

Building a Red Hat OpenShift Environment on IBM Z

Lydia Parziale

Erica Ross

Alexandre de Oliveira

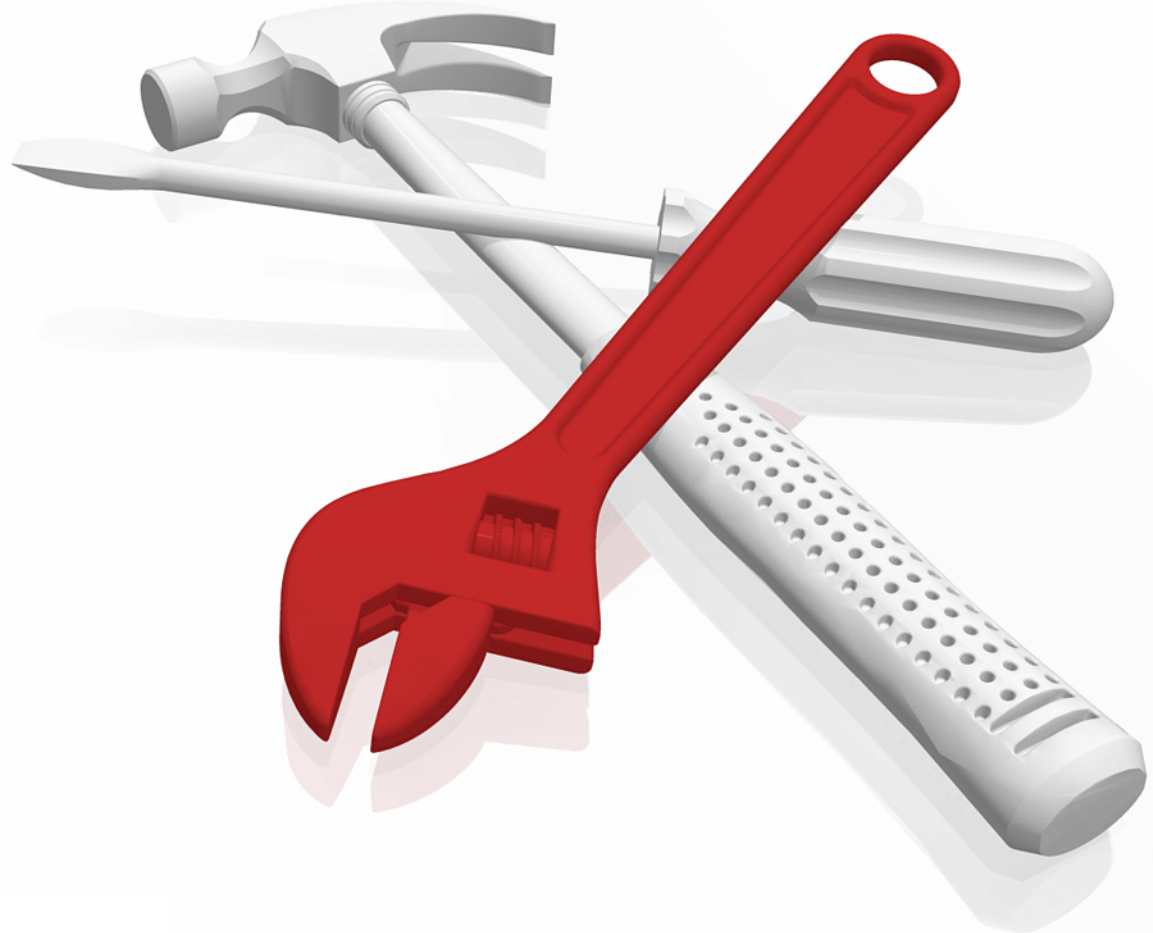
Anna Shugol

Elton De Souza

Manoj Srinivasan

Rakesh Krishnakumar

Wilhelm Mild





IBM Redbooks

Building a Red Hat OpenShift Environment on IBM Z

August 2022

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (August 2022)

This edition applies to Red Hat OpenShift and IBM Cloud Pak for Data.

© Copyright International Business Machines Corporation 2022. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	ix
Now you can become a published author, too!	xi
Comments welcome	xi
Stay connected to IBM Redbooks	xii
Chapter 1. Hybrid cloud overview and introduction to containers and microservices concepts	1
1.1 Executive summary	2
1.1.1 Why you should use Red Hat OpenShift Container Platform	3
1.2 Architectures and tools that are used in this book	5
1.2.1 Microservices	5
1.2.2 Containers	7
1.2.3 Kubernetes	8
1.2.4 Containers versus virtualization	9
1.2.5 Container orchestration frameworks	11
1.3 Red Hat OpenShift architecture	13
1.3.1 Red Hat OpenShift components	15
1.3.2 Pods and ReplicaSets	17
1.4 Red Hat OpenShift Container Platform introduction	18
1.4.1 Red Hat OpenShift Container Platform colocated with traditional workloads . . .	19
1.4.2 IBM Cloud Paks for Data Overview	19
1.5 Use cases overview	19
Chapter 2. Architectural overview	21
2.1 Topology planning	22
2.2 Environment considerations	23
2.2.1 Virtualization options	23
2.2.2 Hardware topology options	24
2.2.3 Red Hat OpenShift Container Platform cluster topology	24
2.2.4 Network topology	25
2.2.5 Storage topology	30
2.2.6 Capacity cores and memory	33
2.2.7 Security	34
2.3 Sizing	36
2.3.1 Workload characteristics	36
2.3.2 Consolidation ratio or factor	36
2.3.3 Multi-tenant or single-tenant cluster	38
2.3.4 Sizing control planes, compute nodes, and infrastructure nodes	39
2.3.5 Sample scenario	40
2.3.6 Cores, memory, and storage	43
2.4 Continuous availability for applications	46
2.4.1 Avoiding single points of failure	47
2.4.2 Deploying applications and integrating DevOps	48
2.5 Planning for HA and DR infrastructure	49
2.5.1 High availability considerations	49

2.5.2 Planning for disaster recovery.	52
Chapter 3. Red Hat OpenShift Container Platform installation for KVM on LinuxONE	55
3.1 Architectural patterns: Red Hat OpenShift cluster on KVM.	56
3.1.1 Storage connectivity	57
3.1.2 Network Connectivity	58
3.1.3 Virtual image format	59
3.2 High-level Red Hat OpenShift installation steps	60
3.3 Red Hat OpenShift Container Platform prerequisites	61
3.3.1 KVM HOST configuration	61
3.3.2 MacVTap configuration.	61
3.3.3 Infrastructure (Bastion) Node deployment and prerequisites	63
3.3.4 Setting up and configuring HAProxy	64
3.4 Deploying HAProxy	68
3.4.1 HTTP server configuration	70
3.4.2 Red Hat OpenShift cluster CLI and certificate generation	70
3.4.3 Creating the installation configuration file	71
3.4.4 Generating manifest and ignition files.	72
3.4.5 Installing Red Hat OpenShift cluster.	73
3.4.6 Validating Red Hat OpenShift cluster deployment	74
3.4.7 Approving Red Hat OpenShift certificates.	75
Chapter 4. Use Case: Cloud Pak for Data platform that is running containerized Db2 service	77
4.1 Overview	78
4.2 Architecture	79
4.3 Solution implementation	80
4.4 Best practices	87
Chapter 5. Use Case: A Red Hat OpenShift environment that is colocated with z/OS transactional services and DB2 data	89
5.1 Colocation benefits on IBM Z	90
5.1.1 Performance and efficiency	90
5.1.2 Operational benefits	90
5.2 Workload and architecture	92
Appendix A. Installing Red Hat OpenShift Container Storage on IBM Z	95
A.1 Overview.	96
A.2 Architecture.	97
A.3 Planning and requirements.	99
A.3.1 Planning	99
A.3.2 Resource requirements	99
A.3.3 Node requirements	100
A.3.4 Storage requirements	100
A.4 Installation.	101
A.4.1 Lab environment	101
A.4.2 Assigning disks to KVM Guest	102
A.4.3 Red Hat OpenShift Container Storage installation by using Operator Hub	106
A.5 Red Hat OpenShift Container Platform registry integration with Red Hat OpenShift Container Storage	126
Related publications	133
IBM Redbooks	133
Online resources	133

Help from IBM	133
---------------------	-----

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Db2®	IBM Research®	Think®
FICON®	IBM Spectrum®	WebSphere®
GDPS®	IBM Watson®	z/OS®
HyperSwap®	IBM Z®	z/VM®
IBM®	Parallel Sysplex®	z/VSE®
IBM Cloud®	Redbooks®	
IBM Cloud Pak®	Redbooks (logo)  ®	

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Ceph, JBoss, OpenShift, Red Hat, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

Cybersecurity is the most important arm of defense against cyberattacks. With the recent increase in cyberattacks, corporations must focus on how they are combating these new high-tech threats.

