Creates a PVC from the snapshot that is created in Step 4, then creates Oracle 19c deployment and service.

To verify persistence and snapshot restore, do the following:

1. Create an Oracle 19c pod using the yaml file `1-initial-19c-deployment.yaml` . This command first creates a persistence volume, and then an Oracle 19c deployment and service. It then mounts the new persistence volume to the Oracle 19c container.

   ```
   # kubectl create -f 1-initial-19c-deployment.yaml
   ```

   a. In the Kubernetes dashboard, examine the newly created persistence volume in the PowerFlex GUI and Oracle 19c pod.

   b. Using Oracle SQL Developer, connect to this container database using the external IP address that was generated from the load balancer. (In this use case, the external IP is 10.20.87.183.)

   c. Using Oracle SQL Developer, add test data into the database.

   d. Log into the pod and check the mounted volume from PowerFlex to the Oracle 19c container.

2. Take a first snapshot of the PVC by using the file `2-snap1-pvc19c.yaml`.

   ```
   # kubectl create -f 2-snap1-pvc19c.yaml
   ```

   a. Using the PowerFlex GUI, check the newly created snapshot.

   b. Using Oracle SQL Developer, add one more row to the initial Oracle 19c database.

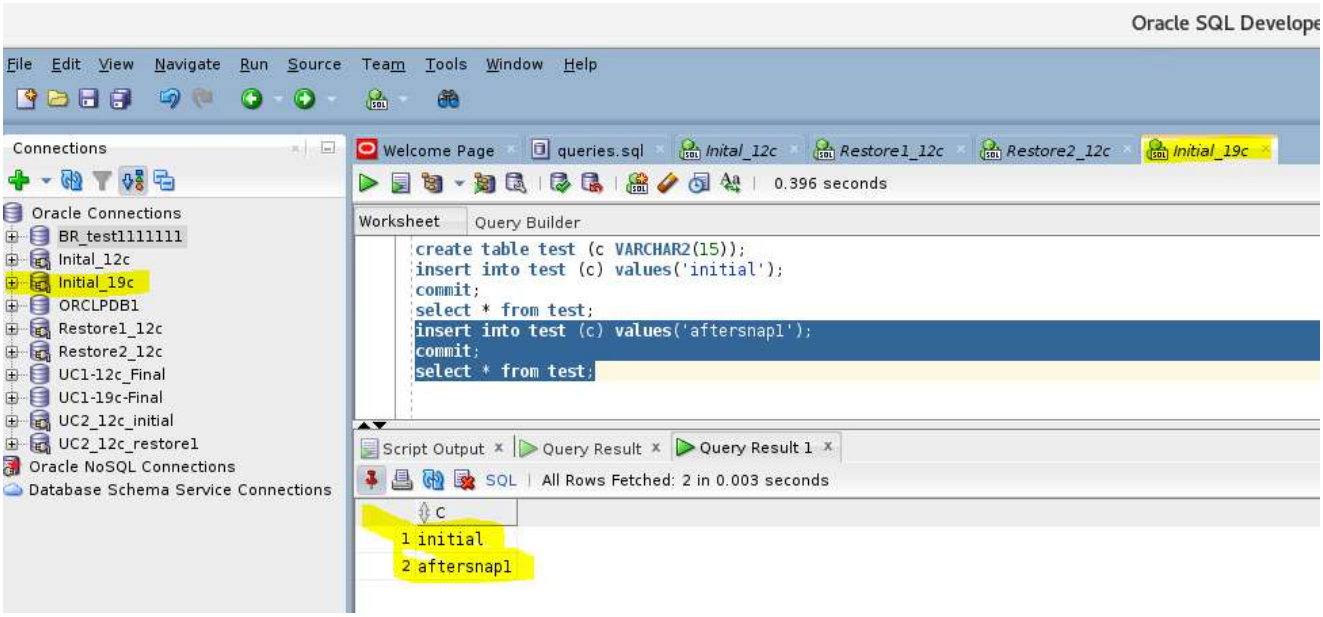

Figure 15. Add a row to the Oracle 19c database

3. Use the snapshot to create an Oracle 19c pod. This step restores the PVC from the snapshot and deploying Oracle 19c database along with its services by mounting the restored volume.

   a. Use the `3-restore1-19c-deployment.yaml` file to restore the Oracle 19c database.

