Microsoft SQL Server 2019 Big Data Cluster on Dell EMC VxRail

June 2020

H18350

White Paper

Abstract

This white paper demonstrates the advantages of using SQL Server 2019 Big Data Cluster technology hosted on a modern Dell EMC VxRail hyperconverged infrastructure to support a scalable data management and analytics platform.

Dell Technologies Solutions

Copyright

Contents

Note: Microsoft SQL Server 2019 Big Data Clusters add-on will be retired. Support for SQL Server 2019 Big Data Clusters will end on January 14, 2025.For more information, see Big data options on the Microsoft SQL Server platform.

Executive summary

Business challenge

For years, data has been generated in many ways, and it continues to grow in various forms. Today, this trend is in a transformative stage. The data generation rate is high, and the type of data being generated in almost every field surpasses the capability of existing data storage techniques.

As reported by Network World, International Data Corporation (IDC) expects that worldwide data will grow to 175 zettabytes within the next 5 years, and nearly 30 percent of that data will be consumed in real time.

Tools and methodologies that solve real-world data challenges

Many tools and techniques are available for managing and analyzing data. However, traditional technologies have limited ability to manage big data and derive useful insights. Thus, to meet those needs, new technologies have been developed.

With SQL Server 2019, you can create a secure, hybrid, machine learning architecture with capabilities ranging from preparing data and training a machine learning model to operationalizing that model and using it for scoring. SQL Server 2019 Big Data Cluster makes it easy to unite high-value relational data with high-volume big data.

The challenge of big data analytics is to explore traditional techniques—such as rule-based systems, pattern mining, decision trees, and other means of data mining—to efficiently develop business rules on large datasets. Big Data Cluster brings together multiple instances of SQL Server with Spark and the Hadoop Distributed File System (HDFS). With this capability, you can more easily unite relational data and big data, and use the data in reports, predictive models, applications, and artificial intelligence (AI).

MongoDB is another prominent, open-source tool that is cross-platform-compatible with many programming languages and supports multiple operating systems. It is an object-oriented NoSQL database that is used for high-volume data storage. The key features of this tool are its support for storing any type of data and partitioning data across multiple nodes and data centers. MongoDB supports cloud-native deployment and offers great flexibility of configuration.

Solution overview

This solution highlights the power of implementing SQL Server 2019 Big Data Cluster technology hosted on Dell EMC VxRail hyperconverged infrastructure (HCI). In the solution use case, we integrate SQL Server 2019 RDBMS, SQL Server Big Data Cluster, MongoDB, and Oracle RDBMS to create a unified data analytics platform. SQL Server 2019 scale-out storage and compute clusters, and new data virtualization techniques are the enabling technologies. This SQL Server 2019 Big Data Cluster solution also benefits from the simplicity of a complete, yet flexible, validated Dell EMC VxRail HCI platform with Kubernetes management and storage integration.

The solution demonstrates the combined value of the following technologies:

- VxRail HCI (all HCI stack components—compute, storage, networking, and data protection)
- Large tables stored on a scaled-out HDFS storage cluster that is hosted by Big Data Cluster
- Smaller, related data tables that are hosted on SQL Server, MongoDB, and Oracle databases
- Distributed queries that are enabled by the PolyBase capability in SQL Server 2019 to process Transact-SQL queries that access external data in SQL Server, Oracle, and MongoDB
- VMware virtualization and management product suites with VMware Cloud Foundation hyperconverged architecture
- VMware vSAN all-flash performance and enterprise-class storage services
- Top-of-Rack (ToR) and out-of-band (OOB) switch integration
- Red Hat Enterprise Linux

With this paper, we include physical and virtual topology diagrams and general connectivity guidelines for the solution. We also provide additional design guidance that focuses on scaling, flexibility, and high availability.

Focusing on the recommended best practices to help make your implementations successful, this white paper describes how to design and build a SQL Server Big Data Cluster solution. In addition, we explore the intersection of SQL Server Big Data Cluster and PolyBase, which supports big data workloads that connect to a larger data sphere. Our solution demonstrates a reliable and repeatable method to deploy the required analytic application services by using automation and orchestration provided by VMware vSphere, Docker, and Kubernetes.

Document purpose

This paper expands on the information that is available from Microsoft and the SQL Server Big Data Cluster ecosystem. It provides insights into each layer of the architecture and describes how the architecture can be applied to a common use case.

The use case in this paper is designed to show how developers and data scientists can easily support a Big Data Cluster ecosystem with VxRail HCI, VMware, Docker, and Kubernetes. The use case also illustrates how using the vSphere CSI driver with VxRail HCI enables comprehensive automation, orchestration, and coordination of server and storage systems for implementing a software-defined infrastructure.

Audience

This white paper is for IT professionals who are interested in learning about the benefits of a data center implementation of SQL Server Big Data Cluster with Docker containers and vSphere automation with the VxRail HCI.

Terminology

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

Table 1. Terminology

Term	Description
Container	A software-defined form of virtualization that combines an application and its dependencies. Docker containers, based on Linux container technology, 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 comprises a set of machines that are known as nodes.
Kubernetes cluster node	A node that runs containerized applications. A Kubernetes cluster can contain a mixture of physical machine and virtual machine (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 co-located on a worker node and can share resources. A pod is the basic scheduling unit and the minimum deployment unit of Kubernetes. Kubernetes pods are assigned a unique IP address within 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.

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 .

Authors: Anil Papisetty, Sam Lucido, Phil Hummel, Sanjeev Ranjan

Contributors: Abhishek Sharma, Karen Johnson

Note: For links to additional documentation for this solution, see the Info Hub for Dell Technologies solutions for Microsoft SQL Server.

Design overview

The goal of this solution is to demonstrate the advantages of using SQL Server 2019 Big Data Cluster with PolyBase to connect data from around the data sphere. The proposed platform design can easily support petabytes of data in the cluster with the additional capability to seamlessly access other data sources to improve analytics reporting scope.

We chose a Dell EMC VxRail platform for our development and testing lab environment. The VxRail HCI incorporates compute, storage, virtualization, and management in one platform offering. The key feature of the VxRail HCI is the integration of vSphere, vSAN, and VxRail HCI System Software for an efficient and reliable deployment and operations experience. For additional automation in our use case, we embedded Kubernetes management into the control plane of vSphere. The overall solution provides unified access to compute, storage, and networking resources. Hosting VMs and containers on an HCI platform using native pods provides end users and administrators with resources that are high-performing, secure, and easy to consume.

For design and component details, see Appendix A: Design architecture and component specifications.

Use case overview

The use case in this white paper shows how the innovative use of container technologies, combined with SQL Server 2019 Big Data Cluster, can simplify management and enable mining of large data volumes with the least amount of operational overhead. The use case demonstrates how you can access your organization's data sphere for improved analytics on an agile platform that is designed for flexibility, automation, and orchestration.

We designed our use case to demonstrate the advantages of using SQL Server 2019 Big Data Cluster for analytic application development. The use case also demonstrates how Docker, Kubernetes, and the vSphere CSI driver accelerate the application development life cycle when they are used with the VxRail HCI platform. The lab environment for development and testing used four VxRail E560F nodes supported by the vSphere CSI driver. With this solution, developers can provision SQL Server Big Data Cluster in containerized environments without the complexities of traditional methods for installing databases and provisioning storage.

We chose Red Hat Enterprise Linux as the operating system for the vSphere VMs. The Red Hat Linux operating system is certified for SQL Server 2019 Big Data Cluster, Docker Enterprise Edition, and Kubernetes. In our use case, we show how to automate the setup of Big Data Cluster using Kubernetes with the vSphere CSI driver on the VxRail system.

Before we tested our use case workloads, we deployed a SQL Server 2019 Big Data Cluster on Kubernetes. We first created a local private registry to manage our Big Data Cluster Docker images. The database team could then update and manage images for use in Kubernetes. For detailed deployment documentation from Microsoft, see How to deploy SQL Server Big Data Clusters on Kubernetes in Microsoft SQL Docs.

To complete the setup after our Big Data Cluster was running, we populated the cluster with data similar to the schema used in a TPC-H benchmark. The official TPC-H benchmark is a decision-support benchmark that requires a certification before publication of results. We chose to replicate only the data generation specification because it defines a schema and relationships that easily scale from 1 TB to 10 TB and larger. We used a 1 TB TPC-H-like dataset in our tests.

We also configured PolyBase to query three external data sources using a join across the schemas. We loaded the largest "fact" table from the TPC-H data into the Big Data Cluster HDFS store. From the same dataset, we loaded smaller "dimension" tables, which were distributed across a SQL Server 2019 database, a MongoDB database instance, and an Oracle 19*c* database instance. We then show how typical TPC-H summary queries can use PolyBase and data virtualization to query three different data sources, all with T-SQL.

This setup allowed us to show how SQL Server 2019 Big Data Cluster technology with PolyBase provides the capability to report on data across different database technologies, using skills that the database team already has. We tested our reporting capabilities in two ways: First, we used our TPC-H queries without any acceleration, and, second, we created a Big Data Cluster data pool and cached the data from the MongoDB database instance and the Oracle database instance. We also loaded a 10 TB version of the data to validate the ingestion of a larger dataset in Big Data Cluster.

Supporting software technology

Container-based virtualization

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

- VMs and a hypervisor
- Container-based virtualization (also referred to as operating system virtualization or containerization)

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

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

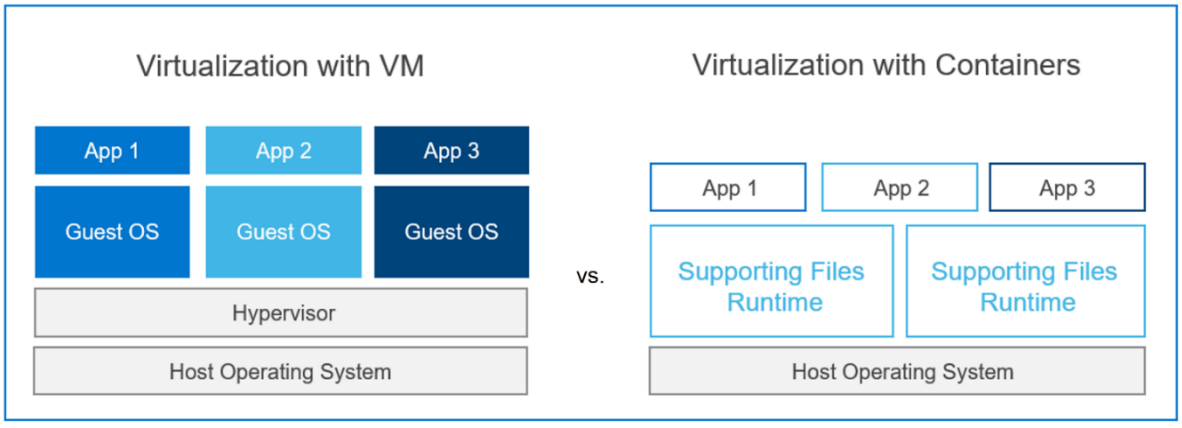

Figure 1. Primary virtualization methods

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

Docker containers

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

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

A Docker registry is supporting technology that is used for storing and delivering Docker images from a central repository. Registries can be public, such as Docker Hub, or private. Docker users install a local registry by downloading from Docker Hub a compressed image that contains all the necessary container components that are specific to the guest operating system and application. A local registry can mitigate many of the challenges that are associated with using a public registry, including high latency during image downloading, depending on Internet connection speed and availability. Although Docker Hub provides the option for users to upload private images, 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. Thus, image provisioning is usually quick. For our validation, we implemented a local private registry to enable fast provisioning without the complexities and cost of hosting in the cloud.

Kubernetes

Modern applications are increasingly being designed to take advantage of container technology—primarily by development of microservices that are packaged with their dependencies and configurations in a container format. Kubernetes, also known as K8s, is a platform for deploying and managing containerized applications at scale. Google released the open-source Kubernetes container-orchestration system in 2014.