When establishing best practices, a corporation must focus on employees' access to specific workspaces and information. IBM Z® focuses on allowing high processing virtual environments while maintaining a high level of security in each workspace.

Organizations not only need to adjust their approach to security, but also their approach to IT environments. To meet new customer needs and expectations, organizations must take a more agile approach to their business.

IBM® Z allows companies to work with hybrid and multi-cloud environments that allows more ease of use for the user and efficiency overall. Working with IBM Z, organizations can also work with many databases that are included in IBM Cloud Pak® for Data. IBM Cloud Pak for Data allows organizations to make more informed decisions with improved data usage.

Along with the improved data usage, organizations can see the effects from working in a Red Hat OpenShift environment. Red Hat OpenShift is compatible across many hardware services and allows the user to run applications in the most efficient manner.

The purpose of this IBM Redbooks® publication is to:

- ▶ Introduce IBM Z and LinuxONE platforms and how they work with the Red Hat OpenShift environment and IBM Cloud Pak for Data
- ▶ Provide examples and the uses of IBM Z with Cloud Paks for Data that show data gravity, consistent development experience, and consolidation and business resiliency

The target audience for this book is IBM Z Technical Specialists, IT Architects, and System Administrators.

Authors

This book was produced by a team of specialists from around the world working at IBM Redbooks, Poughkeepsie Center.

Lydia Parziale is a Project Leader for the IBM Redbooks team in Poughkeepsie, New York, with domestic and international experience in technology management, including software development, project leadership, and strategic planning. Her areas of expertise include business development and database management technologies. Lydia is a PMI certified PMP and an IBM Certified IT Specialist with an MBA in Technology Management and has been employed by IBM for over 25 years in various technology areas.

Erica Ross is a Summit Technical Solution Specialist Intern in Fort Worth, Texas US. She has three internship experiences in the software engineering and data analytics field. She has a degree in Business Information Systems from Texas Christian University. Her areas of expertise include RPA, Data Analytics, and the programming languages Java, Python, R, and SQL. She has written extensively about RPA.

Alexandre de Oliveira is an IT Architect with over 20 years of experience in information technology, supporting pre-sales and sales team to develop new businesses. Alexandre is an expert in containers, Kubernetes, and OpenShift in different platforms, such as Intel (x86) and Mainframe(s390x). He has an MBA on Business Management by Universidade de São Paulo (USP).

Anna Shugol is an IBM Z Hybrid Cloud Solutions Engineer, working for a global team and based in Winchester, England. She has joined IBM in 2011. Anna holds a Computer Science degree from Moscow State Technical University n.a. Bauman. Her areas of interest include large systems performance, containers on IBM Z, and digital assets.

Elton De Souza is based out of the IBM Canada lab and has worked on the IBM Z platform his entire career at IBM. He currently leads Cloud Native Technology adoption on the IBM Z/LinuxONE platforms. The first half of his career was spent on the internals of Java where he worked on exploiting over 200 hardware instructions on IBM Z for mission critical mobile and cloud workloads. He was one of the first technical experts for Docker/Kubernetes on IBM Z in early 2015 and since then, he has worked with IBM's largest clients on successful adoption of cloud native technology, such as Kubernetes, IBM Cloud® Private, and most recently Red Hat OpenShift and IBM Cloud Paks and IBM Hyper Protect Services. He has over 50 publications, contributes to several open-source projects, and leads the design and development of the CareKit SDK for Hyper Protect in partnership with Apple, which uses IBM LinuxONE-based services in IBM Cloud.

Manoj Srinivasan is a Consulting IT Specialist and Emerging Technology (Open Source) enthusiast focused on helping organizations to pursue and pioneer with new transformative and disruptive technologies (such as Blockchain, Machine Learning, and Analytics) through collaborative strategy development, architect solutions, and provide deep dive workshops. Manoj is a distinguished speaker, co-author of several advanced technology publications, and a vocal advocate for open innovation and Startup culture.

Rakesh Krishnakumar is a Software Architect in IBM Competitive Insights team based out Singapore. He is IBM certified Executive IT specialist and Open Group certified Distinguished Technical Specialist with a total IBM Z industry experience of approximately 23 years. His areas of expertise include performing competitive research around various new technology solutions on IBM Z and providing IT Economics consultancy to customers. His consulting work includes new technology solutions around Linux on IBM Z and z/VM® across customers in the Asia Pacific geography, managing Linux on IBM Z development and test life-cycles, and managing various IBM Z implementations in the ISA (India South Asia) market. He has co-authored several IBM Redbooks publications.

Wilhelm Mild is a IBM Executive IT Architect in IBM Research® and Development Laboratory in Germany and an Open Group Certified Distinguished Architect. He has over 2 decades of experience in designing integrated solution architectures for Virtualized heterogeneous environments with Linux on IBM Z® or IBM LinuxONE and traditional workloads. He holds a degree in Computer Science. His areas of expertise are in Digitalization, Mobile, Containerization, Integration Architectures, and Red Hat OpenShift, including Resiliency and High Availability concepts. Wilhelm is extensively teaching workshops and is a speaker in many international conferences and customer events.

Thanks to the following people for their contributions to this project:

Robert Haimowitz
IBM Redbooks, Poughkeepsie Center

Patrik Hysky
IBM Systems Technical Sales Services, Austin

Tom Ambrosio, Bill Lamastro,
IBM CPO

Susanne Wintenberger
IBM Boeblingen

Nilton C dos Santos
IBM Australia

Vinodkumar Ramalingam, Ravinder Aula
IBM India

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an IBM Redbooks residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- Send your comments in an email to:

redbooks@us.ibm.com

- Mail your comments to:

IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>



Hybrid cloud overview and introduction to containers and microservices concepts

In this chapter, we introduce some of the basic concepts of hybrid multi-cloud workloads and especially how IBM Z and IBM LinuxONE play a crucial role in such environments. The basic concepts of microservices and containers and their operational and management frameworks also are discussed.

We also discuss the path of the use of containerization and how it plays a vital role in the hybrid multi-cloud strategy.

This chapter includes the following topics:

- ▶ 1.1, “Executive summary” on page 2
- ▶ 1.2, “Architectures and tools that are used in this book” on page 5
- ▶ 1.3, “Red Hat OpenShift architecture” on page 13
- ▶ 1.4, “Red Hat OpenShift Container Platform introduction” on page 18
- ▶ 1.5, “Use cases overview” on page 19

1.1 Executive summary

The evolution of the IT environment continues with the transformation of organizations across the globe. This evolution is moving toward a rapid adoption of software solutions in hybrid and multi-cloud environments to achieve competitive advantages and ensure customer satisfaction.

The application landscape experiences flexibility by adopting hybrid application development and containerization of microservices and integrating it with traditional applications, with focus on the requirements for secured multi-cloud service landscapes.

Many applications are deployed in a private cloud because of security and compliance aspects, data affinity, and performance requirements. The IT organizations that are responsible for operating the private cloud value simplicity, agility, flexibility, security, and cost efficiency. These features reduce their own barriers to innovation as part of their overall hybrid strategy. A significant focus is on IT optimization toward a better use and operation of dynamic transactional workloads and the use of statistical and AI capabilities for the hybrid data.