      ```
      # kubectl delete -f 3-restore1-19c-deployment.yaml
      ```

   b. Check for the pod and restored PVC.

      ```
      # kubectl get pods -o wide
      ```

   c. Using Oracle SQL Developer, connect to the restored Oracle 19c pod, using the external IP that we got from the load balancer (10.230.87.184). Check for the initial row that we entered before taking snapshot.

4. Take a second snapshot of the initial PVC, using the file `4-snap2-pvc19c.yaml`.

   ```
   # kubectl create -f 4-snap2-pvc19c.yaml
   ```

   Using Oracle SQL Developer, add one more row to the initial Oracle 19c database. At this point, we have a second snapshot which includes the initial 19c database and all its existing rows plus one extra row.

5. Use the *second* snapshot to create an Oracle 19c pod. This step restores the PVC from the second snapshot and deploys the Oracle 19c database along with the services by mounting the restored volume.

   a. Use the `5-restore2-19c-deployment.yaml` file to restore the Oracle 19c database.

      ```
      # kubectl delete -f 5-restore2-19c-deployment.yaml
      ```

   b. Check for the pod and restored PVC.

```
# kubectl get pods -o wide
```

c. Using Oracle SQL Developer, connect to the second restored Oracle 19c pod, using the external IP that we got from the load balancer (10.230.87.185). Check for all the rows that we entered before taking the second snapshot.

6. Show the persistence of the volume by deleting the pod created by the yaml file `restore2-19c-deployment.yaml` in the previous step.. Once we delete the initial pod, it automatically re-creates itself, and its exact data, using the existing PVC.

   a. Display the pod names.

   ```
   # kubectl get pods -o wide
   ```

   b. Delete the initial pod.

   ```
   # kubectl delete pod pod_name
   ```

   c. After the pod is deleted, verify that another pod that has the same PVC was created.

   ```
   # kubectl get pods -o wide
   ```

7. Using Oracle SQL Developer, compare the data between original and restored environments. In the original pod, the data was comprised of existing data plus one extra row. In the post-delete environment, the data in the Oracle 19c database created inside the new pod is the same as in the original environment pod.

Finally, use the Kubernetes dashboard to display all the pods, deployments, services, and PVCs.

Use Case 2 review

In our second use case, using Kubernetes combined with the CSI Driver for Dell EMC PowerFlex simplified and automated the provisioning and removal of containers and storage. In this use case, we used the yaml files along with the kubectl command to deploy and delete the containers and pods.

This use case demonstrated how we can easily shift away from the complexities of scripting and using the command line to implement 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 necessary 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 Google Kubernetes Engine (GKE) to accomplish the developer's objectives. The CSI plug-in integrates with Kubernetes and exposes the capabilities of PowerFlex, enabling the developer to:

- Take a snapshot of the Oracle database, including the sample schema that was pulled from the GitHub site.
- Protect the work of the existing Oracle database, which was changed before taking the snapshot. We can protect any state. Use the CSI plug-in Driver for Dell EMC PowerFlex to create a snapshot that is installed in Kubernetes to provide persistent storage.
- Restore an Oracle 19c database to its pre-deletion state using a snapshot, even after removing the containers and the attached storage.

The power of this enablement is that steps such as these have traditionally required multiple roles—developers and others working with the storage and database administrators, for example—and more time. Kubernetes with the CSI plug-in enables developers and others to do more in less time and with fewer complexities. The time savings means that coding projects can be completed faster, benefiting both the developers and the business-side employees and customers. 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 Dell EMC options

Apart from using the PVCs to take snapshots, the Dell EMC plug-ins for the PowerFlex rack and the PowerFlex appliance provide a broad range of features in the Kubernetes plug-in space. The PowerFlex rack and PowerFlex appliance create a server-based SAN by using PowerFlex 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 integrated rack and PowerFlex appliance is available from Docker Hub.