The following figure shows the Kubernetes architecture:

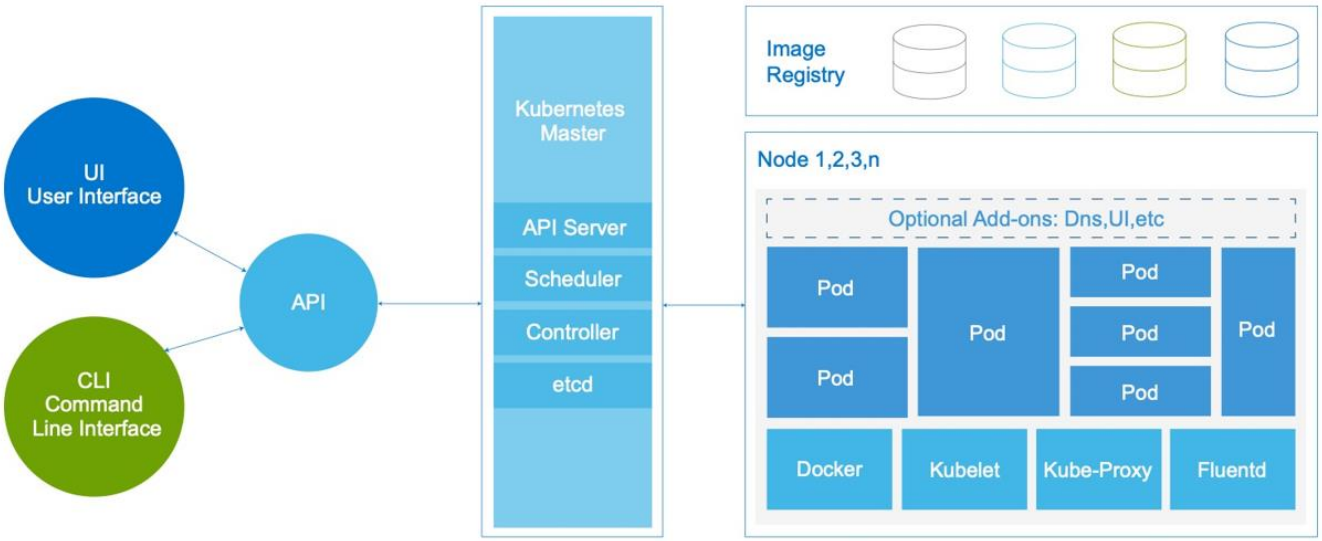

Figure 2. Kubernetes architecture

Kubernetes container-orchestration features for supporting enterprise scale and operations include:

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

Kubernetes Container Storage Interface specification

The Kubernetes CSI specification defines a method for exposing block and file storage systems to containerized workloads through an orchestration layer. Before the development of CSI, Kubernetes used a volume plug-in that was part of the core Kubernetes code and shipped with the compiled Kubernetes binaries. Adding support for a new volume type using the plug-ins approach was challenging when the code was "in-tree." 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. In addition, third-party storage code could cause reliability and security issues in core Kubernetes binaries. The code was often difficult—and 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 having to touch the core Kubernetes code. This capability gives Kubernetes users more storage options and makes the system more secure and reliable. Our solution design highlights these advantages by

using the vSphere CSI driver to show the benefits of Kubernetes storage automation with VxRail.

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. For the use case that we describe in this white paper, we used open-source Kubernetes because of its capability to run anywhere, thus covering 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. The Kubernetes community provides free support for open-source Kubernetes; however, customers requiring enterprise support should explore other versions.

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, focus on security, and enterprise support make OpenShift a popular choice. The Info Hub for Dell Technologies solutions for Red Hat OpenShift Container Platform has a library of related technical guides and papers.

Kubernetes automation with the vSphere CSI driver

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

- **Storage class actions**—Create, list, and delete storage classes (segregation of storage attributes based on vSAN storage policies)
- **Persistent volume actions**—Create, list, delete; create from a snapshot
- **Dynamic volume provisioning**—Create persistent volumes on demand without any manual steps
- **Snapshot actions**—Create, delete, and list

For persistent volumes, the CSI plug-in supports both the ext4 and xfs file systems on worker nodes. For installation details and a download link for the latest vSphere CSI driver, see https://github.com/kubernetes/cloud-provider-vsphere/.

SQL Server on Linux in Docker containers

In recent years, Microsoft has been expanding its portfolio of offerings that are either compatible with or ported to the Linux operating system. With its SQL Server 2017 release, Microsoft delivered SQL Server on Linux and Docker containers. In this white paper, we review the newly released SQL Server 2019 with Big Data Cluster on Docker containers.

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

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

SQL Server Big Data Cluster

SQL Server and similar databases were designed primarily for online transaction processing (OLTP) and are scale-up systems. In a scale-up system, performance benefits come from adding more compute and memory resources in the host server or migrating to a larger server. In contrast, a scale-out database system is designed to use multiple networked servers with storage to distribute data and data processing across a cluster. Scale-out systems are designed to manage big data challenges that do not work well using the traditional scale-up systems approach.

With SQL Server 2019, Microsoft offers an option for hosting scale-out database services. SQL Server 2019 Big Data Cluster is a new scale-out big data solution that combines SQL Server, Spark, and HDFS across a cluster of servers. The SQL Server master instance coordinates connectivity, scale-out query management, metadata, and machine learning. The master instance is a fully functional SQL Server instance and a Tabular Data Stream (TDS) endpoint. It provides application-level protocol that is used by tools such as Azure Data Studio and SQL Server Management Studio.

The master instance contains the scale-out query engine that pushes computations to the compute pool. In Big Data Cluster, the compute pool processes the query across the SQL Server instances. This parallelization enables faster reading of large datasets, thus saving time in returning the results. Management of scale-out queries and other metadata is maintained in the master instance.

Metadata in the master instance includes:

- A metadata database with HDFS-table metadata
- A data plan shard map
- Details of external tables that provide access to the cluster data plane
- PolyBase external data sources and external tables defined in user databases

Beyond the SQL engine and Big Data Cluster management, SQL Server 2019 integrates additional services such as machine learning. Machine learning is the capability to use data in developing models. For example, using machine learning, a model can be created to define the difference between legitimate email and spam. Such a model can then be used to improve email services by removing most spam from a user's inbox.

Machine Learning Services is an add-on feature to the master instance. The feature is installed by default and enables developers to use Java, R, and Python code in SQL Server. This capability provides a pathway for data scientists to use Spark and HDFS tools with the SQL Server master instance, as shown in the following figure:

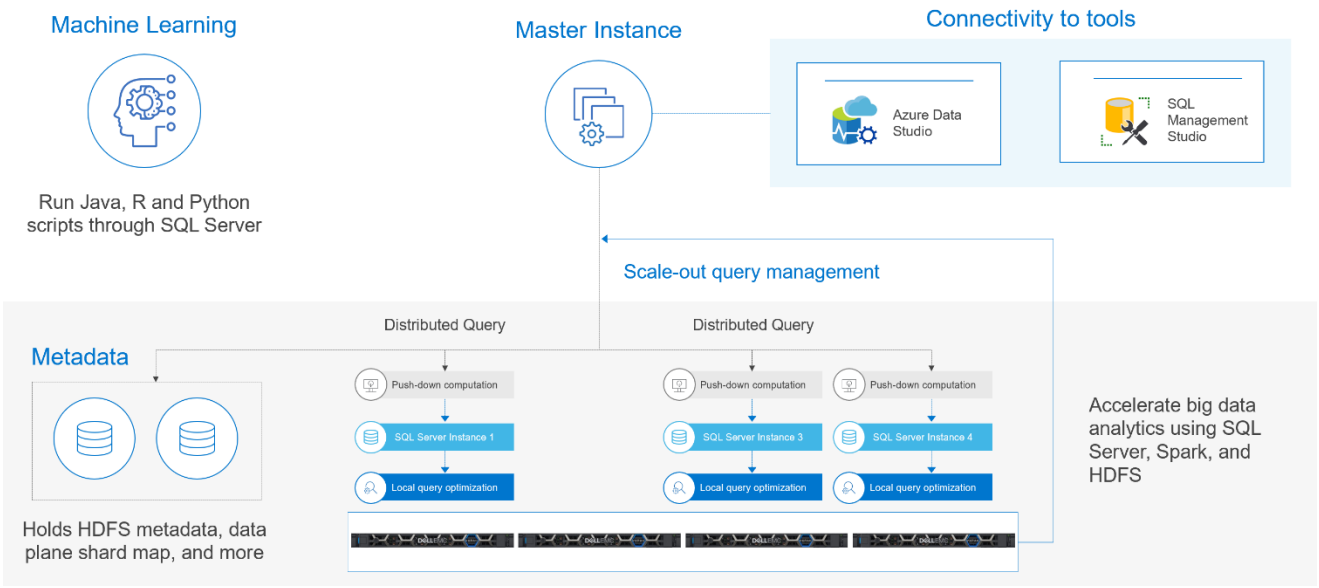

Figure 3. Machine Learning Services installed on SQL Server master instance

Apache Spark services are also available in SQL Server 2019. Spark is a general-purpose cluster computing system that is designed for scale-out data processing performance. Through APIs, Spark supports a broad set of programming languages such as Java, Scala, Python, and R. Spark uses Hadoop's client libraries for accessing HDFS data nodes, meaning Spark is fully compatible with HDFS scale-out storage. Spark provides the ability within Big Data Cluster to perform machine learning (MLib), graph processing (GraphX), Spark Streaming, and a full range of traditional ETL-style workloads. As an option, customers can create a dedicated pool for Spark analytics. A dedicated Spark pool can be beneficial if customers plan to extensively use Spark services.

In addition to the master instance, SQL Server 2019 Big Data Cluster includes several scale-out services. For example, compute pools enable offloading distributed query processing to a dedicated scale-out service. In our lab setup, the Big Data Cluster compute pool has one or more SQL Server compute instances running in containers across each of the VxRail nodes. Other examples of how the compute pool accelerates data processing include:

- Queries with joins combine data from two or more objects in a database or from external data sources. With join queries, you can:
  - Join with HDFS directories containing thousands of files.
  - Join tables in different data resources.
  - Join tables with different partitioning or data distribution schemes.
- Data exports move data out of an application in parallel to HDFS or Azure Blob storage.

The following figure shows the compute pool configuration:

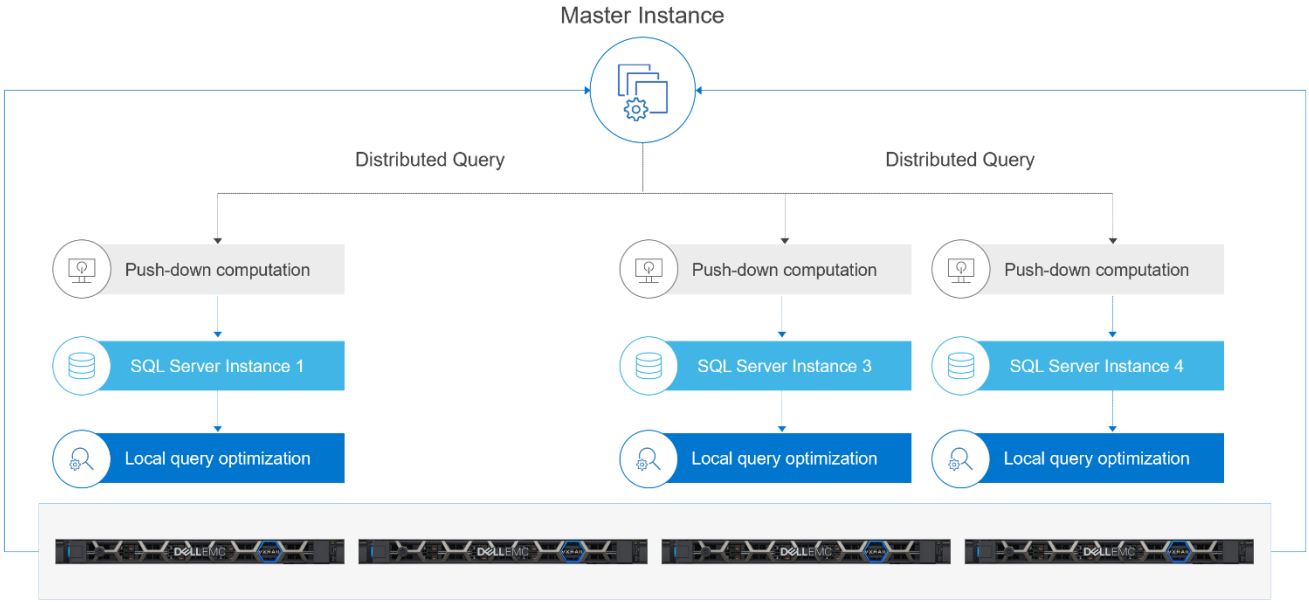

Figure 4. Compute pool configuration

A SQL 2019 Big Data Cluster storage pool is a group of pods hosting the following services: SQL Server engine, HDFS data node, and Spark. An HDFS data node stores and replicates data with other nodes in the storage pool. Because HDFS replicates data in most architectures, traditional disk-based RAID is not needed. The HDFS storage nodes service all read and write operations to the pool.