The IBM Z and IBM LinuxONE platforms are suited for these hybrid multi-cloud environments. They host cloud services that can take advantage of the capability of nondisruptive vertical and horizontal scalability on-demand on the most securable platform by inheriting the reliability, stability, and availability of the mainframe. In colocation with traditional workloads, such as IBM z/OS® data or services, they can enable data gravity in secure computing environments and fulfill the business and IT requirements of today and tomorrow.

IBM positions Red Hat OpenShift Container Platform as the foundational Platform as a Service (PaaS) technology for providing IT as a Service (ITaaS). It also positions the rapid provisioning and lifecycle management of containerized applications and associated application and infrastructure services for cloud users, such as software developers, data scientists, and solution architects.

Red Hat OpenShift Container Platform is the only solution that runs across multiple hardware architectures, such as IBM Z, IBM Power, x86, Arm, and Microsoft. It can run as a private cloud implementation or combined with public clouds. It implements the hybrid capability of running mission-critical applications where they fit best and most effectively.

The value of Red Hat OpenShift Container Platform features the following key characteristics:

- ▶ Implements an enterprise Kubernetes platform for container workloads
- ▶ Enables seamless Kubernetes deployments on any cloud or on-premises environments
- ▶ Features an integrated and automated installation
- ▶ Performs seamless platform and application updates
- ▶ Auto-scales resources and services
- ▶ Integrates tools for development consistency experience
- ▶ Runs enterprise workloads with enterprise CI/CD services, across multiple deployments

With the capability to run Red Hat OpenShift Container Platform on IBM Z and IBM LinuxONE, you can take full advantage of the following platform capabilities and characteristics:

- ▶ Nondisruptive growth with vertical and horizontal scalability that helps to accommodate substantial increase of workload on-demand and unpredictable peak requests.
- ▶ Highest scalability for millions of containers and thousands of Linux guests in one physical machine. For more information, see this IBM Newsroom [web page](#).

- ▶ Containerized workload within a state of the art microservices architecture, while accessing existing transactional z/OS or Linux on IBM Z services and databases.
- ▶ Advantage of highest security and confidential cloud computing, taking advantage of FIPS 140-2 Level 4 certification of the IBM Z Cryptographic accelerators. For more information, see this [web page](#).
- ▶ Multi tenancy with full LPAR isolation (EAL5+) allows administrators to share a single hardware securely. Even virtual machines (VMs) on the same hardware offer EAL4 certification.

1.1.1 Why you should use Red Hat OpenShift Container Platform

Uses of Red Hat OpenShift environment on IBM Z and LinuxONE span multiple industries and show the advantages of deploying a private cloud solution with programmable Infrastructure as a Service (IaaS), Containers as a Service (CaaS), and PaaS capabilities.

Major uses include the following the examples:

▶ Data gravity

A key reason how the Red Hat OpenShift Container Platform can take advantage of the IBM Z and IBM LinuxONE platform is the colocation of containerized applications with traditional workloads, such as databases, transactional systems, or other traditional workloads running in Linux on IBM Z or z/OS. In such a scenario, the applications can be located close to the data to optimize latency, response time, deployment, security, service, and cost.

In recent projects, clients experienced a double digit factor improvement for colocated Red Hat OpenShift environments on IBM Z versus public clouds or distributed environments. The reason is the short communication path to traditional data and services because of the advantage of internal networks in IBM Z, accelerated secure requests, and the effective virtualization for a demanded scalability in the application and service responses.

▶ Consistent development experience

An effective development and deployment capability is highly appreciated in the enterprises and Red Hat OpenShift Container Platform enabled it along with the possibility to automatically deploy the containers on multiple platforms simultaneously.

The software components and Red Hat OpenShift Container Platform Add-ons enable a consistent development experience across multiple hardware architectures and platforms. Therefore, the development can be done by using the following Red Hat OpenShift Container Platform internal tools and components that make it easier to build the containers for a solution:

- Red Hat OpenShift do (**odo**) is a command-line tool that enables coding, building, and debugging the application.
- Red Hat OpenShift CodeReady Workspaces is the capability Red Hat OpenShift Container Platform offers with an integrated development environment in a browser with a graphical front end.
- Visual Studio with the **odo** plug-in enables the development in visual Studio for Red Hat OpenShift.

In addition to these development tools, you can take advantage of Red Hat OpenShift ServiceMesh to secure communications between your microservices, without changing the code.

Another useful add-on is Red Hat OpenShift Serverless, which allows you take advantage of the serverless concepts for auto-scaling and an economic use of resources of an application.

To automate the process for Continuous Integration (CI) and Continuous Delivery (CD), Red Hat OpenShift Pipelines that are based on the Tekton technology can be used.

The overall capability and experience of a common development for hybrid solutions is highly appreciated by users.

- Consolidation and business resiliency

When consolidating a Red Hat OpenShift Container Platform environment from x86 platforms to IBM Z and IBM LinuxONE, you can achieve economic and operational advantages. Most of the three-dimension scalability (vertical, horizontal, and combined) results in high flexibility without the need for a new hardware footprint, scalability on-demand, or granular capacity shift as needed. This key advantage bodes well for dynamic workloads and unpredicted growth.

For business resiliency, IBM Z and IBM LinuxONE provides per-design characteristics, such as an internal network, with significantly more reliability based on a higher degree of redundancy in the hardware. Because virtualization occurs within a single hardware environment, the networking traffic is more predictable with less latency compared to x86 environments.

Building a disaster recovery (DR) setup with IBM Z and IBM LinuxONE is much simpler in such an environment because fewer hardware units exist that must be managed. Most of all, you can take advantage of current Extended Disaster Recovery xDR solutions for IBM Z and IBM LinuxONE based on Geographically Dispersed IBM Parallel Sysplex® (IBM GDPS®).

In summary, the use cases can bring in key benefits to the business by using Red Hat OpenShift Container Platform on IBM Z and IBM LinuxONE.

Red Hat OpenShift Container Platform is the only offering that is running on multiple different on-premises hardware platforms and public clouds (see Figure 1-1). It unites the capabilities of Hybrid Multi-cloud environments, managed from a single point of control with its multi-cloud management capabilities.

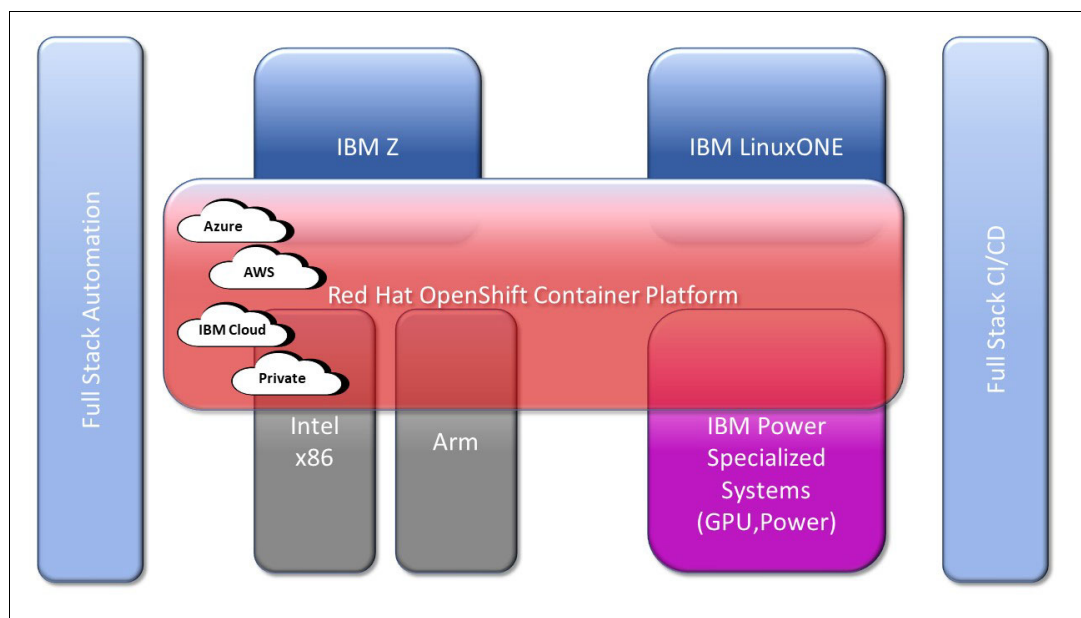


Figure 1-1 Red Hat OpenShift Container Platform

1.2 Architectures and tools that are used in this book

In this section, we introduce some of the architectures and tools that are used in this book.