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

Chapter 7 Conclusion

This chapter presents the following topic:

Summary statement

Innovation drives transformation. In the case of Docker containers and Kubernetes, the key benefit is a shift to rapid application deployment services. Oracle and many others have embraced containers and provide images of applications, such as for the Oracle 12c database, that can be deployed in days and instantiated in seconds. Installations and other repetitive tasks are replaced with packaged applications that enable the developer to work 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. The Docker containers have also eased the workload of Oracle DBAs. They do not have to divide their attention between performing production database maintenance and copying the production copies to the developers for development and testing work.

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 two-fold: 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 was the addition of the Kubernetes orchestration system and the CSI Driver for Dell EMC PowerFlex. Kubernetes brings a rich user interface that simplifies provisioning containers and persistent storage. In our testing, we found that Kubernetes plus the CSI Driver for Dell EMC PowerFlex enabled developers to provision containerized applications with persistent storage. This solution features point-and-click simplicity and frees valuable time so that the storage administrator can focus on business-critical tasks.

Kubernetes, enhanced with the CSI Driver for Dell EMC PowerFlex, provides the capability to attach and manage PowerFlex storage volumes to containerized applications. Our developer worked with a familiar Kubernetes interface to modify a copy of Oracle database schema gathered from the Github repository database and connect it to the Oracle database container. After modifying the database, the developer protected all progress by using the snapshot feature of PowerFlex and creating a point-in-time copy of the database.

Moving to a Docker plus Kubernetes infrastructure provides a faster and more consistent way to package and deploy an Oracle database. In 2017, Oracle developers made the Oracle 12c database available in the Docker Store. The open-source community that supports Docker and Kubernetes has done much of the foundational work. Manual or scripted installation procedures are not necessary, leaving only customization to the business. Dell EMC adds storage value through PowerFlex for enterprises and the CSI Driver for Dell EMC PowerFlex, streamlining the delivery of applications. The goal of this white paper is to jump-start your application development transformation to enable you to achieve these benefits. Dell Technologies can show you how and can provide an infrastructure that optimizes your containerized applications so that Oracle 12c and 19c databases can be rapidly deployed, initiated, and used.

Chapter 8 References

This chapter presents the following topics:

Dell Technologies documentation

- Dell.com/support is focused on meeting customer needs with proven services and support.
- Dell EMC Technical Resource Center on DellTechnologies.com provides expertise that helps to ensure customer success on Dell EMC PowerFlex rack platforms.
- *VxFlex Data Sheet*
- *CSI Driver for Dell EMC PowerFlex product guide*
- *Dell EMC PowerFlex Networking Best Practices and Design Considerations White Paper*
- *CSI Driver for Dell EMC PowerFlex Product Guide*
- *VxRack FLEX and VxFlex Ready Nodes for Kubernetes*

Kubernetes documentation

The following Kubernetes documentation provides additional and relevant information:

- Kubernetes Concepts

Docker documentation

The following Docker documentation provides additional and relevant information:

- Docker Hub

Oracle documentation

The following Oracle documentation provides additional and relevant information:

- *Oracle 12c Database Administrator's Guide*
- *Oracle 19c Database Administration*

VMware documentation

The following VMware documentation provides additional and relevant information:

- *Best Practices for Storage Container Provisioning*

Appendix A Solution Architecture and Component Specifications

This appendix presents the following topics:

Architecture diagram

The following figure shows the design of the system architecture:

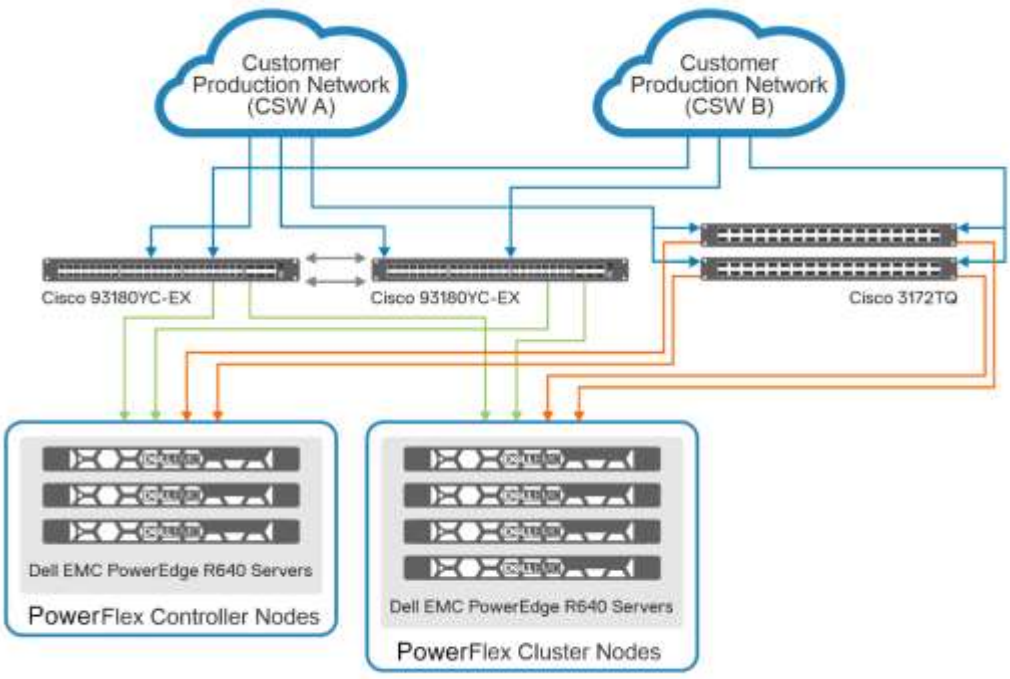

Figure 16. Solution architecture

The PowerFlex family of products enable organizations to consume PowerFlex to best meet their business goals and needs.

PowerFlex rack is a rack-scale engineered system, with integrated networking, that enables customers to achieve the scalability and management requirements of a modern data center. This turnkey system allows businesses to reduce cost and risk by buying versus building an infrastructure platform that can run both traditional and emerging business applications or deploy cloud-native applications.

The PowerFlex rack is a software-defined storage platform designed to deliver flexibility, elasticity, and simplicity with predictable performance and resiliency at scale by combining compute as well as high performance storage resources in a managed unified network. This rack-scale engineered system, with integrated networking, enables customers to achieve the scalability and management requirements of a modern data center.

The PowerFlex rack comes preinstalled with integrated top-of-rack (ToR), aggregation, and out of band (OOB) management switches to provide optimal network traffic flow. The PowerFlex is designed using Cisco Nexus aggregation switches that provide 10/40 GbE IP uplink connectivity to the external network for superior performance. Often, network bottlenecks can limit the scale of an HCI system. The 10/25 GbE ToR switches in the rack eliminate these restrictions and provide a path for future growth.

The PowerFlex appliance is a fully integrated appliance that enables customers to achieve the scalability and management requirements of a modern data center. This turnkey

system allows businesses to reduce cost and complexity with an easy to manage, purpose-built solution. PowerFlex reduces the risk involved in buying versus building an infrastructure platform that can run both traditional and emerging business applications or deploy cloud-native applications.

The VxFlex Ready Nodes support bringing together industry leading PowerEdge servers with PowerFlex in ascalable, reliable, and highly configurable building block. Customers can choose to deploy VxFlex Ready Nodes in a two-layer model, as storage only or in an HCI/single-layer architecture. No matter what configuration they choose, VxFlex Ready Nodes can fit within their existing infrastructure. PowerFlex Manager is not available as part of the VxFlex Ready Node offer.

Server layer

The server layer consists of two separate sets of servers:

- Three PowerEdge R640 servers that are used as PowerFlex controller nodes
- Four PowerEdge R640 servers that are used as PowerFlex customer nodes

The PowerEdge R640 is a 1U rack server that supports up to:

- Two Intel Xeon Processor Scalable Family processors
- 24 DIMM slots supporting up to 1,536 GB of memory
- Two AC or DC power supply units
- 10 + 2 SAS, SATA, Nearline SAS hard drives or SSDs

For more details about PowerFlex cluster controller node setup and configuration, see *Dell EMC PowerFlex Networking Best Practices and Design Considerations White Paper*.