The SQL Server master instance can use the local SQL engine to access the HDFS data nodes, providing the capability to use T-SQL syntax with Big Data Cluster. Thus, the organization can use existing T-SQL reports and fully benefit from developer experience with T-SQL; developers do not have to learn a new programming language. The following figure shows the storage pool architecture as provisioned by Kubernetes in pods:

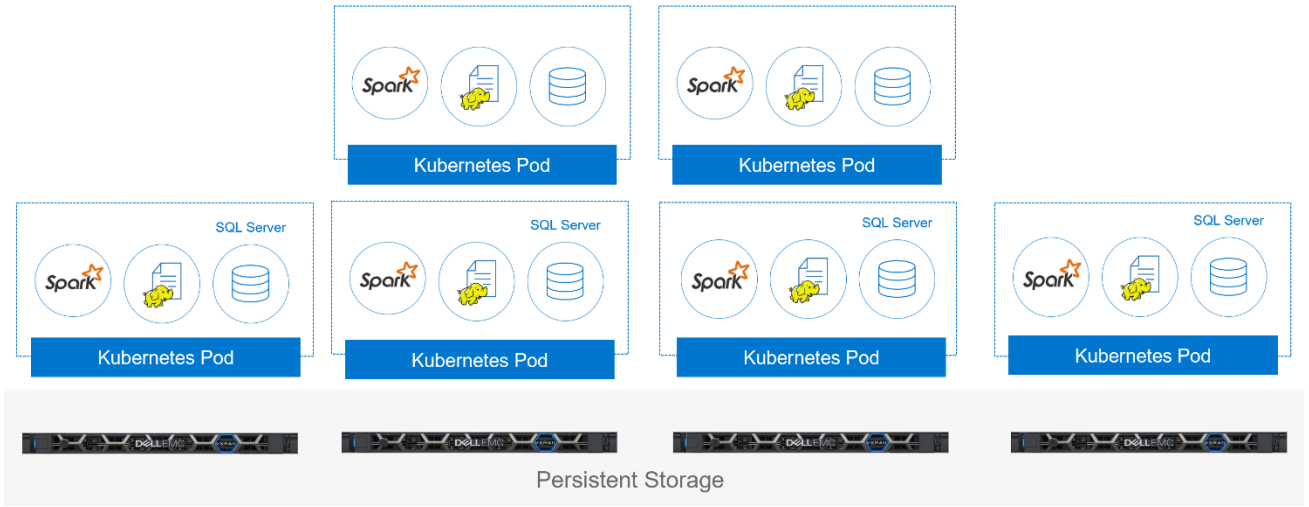

Figure 5. Storage pool architecture

A SQL Server 2019 Big Data Cluster data pool has one or more SQL Server instances. The data pool can be used to ingest data from external sources or cache data from the storage pool. The data pool is connected to the SQL Server master instance for control. Data Definition Language (DDL) is used to manage the data pool and its objects, and Data Manipulation Language (DML) is used for data management. The primary use of the data pool is to offload data processing activities from the SQL Server master instance. The data pool can accelerate data processing because it uses data sharding (partitioning) across the SQL Server instances in the pool. Database shards are data that is distributed in separate containers across multiple database instances residing on separate servers in the data pool. Most of the data is not duplicated or replicated, meaning that each shard contains a single copy of some data. Sharding improves performance by enabling multiple data reads in parallel across multiple database instances, thus reducing the time that is required to complete a query.

PolyBase

SQL Server 2019 uses data virtualization to connect disparate data sources, enabling reporting and analytics without the need for an ETL process to assemble the data in a common data warehouse schema. Microsoft has integrated PolyBase with Big Data Cluster, enabling organizations to unify structured and unstructured data sources. With PolyBase, organizations can access data from Azure SQL Database, Azure SQL Data Warehouse, Oracle, Teradata, MongoDB, Azure Cosmos DB, and HDFS.

A key benefit of PolyBase for developers and data scientists is having one consistent user interface for accessing multiple data sources. T-SQL is used to access external table data, simplifying the creation of applications, reports, and analytics. PolyBase with Big Data Cluster connects multiple datastores into a broad data sphere, enabling a more comprehensive approach to data analysis.

In our Big Data Cluster, we distributed use case tables from the TPC-H benchmark across Oracle, MongoDB, and the SQL Server 2019 Big Data Cluster. The Oracle 19*c* database and MongoDB reside on two separate VMs of the VxRail system. Accessing the Oracle database and MongoDB with PolyBase demonstrates data virtualization and the ability to access data from disparate locations within the data center.

Dell EMC VxRail

Introduction

Dell EMC VxRail HCI offers the performance, capacity, and graphics capabilities that are needed to meet the infrastructure requirements of a small or mid-size enterprise. VxRail HCI provides a simple, cost-effective solution that solves virtualization infrastructure challenges and supports a wide range of applications and workloads.

The VxRail portfolio includes:

- **E Series**—1U, one-node platform with an all-NVMe option and T4 GPUs for a wide range of use cases including AI and machine learning
- **P Series**—Performance-intensive 2U, one-node platform with an all-NVMe option, configurable with one, two, or four sockets, optimized for intensive workloads such as databases
- **V Series**—VDI-optimized 2U, one-node platform with GPU hardware for graphics-intensive desktops and workloads
- **D Series**—Durable, ruggedized, short-depth MIL-STD certified 1U, one-node platform that is designed to withstand extreme conditions such as intense heat and cold, shock, vibration, dust, humidity, and EMI
- **S Series**—Storage-dense 2U, one-node platform for demanding applications such as virtualized Microsoft SharePoint, Microsoft Exchange, big data, analytics, and video surveillance
- **G Series**—Compute-dense 2U, four-node platform for general-purpose workloads

The VxRail design uses a modular, distributed system architecture based on a 1U or 2U platform, with linear scalability per node. Various options are available for compute, memory, and storage configurations to match many use cases. Options include a range of Intel processors, variable RAM, storage, and cache capacity for flexible CPU-to-RAM-to-storage ratios.

VxRail systems are assembled with proven PowerEdge server hardware that has been integrated, tested, and validated as a complete solution by Dell Technologies. The current generation of Dell EMC PowerEdge servers for VxRail HCI uses Intel Xeon E5 processors. The Xeon E5 processor is a multithreaded, multicore CPU that is designed to handle diverse workloads for cloud service, high-performance computing, and networking. Each VxRail model has a different number of cores and memory capacity.

For ease of management, VxRail HCI System Software automates life-cycle management, reduces complexity, and boosts infrastructure security. Administrators can update and patch production environments with no downtime while VxRail Manager intelligently deploys important updates.

VxRail HCI includes vSphere, vSAN, and VxRail HCI System Software to make data center deployment and operations as easy as possible. Because vSphere is the core operational software in VxRail HCI, the system inherits the many vSphere approaches for running containerized and virtualized application workloads.

For more information about VxRail HCI, see the VxRail Info Hub.

VxRail HCI System Software

The VxRail software layers use VMware technology for server virtualization and software-defined storage. VxRail nodes are configured as ESXi hosts. VMs and services communicate using the virtual switches for logical networking.

The following figure provides a high-level overview of the VxRail HCI System Software operating system:

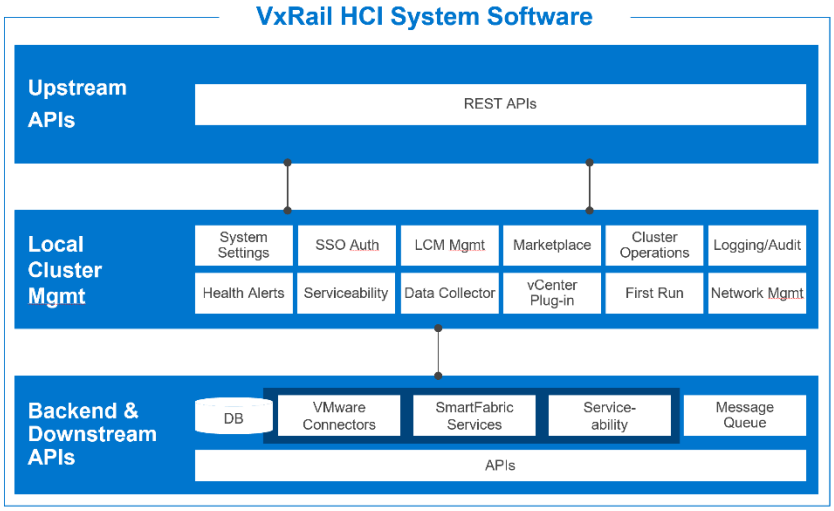

Figure 6. VxRail HCI System Software

The most used and most familiar feature of VxRail HCI System Software is VxRail Manager, which is the primary deployment and element manager for VxRail. VxRail Manager delivers automation, life-cycle management, support, and serviceability capabilities, integrated with vCenter and SDDC Manager.

VxRail Manager functionality is available in vCenter through an HTML5 plug-in. The initial deployment of VxRail clusters uses the native VxRail Manager process. VxRail Manager conducts life-cycle management through a fully integrated and seamless orchestrated process.

VxRail Manager features include:

- Streamlined system deployment, with over 200 automated tasks
- Single-click software updates
- Dashboards for health and event monitoring, and physical views
- Direct access to support, articles, and a community site

VxRail system management

VMware Validated Design is a data center design that spans compute, storage, networking, and management. VMware Validated Design for Software-Defined Data Center (SDDC) is a reference architecture for how to deploy, operate, and maintain a VMware SDDC.

VxRail HCI includes a software stack for system management, virtualization, and VM management. It is deeply integrated with the VMware environment and includes the following management software:

- VMware vSphere ESXi
- VMware vCenter Server
- VMware vSAN (software-defined storage)
- VMware vRealize Log Insight

vSphere ESXi

vSphere ESXi is a bare-metal hypervisor. The VxRail HCI System Software directly installs ESXi onto the physical server host. vSphere ESXi hosts and their resources are pooled into clusters that contain the CPU, memory, network, and storage resources that are available for allocation to the VMs. Clusters scale up to a maximum of 64 hosts and can support thousands of VMs.

vCenter Server

vCenter Server management software runs on a virtual or physical server to oversee multiple ESXi hypervisors as a single cluster. An administrator can interact directly with vCenter Server or use vSphere Client to manage VMs from a browser window anywhere in the world. For example, the administrator can capture the detailed blueprint of a known, validated configuration—a configuration that includes networking, storage, and security settings—and then deploy that blueprint to multiple ESXi hosts.

For day-to-day VM management, you can use vCenter Server to directly manage the VMware stack on the VxRail system. This vCenter Server, which orchestrates the cluster deployment and life cycle management, has the following limitations:

- It manages only the VxRail cluster on which it is deployed; it cannot manage other VxRail clusters or any other ESXi hosts.
- It cannot be used as a customer-supplied vCenter Server.
- It does not support Enhanced Linked Mode.
- The Single Sign On domain is vsphere.local and cannot be customized.
- It does not support encryption in VxRail versions earlier than 4.5.200.
- With stretched clusters, if an inter-switch link (ISL) failure occurs, all VMs that are not on the same site as the vCenter Server are powered off. Thus, special attention is required when planning to deploy an internal vCenter Server.

For more information, see the *Dell EMC VxRail vCenter Server Planning Guide*.

vSAN

VxRail HCI uses VMware vSAN software. vSAN is fully integrated with vSphere and provides full-featured and cost-effective software-defined storage. vSAN software is integrated directly into the hypervisor. This architecture distinguishes vSAN from other solutions that typically install a virtual storage appliance (VSA) running as a guest VM on each host. Embedding vSAN integration into the ESXi kernel layer has obvious advantages in performance and efficient memory footprint requirements. vSAN has little

impact on hypervisor CPU utilization (less than10 percent), and it self-balances based on workload and resource availability.

vSAN presents storage as a familiar data store construct and works seamlessly with other vSphere features such as vMotion. It aggregates locally attached disks from hosts in a vSphere cluster to create a pool of distributed shared storage. Capacity is easily scaled up by adding disks to the cluster nodes and scaled out by adding ESXi hosts. This distributed shared storage provides the flexibility to start with a small environment and scale it over time.

Storage characteristics are configured using vSphere Storage Policy Based Management (SPBM) software. SPBM enables you to set and modify object-level policies, whenever needed, to control storage provisioning and day-to-day storage management using service-level agreements (SLAs).

vSAN is preconfigured when a VxRail system is first initialized and is managed thereafter through vCenter. The VxRail system initialization process discovers locally attached storage disks from each ESXi node in the cluster to create a distributed, shared-storage data store. The amount of storage in the vSAN data store is an aggregate of all the drive capacity in the cluster. The vSAN software presents a robust, secure, and efficient shared datastore to all nodes within a VxRail cluster.

In-kernel vSAN enhances VxRail efficiency by providing:

- 1, 2, and 4 CPU sockets with 6x memory
- Native vSphere vMotion and vSphere Distributed Resource Scheduler (DRS)
- Simple, single management pane
- Per-VM management and policies

vRealize Log Insight

VMware vRealize Log Insight is a highly scalable log management tool designed for big data. Extensible across physical, virtual, and multicloud environments, vRealize Log Insight enables administrators to connect to everything in their environment, providing a single location for log collection, storage, and analysis. Its predictive analytics, machine learning, and root cause analysis tools enable faster problem resolution.