1.2.1 Microservices

On their journey of digitalization and hybrid cloud service establishment, enterprises are focusing on building smaller individual software components that enable faster time to market for new services and a more flexible and granular scalability. The architecture of these components is often based on microservices concepts.

Microservices is an application architectural style in which an application is composed of many, individually separate, and distinct network-connected components.

An application based on microservices architecture can scale granularity without the need to scale the entire application. This ability enhances the economics of resources and compute capacity.

The implementation of a microservice can vary from a development language and data usability perspective. It also extends the flexibility per microservice to different run times, used data, and communication capabilities between microservices.

Figure 1-2 shows a microservices architecture.

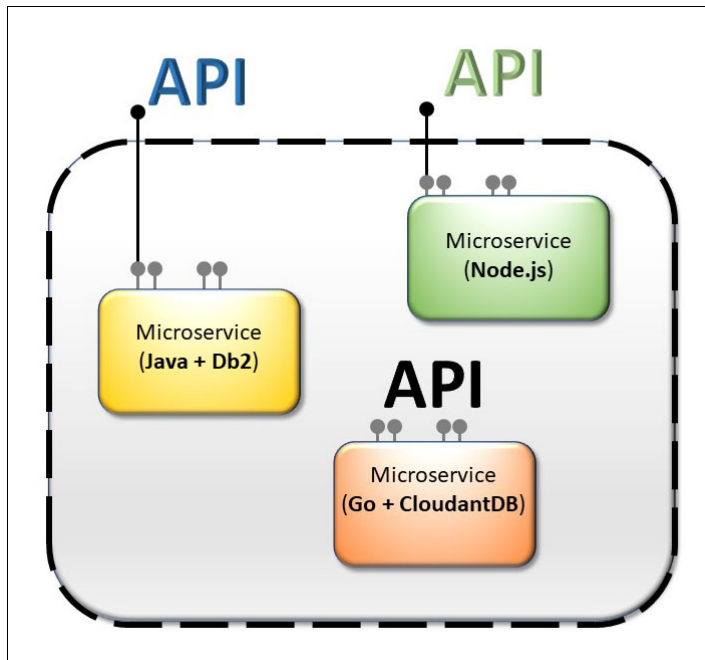


Figure 1-2 Microservices application

Microservices architectures are an important software trend, one that can have profound implications on enterprise IT and the digital transformation of an entire business.

Digital transformation requires organizations to adopt accelerated innovation methods that enable the delivery of new digital services to customers. Monolithic applications might be operationally acceptable, but these applications are not suited for building digital services.

Traditional monolithic architecture and software development methods remain a stumbling block for driving digital transformation.

To efficiently drive digital transformation, organizations are exploring a new software development methods and architecture (called *cloud-based microservices architecture*) whereby IT solutions can be organized around granular business capabilities that can be rapidly assembled to create cloud-based digital experience applications.

A comparison of monolithic architectures and microservices is shown in Figure 1-3.

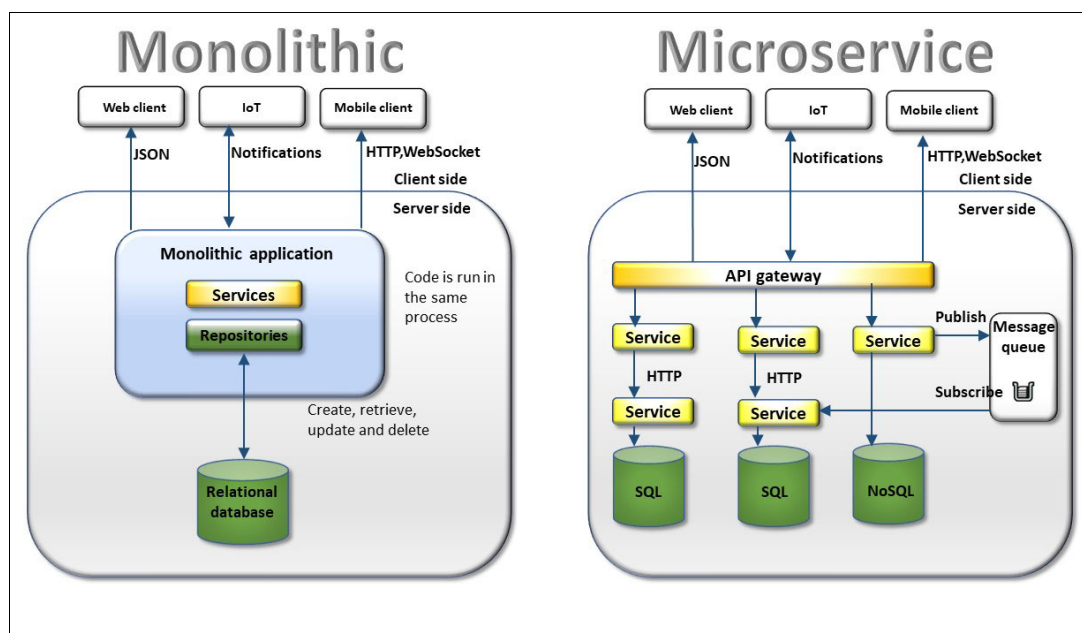


Figure 1-3 Monolithic versus microservices

With LinuxONE unique attributes that are inherited by solutions that are built with Red Hat OpenShift, enterprises can use the combination of hardware features and containerization to create and manage a new generation of portable distributed applications that are composed of discrete, interoperable containers.

By combining Red Hat OpenShift and IBM LinuxONE, we get enterprise-grade automation, orchestration, management, visibility, and control capabilities for the application development and deployment process. In this process, an application can include software components that are hosted in bare metal, VMs, or containers, all fortified by the reliability, security, and scalability of IBM LinuxONE Systems.

1.2.2 Containers

The most efficient way to implement and manage microservices is their implementation as application containers, which can interact with each other through light-weight protocols and use well-defined APIs.

A *container* is a layered approach of a file system that is built starting with a Container Base Image. This image typically is delivered by way of a Linux distribution, such as the Red Hat Universal Base Image (UBI). Container tools are used by the developer to install software components in the container that appear as layers in the container.

By committing a container, it becomes a read-only container image, which is similar to a golden image that is known from virtualized environments. Container images include the information or libraries that are required during execution on the operating system to interface with the container run time and the kernel operating system.

A container abstracts application code from the underlying infrastructure and simplifies version management. It also enables portability across various deployment environments and profits from high application isolation.

Containerized applications can be composed of several container images. A *container image* is a read-only file that often is in a local container image repository or a local or remote container image registry.

Multiple containers can run in a single operating system and scale individually; therefore, they represent an evolution toward flexibility and scalability with fewer resource requirements as compared to a VM.

The concept of containerization is shown in Figure 1-4.

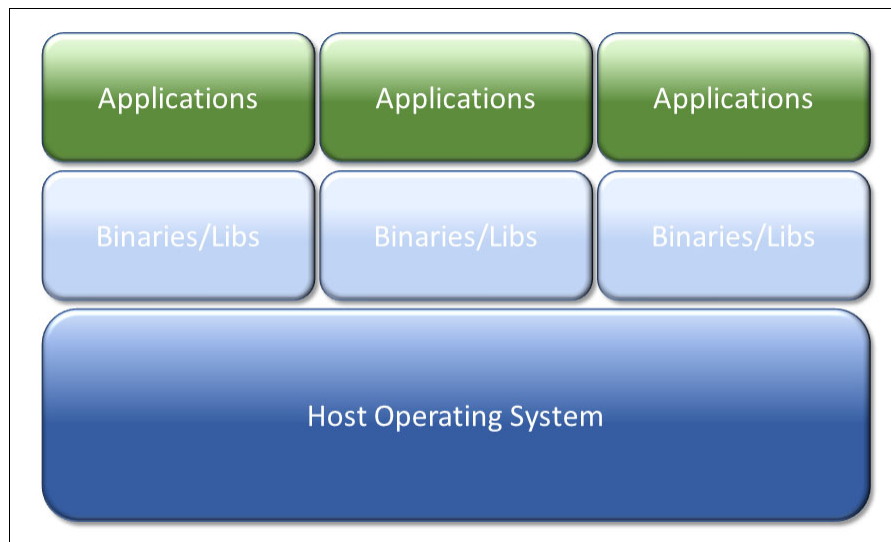


Figure 1-4 Containerization concepts

To build and manage containers, container engines were developed. To run a container, a run time is needed in the operating system where the containers are to run.