The following tables lists the PowerEdge R640 configuration details. While the upper limits of resources for the R640 server are higher, these configurations are enough for supporting a functional design showing the advantages of containerizing Oracle database instances.

The following table shows the R640 server configuration details for the three PowerFlex controller nodes:

Table 9. PowerEdge R640 server configuration

Component	Details
Chassis	1-CPU configuration
Memory	6 DDR4 Dual Rank 32 GB @ 2,666 MHz
Processors	1xIntel Xeon Gold 6230 CPU @ 2.10 GHz, 16c
rNDC	Intel 2P X710/2P I350 rNDC
Add-on NIC	Intel 10 GbE 2P X710 Adapter and Intel Ethernet 10G 2P X550-t Adapter
Power supplies	2 x Dell 900 W
iDRAC	iDRAC9 Enterprise

Component	Details
Physical disks	5 x 1.7 TB SAS SSD for storage

The following table shows the R640 server configuration details for the four PowerFlex customer nodes:

Table 10. PowerEdge R640 server configuration

Component	Details
Chassis	2-CPU configuration
Memory	12 DDR4 Dual Rank 32 GB @ 2,666 MHz
Processors	2 x Intel Xeon Platinum 8268 CPU @ 2.70 GHz, 24c
rNDC	Intel 2P X710/2P I350 rNDC
Add-on NIC	2 x MLNX 25 GbE 2P ConnectX4LX Adpt
Power supplies	2 x Dell 1,260 W
iDRAC	iDRAC9 Enterprise
Physical disks	2 x 224 GB SATA SSD for Boot disk 5 x 1.7 TB SAS SSD for storage

Network layer

The network layer consists of:

- Two Cisco 93180YC-EX 1/10/25 GbE network switches—Provide data connectivity for the VxFlex cluster.
- Two Cisco 3172TQ 1/10 GbE network switches—Provide OOB management network connectivity for the VxFlex cluster.
- Each cabinet is equipped with redundant access switches (Cisco 93180YC-EX). A pair of aggregation switches is installed in the first cabinet and configured in an access/aggregation network topology. If more than one cabinet exists, the aggregation switches can be spread across or installed in other cabinets.

VMware vSphere Distributed Switches (VDSs) manage virtual networking in the PowerFlex rack. The VDSs contain multiple port groups. Each port group is identified with a network label and associated VLANs. Each node contributes physical adapters or vmnics to VDS uplinks, as shown in the following diagram:

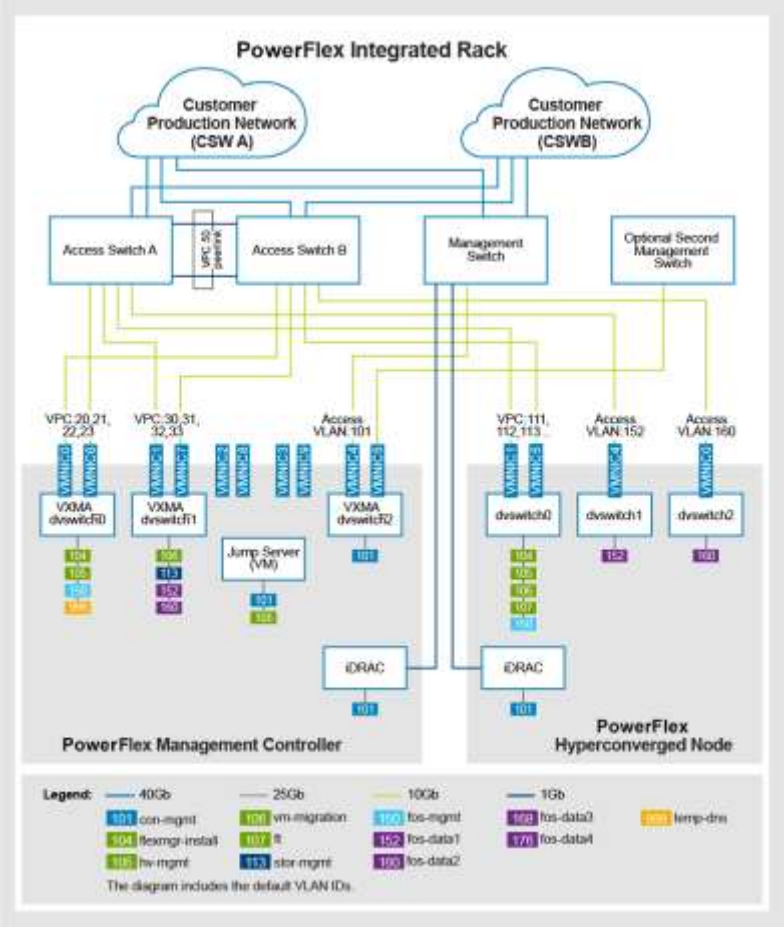

Figure 17. Virtual networking in PowerFlex rack

For the hyperconverged nodes, separate virtual switches are used for each of the PowerFlex rack data networks. The ports that carry data traffic are not aggregated using Virtual Port Channel but are configured as simple access ports. PowerFlex software uses the two networks redundantly. All other communication is through a separate virtual switch that carries all the other VLANs. Customer VLANs (not shown here) are also be added to this virtual switch. The storage-only nodes have a similar arrangement, but do not use virtual switches. Two ports are used for the PowerFlex rack data networks, and a pair of ports are used for the PowerFlex management network. Like the hyperconverged node, the two PowerFlex rack ports are unaggregated access ports, while the two management ports are configured as a bonded interface. The switch connections for these ports are part of a Virtual Port Channel. VMware and customer VLANs are not added to a storage-only node.

CSI Plug-in for Dell EMC PowerFlex

To address the challenges of persistent storage, the Dell EMC CSI plug-in for Dell EMC PowerFlex storage clusters enables containerized applications in Kubernetes clusters to use block storage. The V CSI plug-in implements the Kubernetes CSI specification, and

enables organizations to deliver persistent storage for container-based applications on premises, both at development and production scale.

For additional information, see *CSI Driver for Dell EMC PowerFlex Product Guide*.

Software components

Table 11. Software components and details

Component name	Details
PowerFlex	3.0.1000.208
VMware vCenter and ESXi	6.7 u3
Operating system version	Oracle Linux 7.6 –Mapio, Kernel: 4.14.35-1818.3.3.el7uek
Docker	18.09.6 Enterprise Edition
Kubernetes	1.14.9
Pod network	Calico 0.11.0-amd64
MetalLB (load-balancer)	0.7.3
Oracle Database Enterprise	Oracle 12c, Oracle 19c
Helm	2.16.0
CSI Plugin	PowerFlex CSI Plugin 1.1.3

Appendix B Scaling Up the Database Analytic Workload

This appendix presents the following topic:

Scaling up DB analytic workloads using Intel Optane DC Persistent Memory

Use Intel Optane DC Persistent Memory for scaling up heavy database analytic workloads inside an Oracle database. Further areas of exploration can include scaling up the use cases beyond 500 GB DRAM, especially when using Oracle in-memory database or analytics-heavy workloads. Increasing memory density can enable the entire Oracle workload to be placed in memory, increasing performance significantly. The performance improvement was demonstrated in Oracle Open World 2019. For more information, go to:

- The Database In-Memory on Intel Optane DC Persistent Memory video
- The Delivering Enterprise Value Through Trailblazing Innovations with Oracle: Oracle OpenWorld 2019 video

The configurations shown in the following tables are suggested configurations with Intel Optane DC persistent memory. The Optane DC persistent memory is configured as memory mode and no Oracle software change is required.

Table 12. 1-CPU configuration components and details

Component	Details
Chassis	1-CPU configuration
Memory	6 DDR4 Dual Rank 16 GB @ 2,666 MHz 4 Intel Optane PMem 128 GB @ 2,666 MHz
Processors	1 x Intel Xeon Gold 6230 CPU @ 2.10 GHz, 16c

Table 13. 2-CPU configuration components and details

Component	Details
Chassis	2-CPU configuration
Memory	12 DDR4 Dual Rank 16 GB @ 2,666 MHz 4 Intel Optane PMem 128 GB @ 2,666 MHz
Processors	2 x Intel Xeon Platinum 8268 CPU @ 2.70 GHz, 24c