Key features include:

- Analysis of machine-generated log data—application logs, network traces, configuration files, messages, performance data, and system state dumps
- Real-time monitoring, actionable dashboards, machine learning, analytics, and more for infrastructure and applications
- Proactive issue detection and alerting
- Integrated knowledge of vSphere and other VMware products
- Integration with vRealize Operations, combining unstructured and structured data for end-to-end operations management
- Intuitive UI for simple interactive searches and for deep analytical queries
- Support for third-party extensions
- Customizable data retention

Big Data Cluster services

In our lab environment with Big Data Cluster on VxRail HCI, we balanced services across the four nodes. The following figure provides a visual representation of how we distributed the Big Data Cluster services:

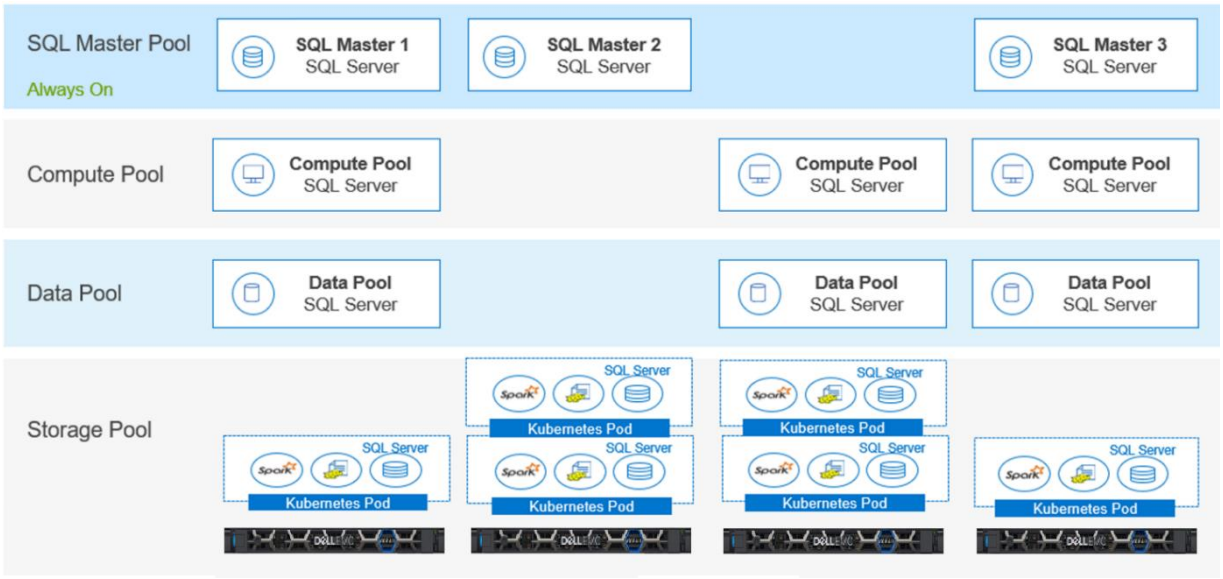

Figure 7. Distribution of Big Data Cluster services

The pools used the nodes as follows:

- The SQL Server master pool used nodes 1, 2, and 4.
- The compute pool used nodes 1, 3, and 4.
- The data pool used nodes 1, 2, 3, and 4.
- The storage pool used nodes 2, 3, and 4.

The fourth node supported the external load balancer. The three SQL master VMs were used in an Always On availability group. SQL Server automatically builds out the Always On availability group when the HA option is selected during deployment. The availability group includes system databases and is integrated into the SQL Server engine so that any newly created databases are automatically added to the availabilty group. The Always On availability group provides the SQL master database architecture with several benefits:

- Protects the SQL master database by providing automatic and planned failover options
- Distributes read-intensive queries to one or more secondary replicas
- Enables backup operations to be offloaded to the secondary replicas
- Provides automatic page repair to protect against page corruption

An Always On cluster provides important features for production implementations of Big Data Cluster, such as high availability, offload support, and self-healing capabilities.

The Big Data Cluster configuration on the VxRail system performed as expected with no challenges or obstacles. Offering easy deployment and integration with Kubernetes and VMware, VxRail HCI is an ideal platform for a Big Data Cluster environment.

Lab validation of SQL Server 2019 Big Data Cluster

Overview

This section broadly describes how we installed, configured, and tested SQL Server 2019 Big Data Cluster. Rather than including step-by-step instructions, we focus on the key elements of our successful Big Data Cluster implementation and provide links to documents that detail the installation requirements.

Based on the working assumption that virtualization is an existing part of the data center, we do not address VMware vSphere installation in this section. Many of the steps apply to bare metal and virtualization; thus, if the organization is considering bare metal, the steps still apply.

The high-level steps for installing, configuring, and testing this solution are as follows. The subsequent sections provide additional details.

1. Install Docker, Kubernetes, and the VMWare vSphere CSI plug-in:
   a. Install the Docker Enterprise Edition in a VM on the VxRail system. The operating system within the VM is Red Hat Enterprise Linux.
   b. Install and configure a local private Docker registry.
   c. Install Kubernetes.
   d. Install and configure the vSphere CSI plug-in.

   Note: Container services exist as instance images that reside in registries. Docker supports its own public registry service called the Docker Hub, which holds more than 2.7 million container applications, including Redis, MongoDB, and MySQL instances, which are all freely available. Similarly, Microsoft has its own Docker registry (Microsoft Container Registry), which holds all the images that are required for Big Data Cluster deployment.

2. Deploy Big Data Cluster:
   a. Pull the most recent SQL Server 2019 Big Data Cluster for Linux container image from the Microsoft Container Registry.
   b. Push the SQL Server 2019 Big Data Cluster to the image in the local private registry.
   c. Configure vSphere dynamic storage provisioning.
   d. Install Big Data Cluster.
   e. Import the TPC-H data into Big Data Cluster.
   f. Test data virtualization.
3. Migrate data to the data pool by ingesting data from the stand-alone SQL Server database.
4. Ingest 10 TB of TPC-H data into Big Data Cluster.

Step 1: Install Docker, Kubernetes, and the vSphere CSI plug-in

Step 1a: Install Docker Enterprise Edition

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

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

```
$ yum -y install docker-ee-19.03 docker-ee-cli-19.03
containerd.io
```

Note: Set up the Docker Enterprise repository before installing Docker.

In a vSphere environment, the VM limits CPU and memory resources. Collaboration between the virtualization and Docker administrators is important because most Docker environments have multiple containers. VMs hosting Docker containers tend to be larger than other VMs, requiring more CPU and memory resources to support containers.

Step 1b: Install the Docker registry

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

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

The Docker administrator must work closely with network engineers and security experts to address the placement of the Docker registry. Depending on the size of the container environment, the velocity can place a significant load on the network. Further, customized registry images might contain sensitive configuration settings that must remain secure.

For our lab validation, we used a local private registry, which addressed our key variety, velocity, and security requirements.

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

```
$ docker pull registry:2
$ mkdir -p /registry/private
docker run -d \
  -p 5000:5000 \
  --restart=always \
  --name registry \
  -v /home/dockerv/registry:/var/lib/registry \
  -v /certs:/certs \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/ca.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/ca.key \
  registry:2
```

Step 1c: Install Kubernetes

Many Kubernetes solutions are available today. For example, turnkey-managed Kubernetes offerings from cloud providers give IT organizations a zero-data-center-footprint solution that

requires no installation. An on-premises private-cloud Kubernetes implementation offers greater control and flexibility but requires investment in infrastructure and training. For this use case, we show an open-source 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.

Install Kubernetes as follows.

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

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

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

systemctl enable --now kubelet
```

Step 1d: Install the vSphere CSI plug-in

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

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

- Dynamic provisioning and decommissioning of volumes
- Attachment and detachment of volumes from a host node
- Mounting and unmounting of a volume from a host node
- Snapshot capabilities: create, delete, and list

The Kubernetes administrator works with the storage administrator to download, modify, and install the vSphere CSI plug-in. To configure Cloud Provider Interface (CPI) and CSI, see the following VMware document on GitHub: *Deploying a Kubernetes Cluster on vSphere with CSI and CPI*.

After you install and configure the CSI plug-in on the VxRail system, check the status of the CSI plug-in-related pods. Check the status by running the `kubectl get pods -n kube-system | grep vsphere` command, as shown in the following example:

```
[root@m01 ~]# kubectl get pods -n kube-system | grep vsphere
vsphere-cloud-controller-manager-5xrxt   1/1     Running   1          6d20h
vsphere-cloud-controller-manager-7dbfs   1/1     Running   0          6d20h
vsphere-cloud-controller-manager-zp2bh   1/1     Running   3          6d20h
vsphere-csi-controller-0                 5/5     Running   0          6d20h
vsphere-csi-node-2qmmt                   3/3     Running   3          6d20h
vsphere-csi-node-94lcd                   3/3     Running   3          6d20h
vsphere-csi-node-9fccg                   3/3     Running   3          6d20h
vsphere-csi-node-hx6dr                   3/3     Running   3          6d20h
vsphere-csi-node-j5pv4                   3/3     Running   3          6d20h
vsphere-csi-node-jh8hw                   3/3     Running   3          6d20h
vsphere-csi-node-krlwr                   3/3     Running   3          6d20h
vsphere-csi-node-kxsxr                   3/3     Running   0          6d20h
vsphere-csi-node-lblss                   3/3     Running   3          6d20h
vsphere-csi-node-qjkkr                   3/3     Running   3          6d20h
vsphere-csi-node-qtwst                   3/3     Running   3          6d20h
vsphere-csi-node-vp4ln                   3/3     Running   3          6d20h
[root@m01 ~]#
```

Figure 8. vSphere CSI plug-in running on Kubernetes worker nodes

Step 2: Deploy Big Data Cluster

Step 2a: Pull the Big Data Cluster image from the Microsoft Container Registry

The Docker administrator downloads the latest Big Data Cluster image from the Microsoft Container Registry to the local private registry. The Docker administrator collaborates with the SQL Server administrator to ensure that all stated requirements for this Big Data Cluster image are met.

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

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

```
$ docker pull
mcr.start.com/mssql/bdc/<SOURCE_IMAGE_NAME>;<SOURCE_DOCKER_TAG>
```

This Docker pull command shows how to manually pull images from trusted public registries. Because we were performing an offline installation from our local private registry, we used a Python script to automate pulling of all the required images.

For details about how to use a Python script to automate this work, see Perform an offline deployment of a SQL Server big data cluster in Microsoft SQL Docs. The key step is to download the script, using the Bash shell, with Curl:

```
curl -o push-bdc-images-to-custom-private-repo.py
"https://raw.githubusercontent.com/Microsoft/sql-server-
samples/master/samples/features/sql-big-data-
cluster/deployment/offline/push-bdc-images-to-custom-private-
repo.py"
```

Step 2b: Push the Big Data Cluster image to the local private registry

To manually populate the local private Docker registry with the Big Data Cluster, the Docker administrator runs the following Docker `push` command:

```
$ docker push
localhost:5000/<SOURCE_IMAGE_NAME>;<SOURCE_DOCKER_TAG>
```

To automate populating the local registry, run the Microsoft Python script with the Linux command:

```
sudo python push-bdc-images-to-custom-private-repo.py
```

Note: If you customize the Big Data Cluster container image (we did not), save the base image or images and any customization to the local private registry with appropriate annotations, if required for your use case.

Step 2c: Configure vSphere dynamic storage provisioning

A key question related to the implementation of this Big Data Cluster environment concerns storage provisioning. In a bare-metal infrastructure, the control plane is Kubernetes. But when vSphere is part of the application stack, does the control plane remain with Kubernetes or vSphere?

vSphere Storage Policy Based Management (SPBM) provides a single unified control plane for storage services such as Kubernetes. SPBM is an abstraction layer that enables disparate storage services to be unified under a universal framework for vSphere. The use of SPBM enables vSphere administrators to manage storage in a container environment, as shown in the following figure:

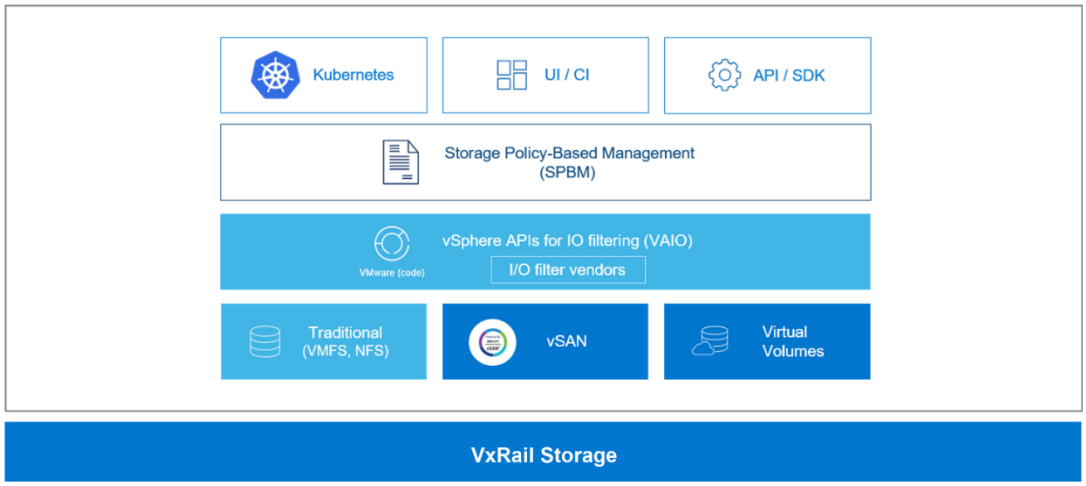

Figure 9. vSphere SPBM

The management process includes creating StorageClass definitions for dynamic storage provisioning. The StorageClass parameters are contained in a YAML file. Parameters in the YAML file for VMFS and NFS include:

- **diskformat**: `thin`, `zeroedthick`, `eagerzeroedthick`
- **storagePolicyName**: *<user defined, example: gold>*

Use of StorageClass parameters enables granular control of persistent storage provisioning.