Figure 1-5 shows the container run time.

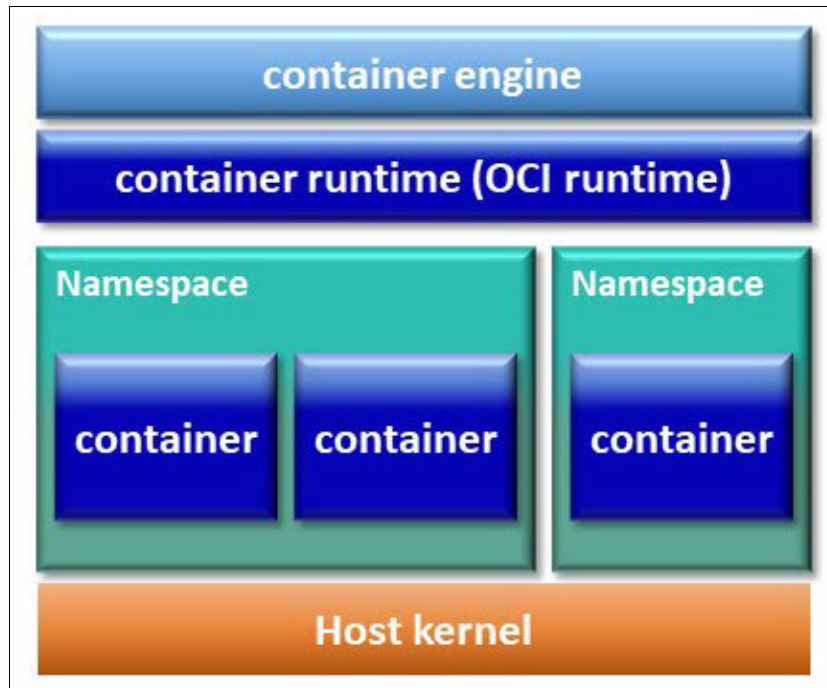


Figure 1-5 Container environment

Starting with Docker, containers became easy to build and use. Shortly after Docker, many different companies developed tools and built an ecosystem around container development and run times.

This diversification led to the [Open Container Initiative](#) (OCI). The OCI defines the standards for container image format and the Container Runtime Interface (CRI).

The OCI is continuously growing, which demonstrates that containerization is evolving rapidly in IT.

1.2.3 Kubernetes

An application can be composed of multiple containers and software solutions and often reach hundreds of containers. Therefore, it became obvious that an effective way of managing containerized applications and solutions is to create specialized tools for container orchestration.

The market for container orchestration tools converged on an Open Source tool (Kubernetes).

Many companies are contributing to the functions and enhancement of Kubernetes and defined the quasi-standard for container orchestrations.

Kubernetes is developed with application high availability by design. It is deployed with three Master Nodes for a high availability quorum and several worker nodes, depending on the container workload.

The container workload runs in pods, which represent the smallest entity in a Kubernetes environment. One or multiple containers can be in a pod, as shown in Figure 1-6.

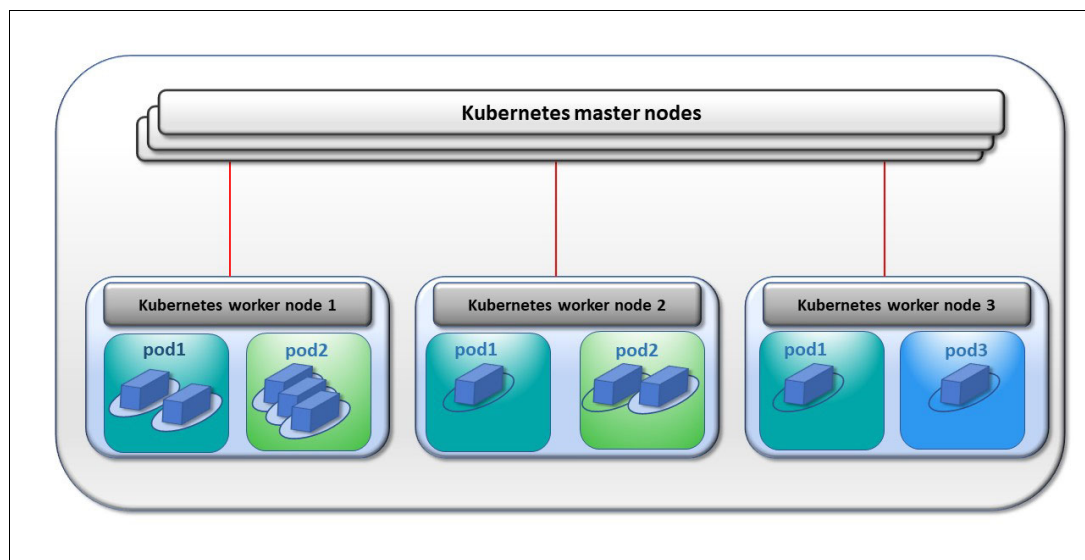


Figure 1-6 A Kubernetes cluster

1.2.4 Containers versus virtualization

In this section, we explain the difference between containers and virtualization.

Containers

Containers make available protected portions of the operating system; that is, they effectively virtualize the operating system. Two containers that are running on the same operating system are unaware that they are sharing resources because each has its own abstracted networking layer, processes, and so on.

A comparison of containerization to virtualization is shown in Figure 1-7.

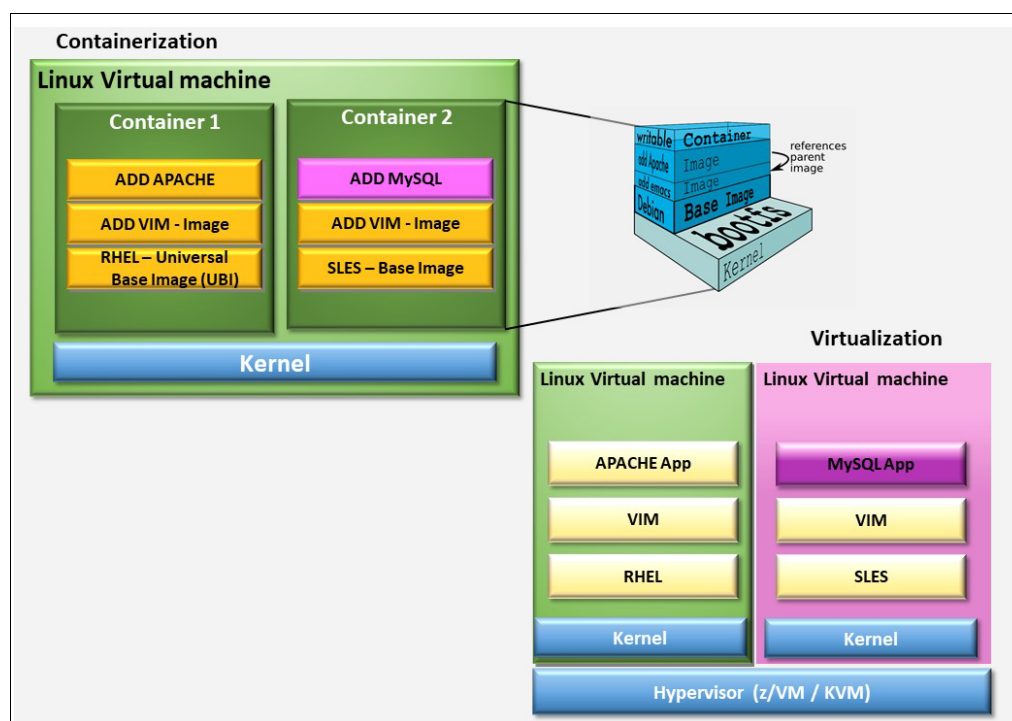


Figure 1-7 Containers versus virtualization

Containers are a lightweight, efficient, and standard way for applications to move between environments and run independently. Everything that is needed (except for the shared operating system on the server) to run the application is packaged inside the container object: code, run time, system tools, libraries, and dependencies.

Containerization is the packaging of software code with only the operating system libraries and dependencies that are required to run the code to create a single lightweight executable (called a *container*) that runs consistently on any infrastructure. More portable and resource-efficient than VMs, containers became the de facto compute units of modern cloud-native applications.

Virtualization

Containers virtualize at the operating system level, whereas hypervisor-based solutions virtualize at the hardware level. Both containers and VMs are virtualization tools.

On the VM side, a hypervisor makes siloed slices of the available hardware. In general, two types of hypervisors are used:

- ▶ Hardware virtualization runs directly on the bare metal of the hardware.
- ▶ Software-based virtualization runs as another layer of software within a guest operating system.

With virtualization, much higher levels of workload density can be realized because many more workloads are running on far fewer servers.

IBM LinuxONE provides the best of both virtualization options, providing extreme performance and security, with EAL5+ (hardware-based virtualization) and EAL4+ (software virtualization).

Virtualized workloads and containerized workloads are compared in Figure 1-8.

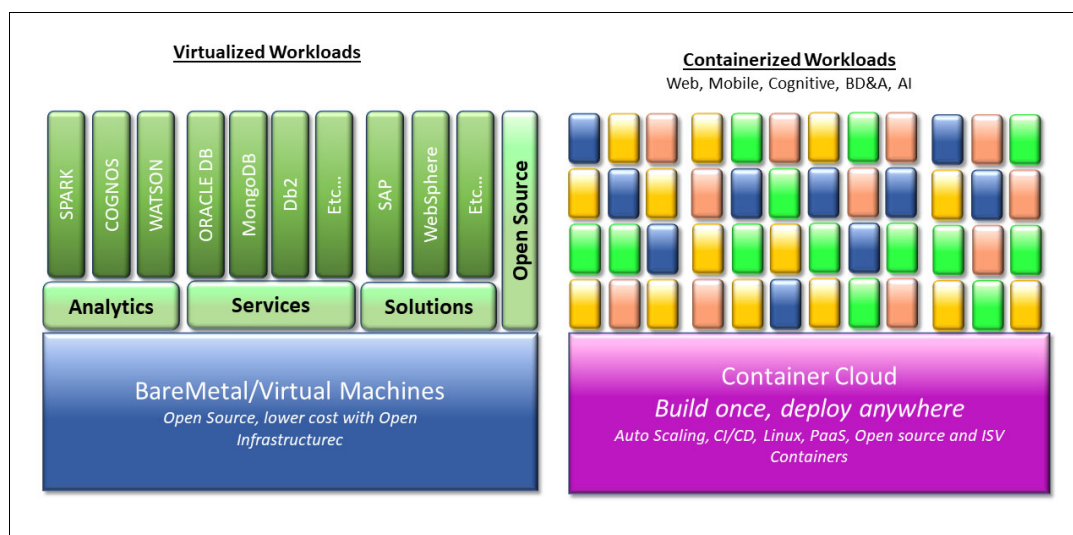


Figure 1-8 Virtualized workloads versus Containerized workloads

1.2.5 Container orchestration frameworks

Containers include packaged, self-contained, ready-to-deploy parts of applications and, if necessary, middleware and business logic (in binaries and libraries) to run the applications. Tools, such as Docker, are built around container engines where containers act as a portable means to package applications. Applications that are packaged in containers results in the need to manage the lifecycle, security, and dependencies between containers in multitier applications. A container orchestration tool can provide these components, their dependencies, and their lifecycle in a streamlined and secure manner.

Container orchestration frameworks, such as Docker Swarm, Kubernetes, Red Hat OpenShift, and Mesos, build upon and extend container run times with more support for deploying and managing a multi-tiered distributed application as a set of containers on a cluster of nodes. Container orchestration frameworks also increasingly are used to run production grade services because they provide the most important PaaS features.

Red Hat OpenShift is an enterprise open source container orchestration platform. It is a software product that includes components of the Kubernetes container management project, but adds productivity and security features that are important to large-scale companies.

The container orchestration framework is shown in Figure 1-9

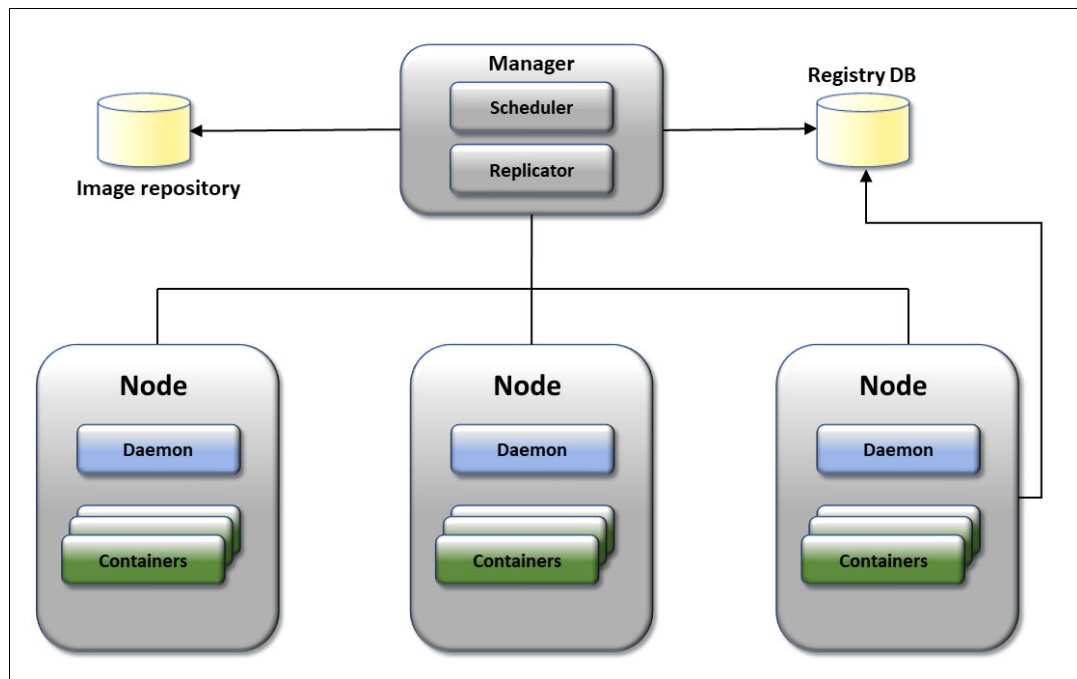


Figure 1-9 Container Orchestration framework

The Red Hat OpenShift framework include the following important features:

- ▶ Cluster architecture and setup
- ▶ CO system customization
- ▶ Container
- ▶ Application configuration and deployment
- ▶ Resource quota management
- ▶ Container QoS management
- ▶ Securing clusters
- ▶ Securing containers
- ▶ Application and cluster management

1.3 Red Hat OpenShift architecture

With containerization came the need for more process-driven enhancements because of continuous operation requirements. Continuous Integration (CI) of new developed functions and features without service interruption was also required, and Continuous Deployment (CD) of new enhancements.

With these CI/CD needs, the lifecycle for an application that is now composed of containers became easier and more efficient to manage. At the same time, the integration of development security and operation (DevSecOps) options for the components of those applications came into focus and the DevSecOps process lead to new tools to be integrated, packaged, and made available.

It is because of this integration that new offerings around containerization evolved and one of the most comprehensive offerings is Red Hat OpenShift (see Figure 1-10). It unifies the establishment of an enterprise grade Kubernetes environment with the surrounding tools for DevSecOps and CI/CD. The various tools and capabilities for development of containerized solutions are integrated with the flexibility of deploying the containers in their own standard Kubernetes environment and enabling the high availability of the applications and lifecycle management of the operating system and the containerized workloads.