```
# cat SQLBDC-storageclass.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: sqlbdc-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: csi.vsphere.vmware.com
parameters:
  storagepolicyname: "Space-Efficient"
  fstype: ext4
# kubectl get storageclass sqlbdc-sc
NAME         PROVISIONER                    AGE
sqlbdc-sc   csi.vsphere.vmware.com   5s
```

After the storage classes have been defined, Kubernetes can create PersistentVolumes for Big Data Cluster.

Note: For more information about using vSphere to dynamically provision storage in a Kubernetes environment, see Dynamic Provisioning on GitHub and Storage Policy Based Management in VMware Docs.

Step 2d: Install Big Data Cluster

Big Data Cluster installation prerequisites include installing tools such as Python, azdata, and kubectl. For detailed installation steps, see How to deploy SQL Server Big Data Clusters on Kubernetes in Microsoft SQL Docs.

1. Run the following command to create a copy of the base deployment profile in the custom directory from the source bdc-test-profile01:

   ```
   azdata bdc config init --source bdc-test-profile –target custom
   ```

2. Customize your copy of the base deployment profile for your infrastructure by using Visual Studio Code or a similar JSON editing tool.

3. Run the following command to install Big Data Cluster:

   ```
   azdata bdc create --config-profile custom –accept-eula yes
   ```

   In approximately 15 to 30 minutes, the following notification is displayed in the window where the installation was started, indicating that the controller pod is up and running:

   ```
   Cluster controller endpoint is available at
   <IP_address>:<port_number>

   Cluster control plane is ready.
   ```

When the entire installation is complete, the following message is displayed:

```
Cluster deployed successfully.
```

The use of a local private registry requires more preparation but significantly accelerates the deployment of Big Data Cluster. In contrast, an installation using the Microsoft Container Registry requires pulling all the image files over the Internet, which might take hours for every Big Data Cluster installation.

Useful commands to check the health of the cluster post installation are:

- `azdata bdc endpoint list -o table`—Provides a description of each service with its corresponding endpoint, as shown in the following figure:

```
C:\Users\admin>azdata bdc endpoint list -o table
Description                                          Endpoint                                                  Name                       Protoco
---------------------------------------------------  --------------------------------------------------------  -------------------------  -------
Gateway to access HDFS files, Spark                  https://[redacted]3                                       gateway                    https
Spark Jobs Management and Monitoring Dashboard       https://[redacted]3/gateway/default/sparkhistory          spark-history              https
Spark Diagnostics and Monitoring Dashboard           https://[redacted]3/gateway/default/yarn                  yarn-ui                    https
Application Proxy                                    https://[redacted]3                                       app-proxy                  https
Management Proxy                                     https://[redacted]7                                       mgmtproxy                  https
Log Search Dashboard                                 https://[redacted]7/kibana                                logsui                     https
Metrics Dashboard                                    https://[redacted]7/grafana                               metricsui                  https
Cluster Management Service                           https://[redacted]9                                       controller                 https
SQL Server Master Instance Front-End                 [redacted]                                                sql-server-master          tds
SQL Server Master Readable Secondary Replicas        [redacted]                                                sql-server-master-readonly tds
HDFS File System Proxy                               https://[redacted]/gateway/default/webhdfs/v1             webhdfs                    https
Proxy for running Spark statements, jobs, applications https://[redacted]/gateway/default/livy/v1             livy                       https

C:\Users\admin>
```

Figure 10. Big Data Cluster services and corresponding endpoints

- `azdata bdc status show`—Shows a detailed list of service names, their state, health status, and details

The following figure shows the status of SQL Server 2019 Big Data Cluster from Azure Data Studio:

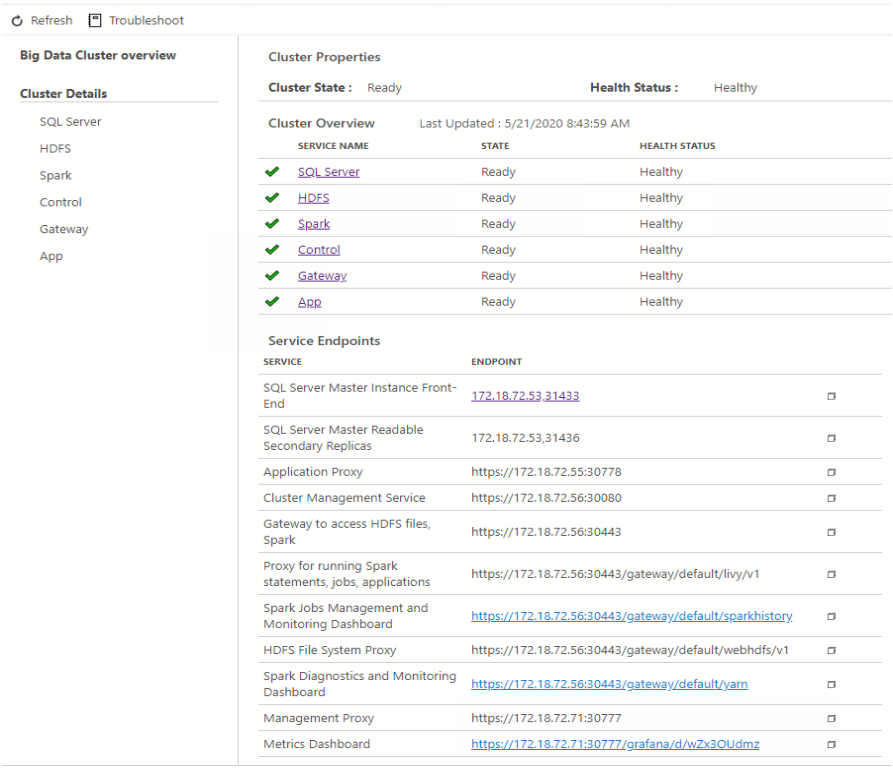

Figure 11. Big Data Cluster status in Azure Data Studio

Key benefits of automated Big Data Cluster installation

The simplicity of the automated installation of Big Data Cluster overshadows the significant value that the automation provides. The application architecture is complex, consisting of many tools and software stacks that might take days, if not weeks, to manually install. In this step, coordination of azdata, Kubernetes, and Docker enables automated installation of Big Data Cluster in minutes or hours. This time savings reflects one of the core benefits of containers, which is the capability to provision Big Data Cluster and other applications quickly.

Kubernetes and azdata combined are the automation and orchestration engines. When the Big Data Cluster installation is complete, the containers, configuration, and storage are provisioned, and the cluster is in a ready state. This automation and orchestration move Big Data Cluster into a self-serve, on-demand platform. The IT organization can then easily stand up multiple Big Data Clusters for different teams within the organization.

Some Big Data Clusters, including dev/test Big Data Clusters that are regularly refreshed, might only be needed for a few months. Docker containers are designed to be ephemeral—that is, they can be stopped and destroyed easily, and new containers can be built from the same image and put in place with minimal setup and configuration. The ephemeral nature of Docker containers is ideal for dev/test scenarios, where most developers and other team members use the containers for only a short time.

Step 2e: Import the TPC-H data into Big Data Cluster

You can ingest data into Big Data Cluster through any of the following methods:

- Load data into the storage pool by using the Curl utility. See Using Curl to ingest data.
- Load data into the data pool by using SQL Server and T-SQL. See Using SQL Server and T-SQL to ingest data.
- Load data into the data pool by using Spark jobs. See Using Spark jobs to ingest data.

The optimal method for ingesting data depends on the amount of data, type of data, and the target location of data. For example, if the goal is to ingest files into the storage pool (HDFS data node), using the Curl command line works well. In contrast, using SQL Server and Spark jobs works well if the goal is to ingest data into the data pool. In this use case, the goal was to ingest approximately 1 TB of TPC-H data into the Big Data Cluster storage pool, with smaller tables going into a stand-alone Oracle database and a stand-alone SQL Server database.

TPC-H data uses eight distinct and separate tables. The larger tables with more rows were ingested into the Big Data Cluster data pool. The medium-size tables were ingested into a stand-alone SQL Server database unrelated to the Big Data Cluster configuration. Finally, the smallest tables were ingested into the Oracle database. The goal was to place the largest tables in Big Data Cluster.

The following table summarizes where the TPC-H data was ingested:

Table 2. TPC-H data locations after ingestion

Data location	Table	Scale Factor (SF)	Table size
Big Data Cluster storage pool (HDFS)	PARTSUPP	SF*800,000	Large
	CUSTOMER	SF*150,000	Large
	LINEITEM	SF*6,000,000	Large
	ORDERS	SF*1,500,000	Large
Big Data Cluster data pool (SQL Server)	PART	SF*200,000	Medium
	SUPPLIER	SF*10,000	Medium
External data source—MongoDB	NATION	25	Small
External data source—Oracle	REGION	5	Small

The scale factor column in the table refers to the size of the TPC-H database in gigabytes. In our use case, the database size was 1,000 GB (1 TB); thus, the scale factor is 1,000. To calculate the number of rows in a table, multiply the SF by the base row number. For example, for the first table, PARTSUPP:

1,000 (SF) * 800,000 = 8,000,000,000 rows

For more information, see *TPC Benchmark H*.

Key benefits of Big Data Cluster data ingestion

A key benefit of Big Data Cluster is the ability to easily ingest big data. In our use case, we used Curl to ingest approximately 1 TB of decision-support data. The tutorials in Microsoft SQL Docs enabled our SQL Server experts to quickly start the ingest of data. In testing, we took one extra step to optimize performance by moving from comma-separated value (CSV) files in HDFS to Parquet files, as described in Using Curl to ingest data.

By placing the medium-size tables in a stand-alone SQL Server database and the smaller tables in the Oracle database, our data virtualization tests validate:

- **Access to disparate data in its native form**—The capability to access data without any transformation.
- **Access to data outside of Big Data Cluster**—The capability to access data almost anywhere and without modification, that is, without having to modify or install an agent on the data source to facilitate data access.

Using Curl to ingest data

Curl is a command-line tool that is designed for transferring data over a broad set of protocols. The benefit of using Curl is the ability to put local files into HDFS, which is the approach that we used in our lab testing.

We cleansed the TPC-H data and saved it to a CSV file. We then used a PowerShell script to dynamically create and run Curl to ingest the CSV file into HDFS. Part of the script creates an HDFS directory:

```
curl -L -k -u root:%KNOX_PASSWORD% -X PUT
"https://%KNOX_ENDPOINT%/gateway/webhdfs/v1/%DIR_NAME%/csv/PART?O
P=MKDIRS"
```

where:

- `%KNOX_PASSWORD%` is the root user's password for the Apache KNOX gateway.
- `%KNOX_ENDPOINT%` is the KNOX IP and port in <*KNOX_IP*>:<*KNOX_PORT*> format.
- `%DIR_NAME%` is the name of directory that will be created at HDFS (if it does not exist); the CSV files will be uploaded to this location.

We then put the CSV files into the HDFS directory:

```
curl -L -k -u root:%KNOX_PASSWORD% -X PUT
https://%KNOX_ENDPOINT%/gateway/webhdfs/v1/%DIR_NAME%/csv/part/fi
nal_F00001_part.csv?op=create&overwrite=true -H "Content-
type:application/octet-stream" -T
"C:\stage\ps\v5\final\part\final_F00001_part.csv"
```

In the last Curl command, the file name is `final_F00001_part.csv` because the CSV file was split into eight parts and placed in the HDFS storage pool. For each of the eight files, the number in the file name was incremented (00001–00008). Splitting the file into eight parts enabled distribution of the data across the nodes in the storage pool, which optimizes performance.

In addition, to improve performance and gain capacity savings, we converted the CSV files in HDFS to Parquet files. Parquet is an Apache Software Foundation tool that is used to convert files into columnar data. When the data is in columnar form, Parquet can efficiently compress and encode it.

Using SQL Server and T-SQL to ingest data

A second way to ingest data is to bring it into the data pool by using SQL Server and T-SQL. The first step in this process is to create an external table in the data pool. An external table is an object that enables PolyBase to access data that is stored outside the database. Using an external table, PolyBase can access data in an Oracle database, Hadoop cluster, Azure blob storage, and, as in our lab tests, a text file with TPC-H data.

Create and use an external table, and then ingest data, as follows:

1. Connect to the SQL Server master instance and create an external data source to the remote database:

```
IF NOT EXISTS(SELECT * FROM sys.external_data_sources
          WHERE name = 'SqlDataPool')
  CREATE EXTERNAL DATA SOURCE SqlDataPool
  WITH (LOCATION = 'sqldatapool://controller-svc/default');
GO
```

2. Create the schema at the storage pool (HDFS) that serves as the source table (`EXT_HDFS_PARTSUPP`) on the storage pool and points to the Parquet files on the `/TPCH1TB/parquet/partsupp` directory on HDFS:

```
USE [TPCH1TB]
GO

IF NOT EXISTS(SELECT * FROM sys.external_tables WHERE name
= 'EXT_HDFS_PARTSUPP')
    CREATE EXTERNAL TABLE EXT_HDFS_PARTSUPP (
      "PS_PARTKEY" BIGINT NOT NULL,
      "PS_SUPPKEY" bigint,
      "PS_AVAILQTY" INT,
      "PS_SUPPLYCOST" float,
      "PS_COMMENT" VARCHAR(199)
)
    WITH
    (
        DATA_SOURCE = SqlStoragePool,
        LOCATION = '/TPCH1GB/parquet/partsupp',
        FILE_FORMAT = parquet_file
    );
GO
```

3. Create an external table named `EXT_DP_PARTSUPP` in the data pool with round-robin data distribution:

```
   IF NOT EXISTS(SELECT * FROM sys.external_tables WHERE
name = 'EXT_DP_PARTSUPP')
   CREATE EXTERNAL TABLE [EXT_DP_PARTSUPP] (
```

```
      "PS_PARTKEY" BIGINT NOT NULL,
      "PS_SUPPKEY" bigint,
      "PS_AVAILQTY" INT,
      "PS_SUPPLYCOST" float,
      "PS_COMMENT" VARCHAR(199)
    )
   WITH (
      DATA_SOURCE = SqlDataPool,
      DISTRIBUTION = ROUND_ROBIN
   );
GO
```

4. Ingest the data into the data pool:

```
INSERT INTO EXT_DP_PARTSUPP(
      "PS_PARTKEY",
      "PS_SUPPKEY",
      "PS_AVAILQTY",
      "PS_SUPPLYCOST",
      "PS_COMMENT"
)
SELECT
      "PS_PARTKEY",
      "PS_SUPPKEY",
      "PS_AVAILQTY",
      "PS_SUPPLYCOST",
      "PS_COMMENT"
FROM EXT_HDFS_PARTSUPP
GO
```

5. To test if the ingest was successful, select count from the external table to show the number of rows:

```
Select count(1) from [owner_name].[external table name]
```

   For example:

```
Select count(1) from [dbo].[EXT_DP_PARTSUPP]
```

Using Spark jobs to ingest data

The third way to ingest data is with Spark jobs. The following steps provide a broad outline of how to ingest data using a Spark job.

Note: For detailed steps, see Tutorial: Ingest data into a SQL Server data pool with Spark jobs in Microsoft SQL Docs.

---

1. Create an external data source.
2. Create an external table.

   See the example in Using SQL Server and T-SQL to ingest data.
3. Configure Spark-SQL connector parameters and run the Spark job.

Step 2f: Test data virtualization

After ingesting the decision-support data into Big Data Cluster, SQL Server, MongoDB, and Oracle, test data virtualization as follows:

1. Create a data source for Big Data Cluster, Oracle, and MongoDB.

   A data source is a row in the `sys.external_data_sources` table in the Big Data Cluster master SQL Server database. The row contains the name, source, and location of the data source. Creating data sources enables PolyBase in Big Data Cluster to access the other database systems.

   Note: The stand-alone SQL Server database requires no data source because Big Data Cluster can natively access the data.

   a. Run the following script to create a data source for the Big Data Cluster storage pool:

   ```
   IF NOT EXISTS(SELECT * FROM sys.external_data_sources
   WHERE name = 'SqlStoragePool')
       CREATE EXTERNAL DATA SOURCE SqlStoragePool
       WITH (LOCATION = 'sqlhdfs://controller-
   svc/default');
   ```

   b. Create a data source for the Oracle database as follows:

      i Create a master key for encrypting the database password:

      ```
      CREATE MASTER KEY ENCRYPTION BY PASSWORD =
      '@StrongPasword!!!';
      ```

      ii Create the credentials to connect to the Oracle database in the master SQL Server instance.

      The following T-SQL command assigns a name (oracle_cred) and saves a user account (oracle_user) with the password that you created in the preceding step:

      ```
      CREATE DATABASE SCOPED CREDENTIAL oracle_cred
      WITH IDENTITY = 'oracle_user', SECRET =
      'StrongOraclePassword!!!!';
      ```

      iii Run the following T-SQL command to finish creating the data source for the Oracle database:

      ```
      CREATE EXTERNAL DATA SOURCE Oracle_Data_Source
      WITH
      (    LOCATION   =
      'oracle://<Oracle_instance_IP>:<Oracle_Instance_Port
      >'
      ,    CREDENTIAL = oracle_cred
      );
      ```

c. Create a data source for the MongoDB database as follows:

i Create the MongoDB database credentials:

```
USE DATABASE TPCH1TB
GO
CREATE DATABASE SCOPED CREDENTIAL mongodb_poly
WITH IDENTITY = 'superuser', Secret = 'test123';
```

ii Create the data source:

```
CREATE EXTERNAL DATA SOURCE source_mongodb_poly
WITH (
LOCATION = 'mongodb://mongoserver:27017',
 CONNECTION_OPTIONS='SSL=0',
 PUSHDOWN = OFF,
 CREDENTIAL = mongodb_poly
);
```

2. For Big Data Cluster, create external tables that define the DDL of the remote tables:

```
IF NOT EXISTS(SELECT * FROM sys.external_tables WHERE name
= 'EXT_HDFS_ORDERS')
    CREATE EXTERNAL TABLE EXT_HDFS_ORDERS (
      "O_ORDERKEY" bigint NOT NULL, "O_CUSTKEY" bigint NOT
NULL,
"O_ORDERSTATUS" char(1) NOT NULL,
"O_TOTALPRICE" decimal(12, 2) NOT NULL,
"O_ORDERDATE" date NOT NULL,
      "O_ORDERPRIORITY" char(15) NOT NULL,
"O_CLERK" char(15) NOT NULL,
"O_SHIPPRIORITY" int NOT NULL,
"O_COMMENT" varchar(79) NOT NULL
      )
    WITH
    (
        DATA_SOURCE = SqlStoragePool,
        LOCATION = '/TPCH1GB/parquet/orders',
        FILE_FORMAT = parquet_file
    );
GO
```

This script:

- Uses a table name (EXT_HDFS_PART) that easily identifies the table type (EXT), location (HDFS), and the name (PART)
- Uses the DDL to define the table
- References the data source and location
- References the file format—in this case, the Parquet files in the HDFS storage pool

3. For Oracle, create external tables that define the DDL of the remote tables, as follows:

```
IF NOT EXISTS(SELECT * FROM sys.external_tables WHERE name
= 'EXT_ORA_REGION')
CREATE EXTERNAL TABLE EXT_ORA_REGION (
"R_REGIONKEY" int,
"R_NAME" Varchar(25) COLLATE Latin1_General_100_BIN2_UTF8,
"R_COMMENT" VARCHAR(152) COLLATE
Latin1_General_100_BIN2_UTF8)
WITH (DATA_SOURCE=bdcOracleDataSource,
LOCATION='<oracle-database>.<oracle-schema>.REGION');
GO
```

4. For MongoDB, create external tables that define the DDL of the remote tables, as follows:

```
IF NOT EXISTS(SELECT * FROM sys.external_tables WHERE name
= 'EXT_MONGO_NATION')
CREATE EXTERNAL TABLE EXT_MONGO_NATION (
N_NATIONKEY int,
N_NAME NVARCHAR(25) COLLATE SQL_Latin1_General_CP1_CI_AS,
N_REGIONKEY int,
N_COMMENT NVARCHAR(152) COLLATE SQL_Latin1_General_CP1_CI_A
S)
WITH(
    LOCATION='TPCH1TB.nation',
    DATA_SOURCE=source_mongodb_poly
);
GO
```

   For this use case, we created additional external tables for relevant tables in HDFS, Oracle, and MongoDB.

5. Implement container automation.

   When the setup of connecting and defining external data sources is complete, the transition to data virtualization is seamless. Now, consider whether to use custom images or scripting for container automation.

   If you will need to refresh the Big Data Cluster environment, save the state of the existing containers as custom images in the registry. Saving these custom images enables you to deploy a Big Data Cluster environment with the data virtualization configuration completed. Developers and data scientists can then start working without having to wait for additional setup.

   Another approach is container automation through scripting. In testing Big Data Cluster, our experts created scripts to automate manual steps such as creating connections to external data resources. As part of the validation work, the Big Data Cluster was refreshed to ensure that the steps were repeatable. The initial container rollout took approximately 3 hours. Any refresh of the Big Data Cluster system took approximately 30 minutes. Scripting the post-installation steps

worked well and enabled teams to start using the Big Data Cluster system quickly.

6. Run a T-SQL query that uses all the decision-support tables.

   The ability to access all three data sources without any data transformation proves the value of data virtualization. The TPC-H benchmark has several queries that run against decision-support data. We selected TPC-H query 8, which is called National Market Share, to test our data virtualization configuration:

```
/* TPC_H  Query 8 - National Market Share */

SELECT      O_YEAR,
      SUM(CASE    WHEN  NATION      = 'CHINA'
                  THEN  VOLUME
                  ELSE  0
                  END) / SUM(VOLUME)      AS MKT_SHARE
FROM  (     SELECT      datepart(yy,O_ORDERDATE)
      AS O_YEAR,
                  L_EXTENDEDPRICE * (1-L_DISCOUNT)    AS
VOLUME,
                  N2.N_NAME                     AS NATION
            FROM  EXT_SQL_PART,     -- Points to SQL Server
                  EXT_SQL_SUPPLIER, -- Points to SQL Server
                  EXT_HDFS_LINEITEM, -- Points to HDFS
                  EXT_HDFS_ORDERS,  -- Points to HDFS
                  EXT_HDFS_CUSTOMER,      -- Points to HDFS
                  EXT_MONGO_NATION N1,    -- Points to
Mongo
                  EXT_MONGO_NATION N2,    -- Points to
Mongo
                  EXT_ORA_REGION    -- Points to Oracle
            WHERE P_PARTKEY   = L_PARTKEY AND
                  S_SUPPKEY   = L_SUPPKEY AND
                  L_ORDERKEY  = O_ORDERKEY AND
                  O_CUSTKEY   = C_CUSTKEY AND
                  C_NATIONKEY = N1.N_NATIONKEY AND
                  N1.N_REGIONKEY    = R_REGIONKEY AND
                  R_NAME            = 'ASIA' AND
                  S_NATIONKEY = N2.N_NATIONKEY AND
                  O_ORDERDATE BETWEEN '1995-01-01' AND
'1996-12-31' AND
                  P_TYPE            = 'SMALL BRUSHED
COPPER'
      )     AS    ALL_NATIONS
GROUP BY    O_YEAR
ORDER BY    O_YEAR
```

The body of the query (between FROM and WHERE) shows all the selected tables. This query uses seven of the eight tables in the decision-support benchmark. (PARTSUPP was not selected.) Of the seven selected tables, each of the four data sources (SQL, HDFS, Oracle, and MongoDB) were used:

- Big Data Cluster: EXT_HDFS_LINEITEM, EXT_HDFS_ORDERS, EXT_HDFS_CUSTOMER
- SQL Server: EXT_SQL_SUPPLIER, EXT_SQL_PART
- Oracle database: EXT_ORA_REGION
- MongoDB database: EXT_MONGO_NATION

The query completed quickly with no errors. PolyBase with Big Data Cluster enabled data virtualization.

Key benefits of implementing Big Data Cluster with data virtualization

Data scientists and developers can now access external databases without having to transform the data or migrate it into a data warehouse or data lake. In this use case, data virtualization simplified accessing data and eliminated the complexity that is associated with ETL processes. Less complexity can translate into time savings. Further, in our testing of Big Data Cluster with PolyBase data virtualization, we experienced impressively fast connection to external data sources.

Big Data Cluster unifies and centralizes big data and connects to external data sources. Combining big data and data virtualization gives data scientists one place to access information. Centralized data access, a common set of tools, and the ability to access information with less complexity provides the foundation for increased productivity.

Step 3: Migrate data to the data pool

Data scientists and others at times might need to move data from the original source to the Big Data Cluster data pool. For example, they might need to work on a subset of the data for data analytics, or they might need to accelerate performance. The primary use of the data pool is to offload data processing from the SQL Server master instance and, in doing so, accelerate performance. The data pool can accelerate performance in the following ways:

- **Offloading of data processing to servers in the data pool**—The data pool can consist of one or more VxRail nodes running SQL Server. The VxRail system in our use case had three SQL Server data pool instances, each on a separate node. The capability to distribute processing can accelerate analytics.
- **Parallelization of data processing**—Beyond the advantage of the physical configuration enabling offloading of data processing to separate nodes, the SQL Server master instance can parse and delegate processing. Distribution of the workload across the SQL Server instances further accelerates data processing.
- **Use of shards**—Data that is ingested into the data pool is distributed across the SQL Server instances. Distribution across the instances benefits performance. Because each SQL Server instance in the data pool reads a small subset of information, the master SQL Server instance receives results more quickly.

In this use case, we ingested all the data from the stand-alone SQL Server instance into the data pool. The stand-alone SQL Server instance was selected because it contained TPC-H tables that were larger than those tables in the Oracle database.

Ingest the data as follows:

1. Create the external data source:

```
IF NOT EXISTS(SELECT * FROM sys.external_data_sources WHERE
name = 'SqlDataPool')
  CREATE EXTERNAL DATA SOURCE SqlDataPool
  WITH (LOCATION = 'sqldatapool://controller-svc/default');
GO
```

2. In the data pool, create an external table named `BDC_DP_SUPPLIER`:

```
IF NOT EXISTS(SELECT * FROM sys.external_tables WHERE name
= 'BDC_DP_SUPPLIER')
   CREATE EXTERNAL TABLE [BDC_DP_SUPPLIER] (
"S_SUPPKEY" BIGINT ,
"S_NAME" CHAR(25),
"S_ADDRESS" VARCHAR(40),
"S_NATIONKEY" INT,
"S_PHONE" CHAR(15),
"S_ACCTBAL" Decimal(12,2),
"S_COMMENT" VARCHAR(101) )
   WITH (
      DATA_SOURCE = SqlDataPool,
      DISTRIBUTION = ROUND_ROBIN
   );
GO
```

3. Ingest the data into the data pool:

```
INSERT INTO BDC_DP_SUPPLIER(
    "S_SUPPKEY",
"S_NAME",
"S_ADDRESS",
"S_NATIONKEY",
"S_PHONE",
"S_ACCTBAL",
"S_COMMENT"
)
SELECT "S_SUPPKEY",
"S_NAME",
"S_ADDRESS",
"S_NATIONKEY",
"S_PHONE",
"S_ACCTBAL",
"S_COMMENT"
FROM EXT_SQL_SUPPLIER
GO
```

After migrating the database to the data pool, you can use the following script as a test.

Note: The following script selects data only from tables in the data pool, Oracle, and HDFS.