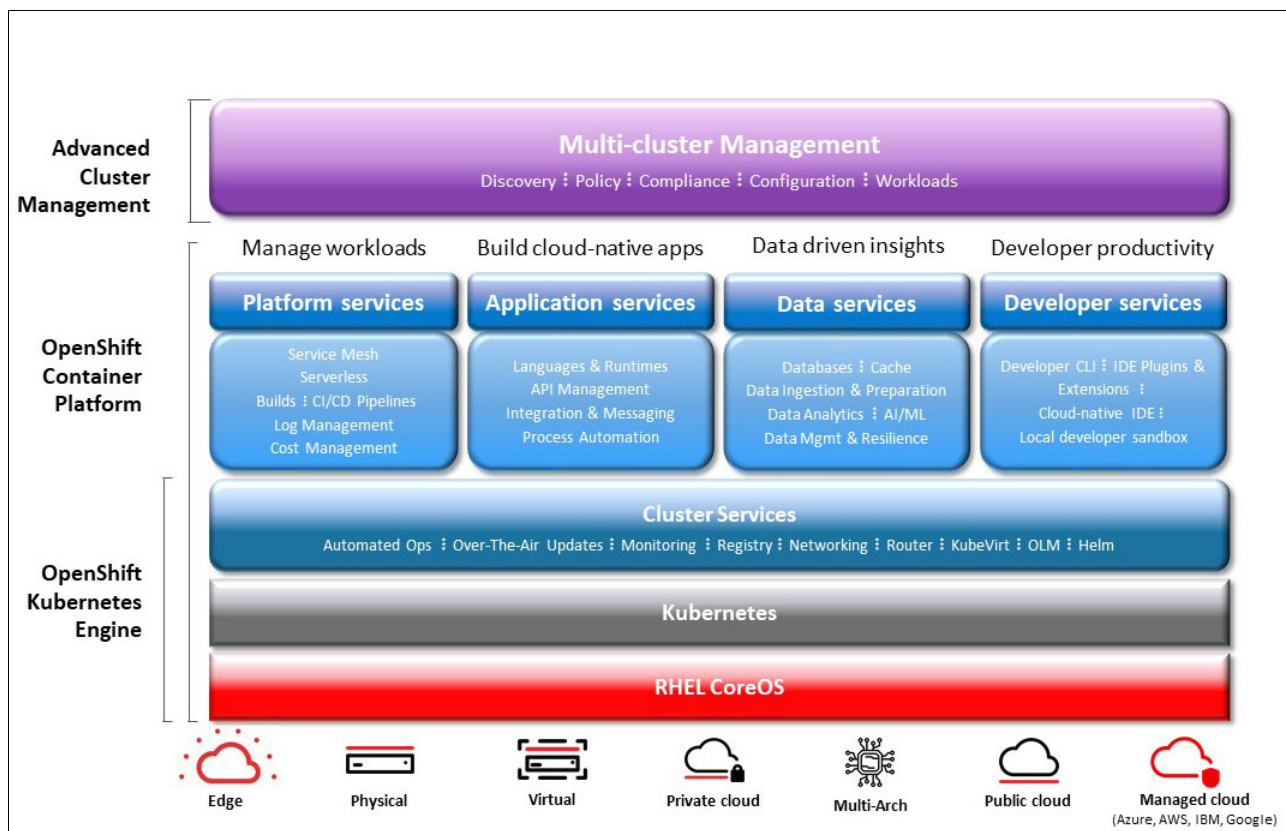


Figure 1-10 Red Hat OpenShift Container Platform architecture

Red Hat OpenShift is the only solution that can run on different hardware architectures. By using multi-cluster management, you can manage hybrid Red Hat OpenShift environments from one place.

Red Hat OpenShift can fully be deployed on IBM Z and IBM LinuxONE in a virtualized environment.

Red Hat OpenShift is a trusted Kubernetes enterprise platform that supports modern, hybrid-cloud application development. It also provides a consistent foundation for applications anywhere across physical, virtual, private, and public clouds.

The core architectural pattern that underlies the Red Hat OpenShift container cluster is based on the Master-Workers architecture. This architecture features a Master node that ensures that running applications are always in their wanted state by scheduling containers to the Worker nodes and by monitoring the actual runtime state of nodes and containers. Master nodes use a distributed data store for storing the configuration state about all deployed containers and services.

Each Red Hat OpenShift cluster features two parts: a control plane and worker nodes. Containers run in the worker nodes, each of which has its own operating system. The control plane maintains the cluster's overall state (such as what applications are running and which container images are used), while worker nodes do the computing work. An overview of the high-level Red Hat OpenShift architecture is shown in Figure 1-11.

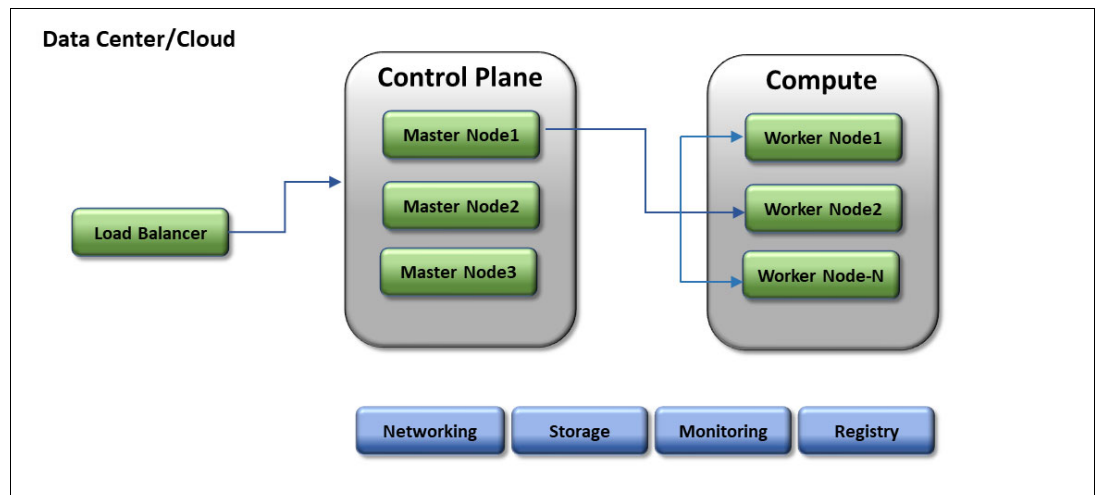


Figure 1-11 High-level Red Hat OpenShift architecture

Red Hat CoreOS

To cater for the low overhead requirements that provide faster spin-up time, Red Hat developed a container-optimized OS build: Red Hat CoreOS. Red Hat CoreOS contains minimalistic subsystems for containers to run.

The underlying operating system consists primarily of Red Hat Enterprise Linux components. The same quality, security, and control measures that support Red Hat Enterprise Linux also support Red Hat CoreOS. Although it contains Red Hat Enterprise Linux components, Red Hat CoreOS is managed more tightly than a default Red Hat Enterprise Linux installation. Management is performed remotely from the Red Hat OpenShift Container Platform cluster.

Red Hat CoreOS includes the following key features:

- ▶ Based on Red Hat Enterprise Linux
- ▶ Controlled immutability
- ▶ CRI-O container run time
- ▶ Set of container tools
- ▶ Upgrades to rpm-ostree

- ▶ bootupd firmware and bootloader updater
- ▶ Updated through the Machine Config Operator

Note: For more information about Red Hat CoreOS, see this [web page](#).

1.3.1 Red Hat OpenShift components

Red Hat OpenShift includes everything that you need for hybrid cloud, such as a container run time, networking, monitoring, container registry, authentication, and authorization. Red Hat OpenShift consists of the following important components, and each component has its own responsibilities:

- ▶ Control plane
- ▶ Worker nodes
- ▶ Registry
- ▶ Storage controller

Control plane

The control plane, which is composed of master nodes, manages the Red Hat OpenShift Container Platform cluster. The control plane machines manage workloads on the compute machines, which are also known as *worker machines*. The cluster manages all upgrades to the machines by the actions of the Cluster Version Operator, the Machine Config Operator, and a set of individual Operators.