```
/* TPC_H  Query 8 - National Market Share */

SELECT O_YEAR,
       SUM(CASE      WHEN   NATION = 'CHINA'
                     THEN   VOLUME
                     ELSE   0
                     END) / SUM(VOLUME)  AS MKT_SHARE
FROM   (      SELECT datepart(yy,O_ORDERDATE)          AS O_YEAR,
                     L_EXTENDEDPRICE * (1-L_DISCOUNT)  AS VOLUME,
                     N2.N_NAME                         AS NATION
              FROM   BDC_DP_PART,        -- Points to Data Pool
                     BDC_DP_SUPPLIER,    -- Points to Data Pool
                     BDC_DP_LINEITEM,    -- Points to Data Pool
                     BDC_DP_ORDERS,      -- Points to Data Pool
                     BDC_DP_CUSTOMER,    -- Points to Data Pool
                     BDC_DP_NATION N1,   -- Points to Data Pool
                     BDC_DP_NATION N2,   -- Points to Data Pool
                     BDC_DP_REGION-- Points to Data Pool
              WHERE  P_PARTKEY     = L_PARTKEY AND
                     S_SUPPKEY     = L_SUPPKEY AND
                     L_ORDERKEY    = O_ORDERKEY AND
                     O_CUSTKEY     = C_CUSTKEY AND
                     C_NATIONKEY   = N1.N_NATIONKEY AND
                     N1.N_REGIONKEY      = R_REGIONKEY AND
                     R_NAME        = 'ASIA' AND
                     S_NATIONKEY   = N2.N_NATIONKEY AND
                     O_ORDERDATE   BETWEEN '1995-01-01' AND '1996-12-31'
AND
                     P_TYPE        = 'SMALL BRUSHED COPPER'
       )      AS     ALL_NATIONS
GROUP  BY     O_YEAR
ORDER  BY     O_YEAR
```

Reviewing performance was not the goal of this white paper; however, our SQL Server experts did observe performance. They first modified the 22 queries that the TPC-H benchmark uses. They then ran queries against the data in the data pool and compared the time it took to run those queries to the time it took to run the queries against SQL Server, Oracle, MongoDB, and HDFS. Before reviewing the result, note that the data pool was not designed for performance, and the amount of data that was migrated to the data pool was not large. Thus, any performance findings are not significant.

All 22 queries ran faster on the data in the Big Data Cluster data pool than when they ran on the virtualized dataset outside of Big Data Cluster.

Using the data pool accelerated 78 percent of the TPC-H queries. Depending on how your organization plans to use Big Data Cluster, you can configure your VxRail system to address your performance requirements.

Conclusion

This paper provides an overview of SQL Server 2019 Big Data Cluster with VxRail HCI. It illustrates how developers and data scientists can benefit from using this data management and analytics platform with Docker containers, Kubernetes, vSphere, and SQL Server on Red Hat Enterprise Linux.

We have used these reference configurations with various sizes of big datasets to characterize and tune the cluster nodes, pods, and HDFS parameters to help you get a head start at deploying this solution. Using Intel Xeon Scalable processors, VxRail HCI allows customers to start small and grow by scaling up capacity and performance.

Our case study addresses big data storage and tools for handling big data. We split the eight tables of TPC-H data across multiple data sources—Big Data Cluster storage pool (HDFS), Big Data Cluster data pool (SQL Server), and stand-alone MongoDB database and Oracle database instances. Our SQL Server Solutions engineering team modified all 22 of the TPC-H queries so that tables were selected from different data sources. Using data virtualization with PolyBase, the queries were successful, running without error and returning the results that joined all four data sources.

Data virtualization does not involve physically copying and moving the data, so the data is available to business users in real time. Big Data Cluster simplifies and centralizes access to and analysis of the organization's data sphere. It enables IT to simplify management by consolidating big data and data virtualization on one platform with a proven set of tools.

To help ensure an organization's success on the digital transformation journey, IT needs a highly scalable infrastructure and service automation. We showed how the combination of Docker containers, Kubernetes, and the vSphere CSI plug-in enables fast and easy provisioning of the Big Data Cluster services. The initial installation of Big Data Cluster took approximately 3 hours. With automation, subsequent refreshes took less than 30 minutes.

In our use case, the key to this increase in deployment speed was the capability to seamlessly provision persistent storage on the VxRail system. Without automated storage provisioning, administration would have been required for all the Big Data Cluster services—the SQL Server master instance, data pool, compute pool, and storage pool.

Our testing shows that customers can virtualize Docker containers and achieve important benefits including the capability to securely isolate a Big Data Cluster instance on a VxRail system.

Resources

SQL Server solutions

For more information about SQL Server solutions, see the Dell Technologies Solutions for Microsoft SQL Server Info Hub.

VxRail HCI

For more information about VxRail HCI, see the following resources:

- Info Hub: Dell EMC VxRail
- Network Planning Guide: Dell EMC VxRail
- Info Hub: Dell EMC VxRail Networking Solutions
- White Paper: Dell EMC VxRail—Comprehensive Security by Design
- TechBook: Dell EMC VxRail System
- Quick Start Guide: Dell EMC VxRail
- YouTube: Dell EMC VxRail on YouTube

Appendix A: Design architecture and component specifications

Physical architecture

The following figure shows the validated design architecture:

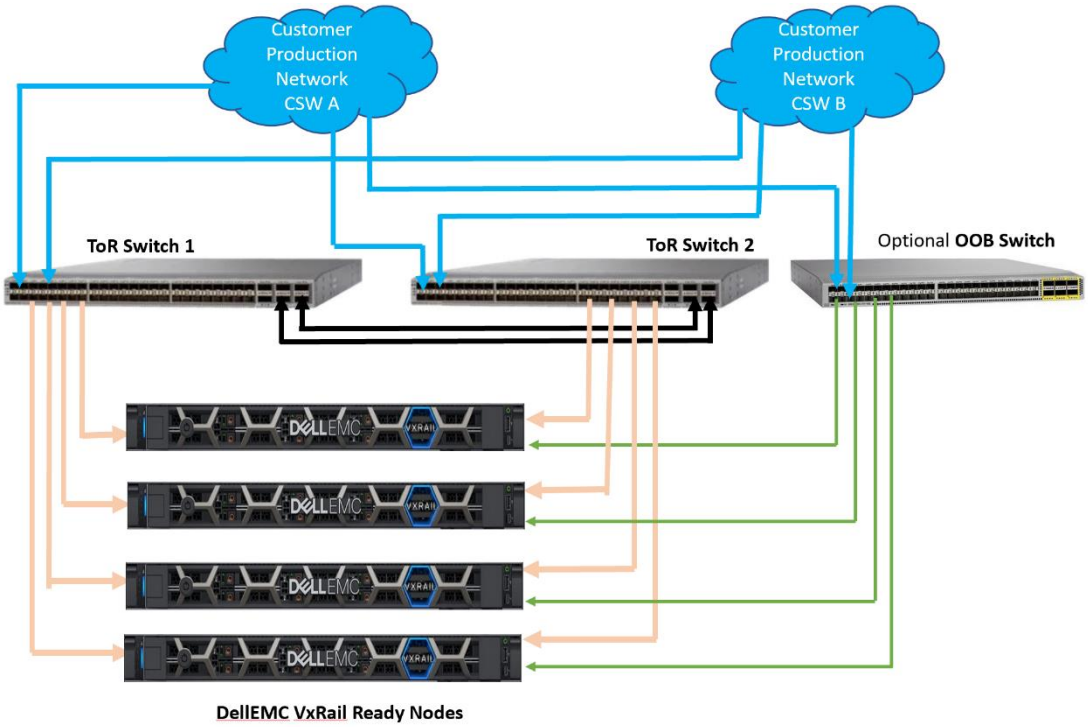

Figure 12. Physical architecture

Server layer

All VxRail nodes support Big Data Cluster deployment. For our use case, we chose the VxRail E560F, as described in the following table:

Table 3. VxRail E560F specifications

Component	Details
Form factor	Single system, 1U
Processors	Single or dual socket, up to 28 cores per CPU
Memory	Up to 3,072 GB RAM (1,536 GB per processor)
Additional network options	Up to 1 NIC per processor 2 x 10 GbE SFP+, 2 x10GbE RJ45, 4x10GbE RJ45, 2 x 25 GbE SFP28, 2 x 100 GbE QSFP28
Drive configuration	• 10 x 12 Gbps 2.5-inch SAS drive slots • 1 or 2 disk groups with up to 4 capacity drives each
Max capacity	76.8 TB SSD or 19.2 TB hard drive
Power supplies	1,100 W 100–240 V AC, 1,100 W 48 V DC, 1,600 W 200–240 V AC
Boot	BOSS with 2 x 240 GB SATA M.2

For more details about server node specifications and configurations for VxRail HCI, see the *Dell EMC VxRail product page*.

The following table lists the details of the VxRail E560F server that we used for our lab validation:

Table 4. VxRail E560F server cluster configuration

Component	Details
Processors per server	2 x Intel Xeon Gold 6512 @ 2.10 GHz
Cores per server	44
vCores (threads) per server	88
DIMMs per server	12 x 64 GB @ 2,666 MHz
Memory per server	768 GB
rNDC	1 x Intel 10 GbE 4P X710 rNDC
NICs	1 x Intel 10 GbE 2P X710 adapter
Boot disks	2 x 224 GB SSDs @ 6 Gbps
Disk groups	2
Cache disks per disk group	1 x 800 GB Toshiba SSD @ 12 Gbps
Capacity disks per disk group	3 x 3.84 TB Toshiba SSD @ 12 Gbps
Power supplies	2 x 1,100 W

Network layer

The network layer consists of:

- **Two 10 GbE network switches**—Provide data connectivity for the VxRail cluster
- **Two 1/10 GbE network switches**—Provide OOB management network connectivity for the VxRail cluster

The physical connections between the ports on the network switches and the NICs on the VxRail nodes enable communications for the virtual infrastructure within the VxRail cluster. The virtual infrastructure within the VxRail cluster uses the virtual distributed switch to enable communication within the cluster, and communication to IT management and the application user community.

VxRail predefined logical networks manage and control traffic within the cluster and outside of the cluster. Certain VxRail logical networks must be made accessible to the outside community. For instance, IT management requires connectivity to the VxRail management system, and end users and application owners require access to their VMs running in the VxRail cluster. The network traffic supporting I/O to the vSAN datastore or the vMotion network used to dynamically migrate VMs between VxRail nodes to balance workload can stay within the VxRail cluster; otherwise, they can be configured with a routable network. The internal network used for device discovery is isolated and does not exit the ToR switches.

Virtual LANs (VLANs) define the VxRail logical networks within the cluster. They also define the method that is used to control the paths a logical network can pass through. A VLAN, represented as a numeric ID, is assigned to a VxRail logical network. The same

VLAN ID is also configured on the individual ports on the ToR switches and on the virtual ports in the virtual distributed switch during the automated implementation process.

When an application or service in the VxRail cluster sends a network packet on the virtual distributed switch, the VLAN ID for the logical network is attached to the packet. The packet can only pass through the ports on the ToR switch and the virtual distributed switch when the VLAN IDs match. We highly recommend isolating the VxRail logical network traffic by using separate VLANs. We recommend using a “flat” network only for test or nonproduction purposes.

The following figure shows the network layer of the VxRail cluster:

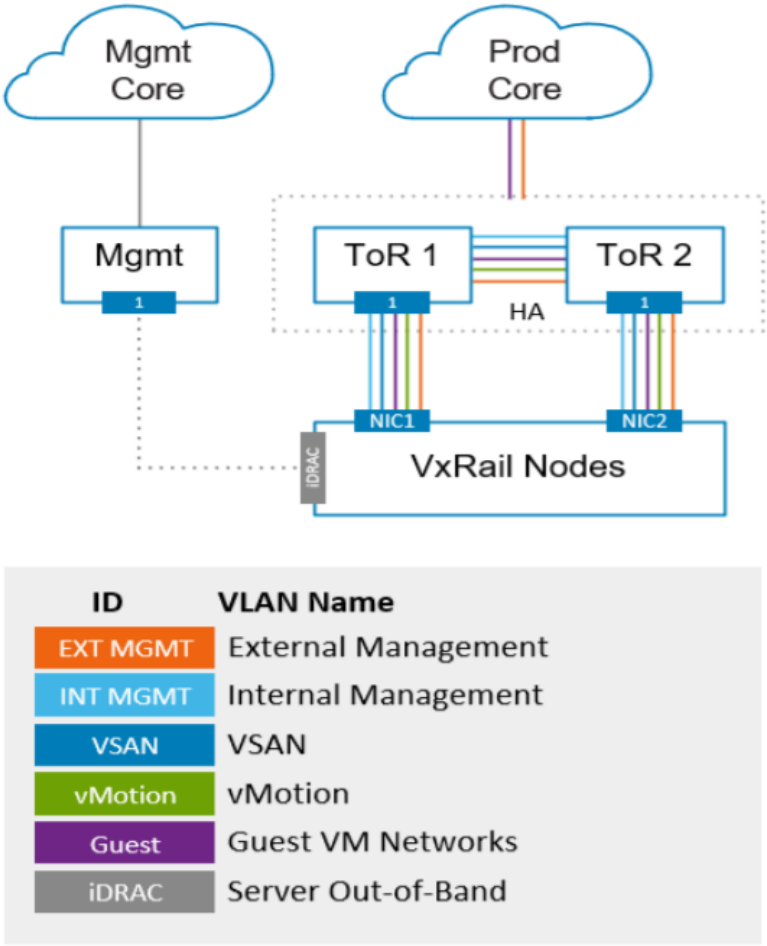

Figure 13. Network layer

The VxRail system groups the logical networks in the following categories: External Management, Internal Management, vSAN, vSphere vMotion, and Virtual Machine. The system assigns the settings that you specify for each of these logical networks during the initialization process.

Before VxRail version 4.7, both external and internal management traffic shared the external management network. Starting with VxRail 4.7, the external and internal management networks are separate networks.

VMware Cloud Native Storage

Cloud Native Storage is integrated with vSphere to enable comprehensive data management for stateful cloud-native applications. It includes a CSI plug-in for Kubernetes and a control plane within vCenter. Using storage that is exposed by vSphere, Cloud Native Storage enables you to create persistent container volumes that are independent of the VM and container life cycle.

For more information, see *Getting Started with VMware Cloud Native Storage* in VMware Docs.

Software components

The following table lists the software that we used during our design validation:

Table 5. Software components

Software component	Version
Dell EMC VxRail	4.7.400
VMware vCenter and ESXi	6.7 u3
Red Hat Enterprise Linux operating system	7.6 Maipo; kernel: 3.10.0-957.el7.x86_64
Docker	19.03 Enterprise Edition
Kubernetes	1.14.2
Pod network	Flannel
NGINX (load balancer)	1.18
SQL Server 2019	Big Data Cluster 2019 CU4
azdata	15.0.4033

Kubernetes cluster architecture

The following figure depicts the high-level architecture for our Kubernetes cluster:

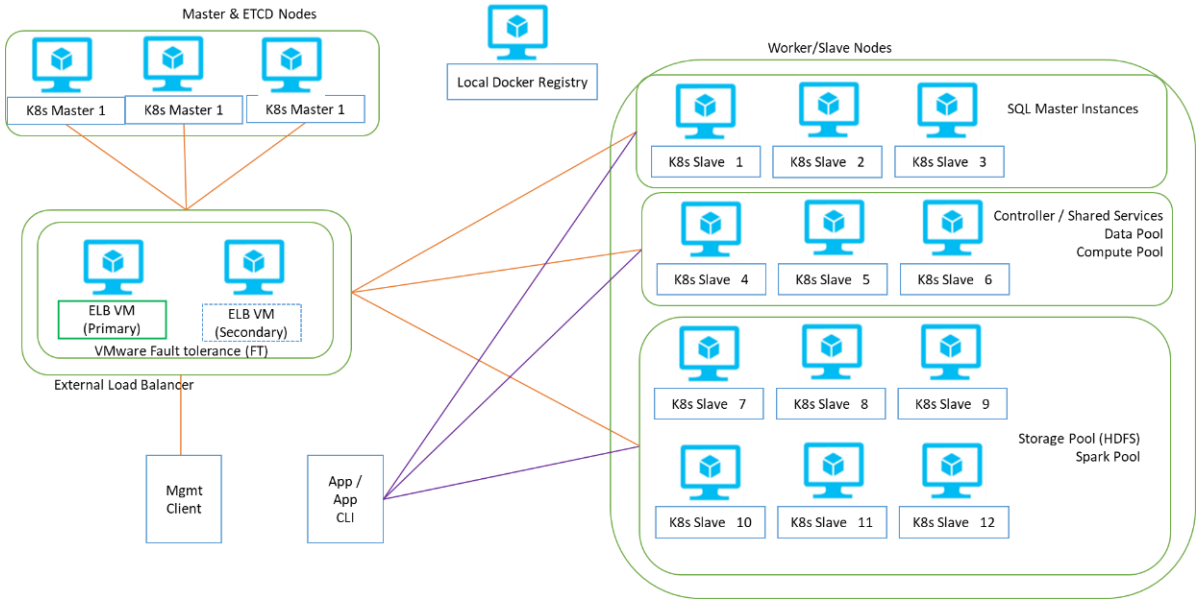

Figure 14. High-level architecture of our Kubernetes cluster

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 also 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 with HA configured.
- **Kubernetes master nodes**—Three dedicated Red Hat Enterprise Linux VMs provide HA for Kubernetes master nodes in case of a failure. The etcd is also deployed on these three master nodes. You can also deploy separate etcd cluster nodes in their own dedicated VM.
- **Kubernetes worker nodes**—Nine dedicated Red Hat Enterprise Linux VMs work as workload driver nodes. We put the SQL Server Big Data Cluster pods on these VMs.

The following table shows the VM placement across the four-node VxRail cluster:

Table 6. VM placement on VxRail cluster nodes

Physical host	vCPU physical host	Memory (GB) physical host	VM name	vCPU on VM	vMemory (GB) on VM	Description
Host 1	88	768	kubeLB (primary)	4	16	Load balancer VM
			k8s-m1	4	16	Kubernetes master VM
			w2-bdc-m2	16	160	SQL Big Data Cluster master VM
			w6-bdc-dp3	16	160	SQL Big Data Cluster data and compute pool VM
				56	512	
Host 2	88	768	k8s-m2	4	16	Kubernetes master VM
			w3-bdc-m3	16	160	SQL Big Data Cluster master VM
				52	496	
Host 3	88	768	k8s-m3	4	16	Kubernetes master VM
			w4-bdc-dp1	16	160	SQL Big Data Cluster data and compute pool VM
				52	496	
Host 4	88	768	Docker Registry	4	16	Local Docker secure registry
			kubeLB (secondary)	4	16	Load balancer VM
			w1-bdc-m1	16	160	SQL Big Data Cluster master VM
			w5-bdc-dp2	16	160	SQL Big Data Cluster data and compute pool VM
				56	512	

SQL Server Big Data Cluster components

SQL Server Big Data Cluster consists of the following planes for component aggregation:

- **Control plane—**This plane consists of SQL master instances along with monitoring and logging components. It also hosts other shared components such as Hive, SQL Proxy, and so on.
- **Compute plane—**This plane consists of SQL Server instances to offload compute workload from a SQL master instance. This plane does not store any user data.
- **Data plane—**This plane consists of a SQL Server instance that hosts user data so that the master instance can offload any data (tables with full data or intermediate data) to this plane. This plane also hosts semi-structured and nonstructured data.

Our design distributes the SQL Server Big Data Cluster components across dedicated worker nodes in the Kubernetes cluster, as shown in the following figure. This distribution enables better performance and improved workload segregation:

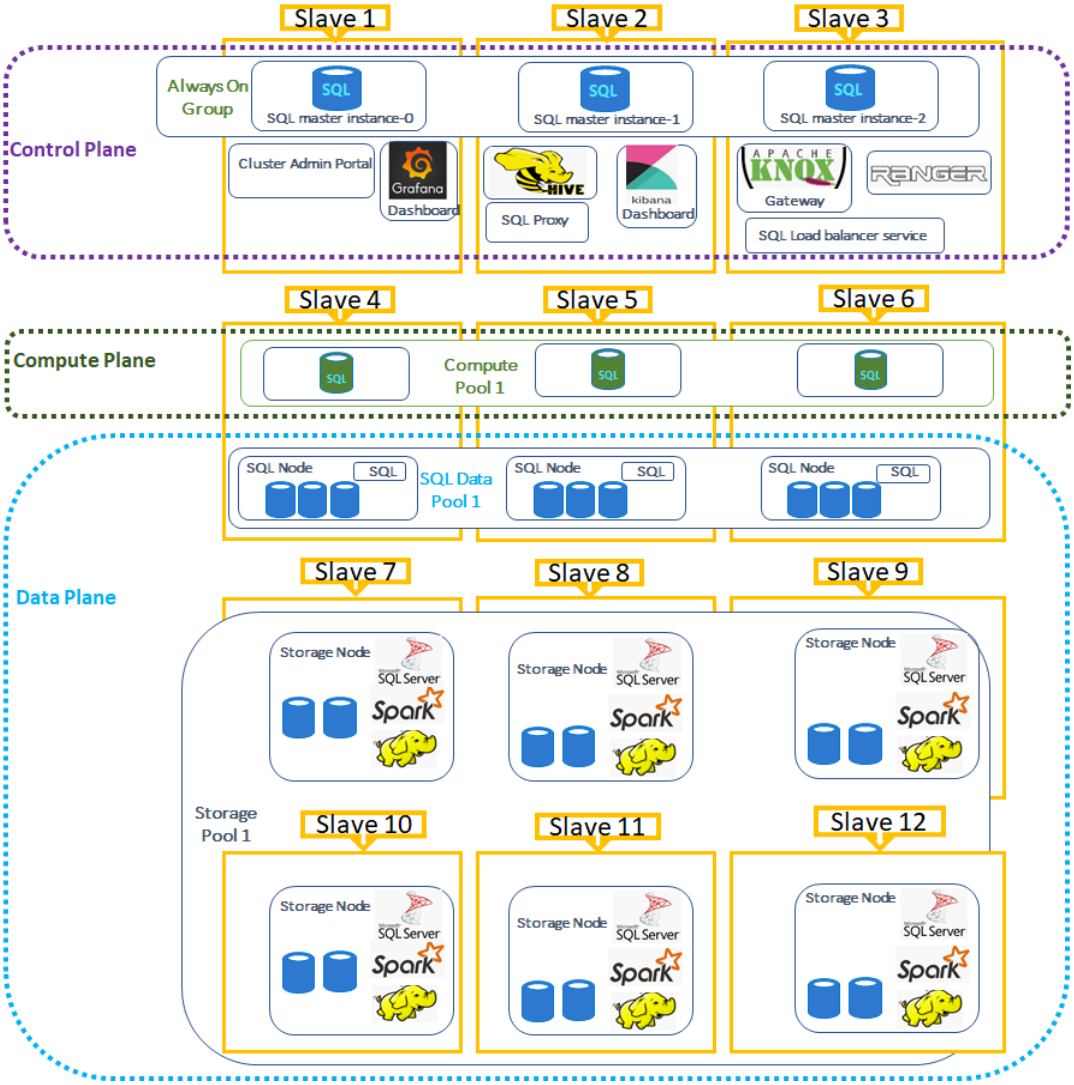

Figure 15. Distribution of Big Data Cluster components

Apart from component-aggregating planes, the SQL Server Big Data Cluster also has multiple pools:

- **Compute pool**—SQL Server instances to offload compute workloads from master instances
- **Data pool**—SQL Server instances to offload data from master instances
- **Storage pool**—SQL server instance, Spark components, and HDFS components to store and process structured, nonstructured, or semi-structured data

The following separate pools also are part of SQL Server Big Data Cluster deployments; however, these pools are not shown in Figure 15 because we did not use them exclusively in our validation scenarios:

- **App pool**—Any application container that is directly inside SQL Server Big Data Cluster
- **Spark pool**—Spark components (segregated from the storage pool)