The Control Plane with the Controller Manager, Scheduler, API Server, and etcd are shown in Figure 1-12.

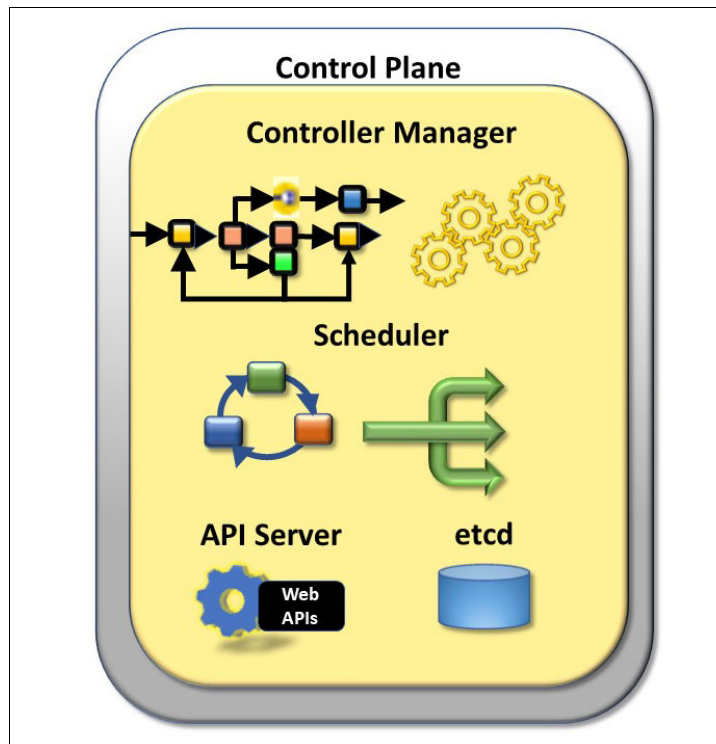


Figure 1-12 Control Plane Master nodes

Controller manager

The controller manager implements governance across the cluster and runs a set of controllers for the running cluster. The key controllers are replication controller, endpoint controller, namespace controller, and service account controller. The controller manager runs different kind of controllers to handle nodes, the endpoint, and so on.

API server

The API server acts as the front end to the cluster. All external communications to the cluster are by way of the API Server. The API Server is the only component that directly communicates with the distributed storage component, etcd.

The API Server provides the following core functions:

- ▶ Serves the Kubernetes API that is used cluster-internally by the worker nodes and externally by kubectl
- ▶ Proxies cluster components, such as the Kubernetes UI
- ▶ Allows the manipulation of the state of objects; for example, pods and services
- ▶ Persists the state of objects in a distributed storage (etcd)

etcd

This cluster state database provides a consistent and highly available key value store, which is used as Kubernetes' backing store for all cluster data. It is a high availability key value store that can be distributed among multiple nodes. It is accessible only by Kubernetes API server because it might include sensitive information.

Scheduler

The Scheduler primarily schedules activities to the worker nodes based on events that are occurring on the etcd. It also holds the nodes resources plan to determine the suitable action for the triggered event. For example, the Scheduler determines which worker node is to host a newly scheduled POD.

Storage

Kubernetes creates permanent storage mechanisms for containers that are based on Kubernetes persistent volumes (PV) and refers to any resource that applies to the entire cluster that allows users to access data far beyond their pod's total lifespan.

Red Hat OpenShift Container Platform uses the Kubernetes PV framework to allow cluster administrators to provision persistent storage for a cluster. Developers can use persistent volume claims (PVCs) to request PV resources without having specific knowledge of the underlying storage infrastructure.

Red Hat OpenShift deployment on IBM LinuxONE supports the following storage options:

- ▶ Network File System (NFS)
- ▶ IBM Spectrum® Scale
- ▶ Red Hat OpenShift Container Storage (OCS)

Note: For more information about the OpenShift-based storage deployment and options, see this [web page](#).

1.3.2 Pods and ReplicaSets

In this section, we describe pods and ReplicaSets.

Pods

A *pod* is a collection of containers that are running on the Worker node. A pod is the smallest and simplest Kubernetes object that can be defined, deployed, and managed in a cluster. Each pod is allocated its own internal IP address and shares a network and hostname with PID Namespaces.

Kubernetes pods and their containers are shown in Figure 1-13.

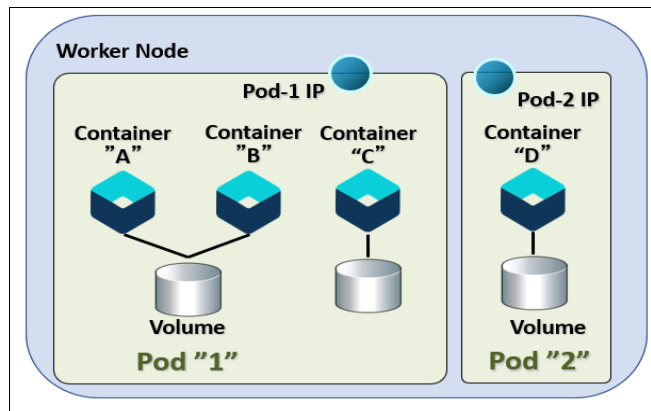


Figure 1-13 Kubernetes pods

ReplicaSets

ReplicaSets are used maintain a stable set of replica pods that are running at any time. A ReplicaSet ensures and guarantees that a specified number of pod replicas are running at any time (see Figure 1-14).

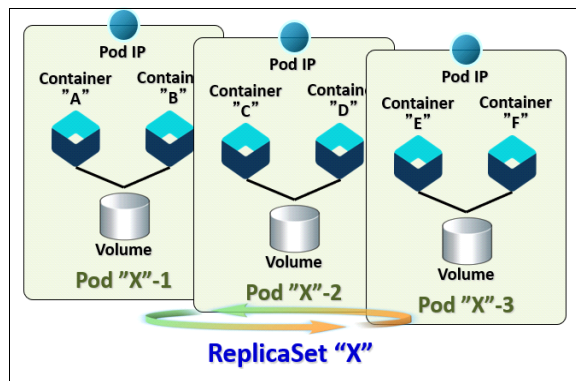


Figure 1-14 Kubernetes Replicasets

Service

A small service is available in each node, which is responsible for relaying information to and from the control plane service. It interacts with the etcd store to read configuration details and values.

The service communicates with the master component to receive commands and work. The kubelet process then assumes responsibility for maintaining the state of work and the node server. It also manages network rules, port forwarding, and so on.

The Kubernetes service is shown in Figure 1-15.

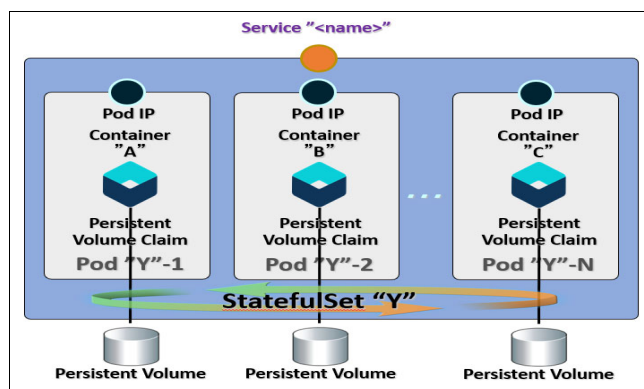


Figure 1-15 Kubernetes service

Worker nodes

Workers nodes are the compute nodes in the Red Hat OpenShift cluster where all application containers are deployed to run. Worker nodes advertise their resources and resource utilization so that the scheduler can allocate containers and pods to worker nodes and maintain a reasonable workload distribution. The Kubernetes CRI-O kubelet service runs on each worker node. This service receives container deployment requests and ensures that they are instantiated and put into operation.

Registry

Red Hat OpenShift inherits the Kubernetes registry. It provides an immediate solution for users to manage the images that run their workloads, and runs on top of the cluster infrastructure. The registry is typically used as a publication target for images that are built on the cluster, and as a source of images for workloads that are running on the cluster.

1.4 Red Hat OpenShift Container Platform introduction

The deployment of Red Hat OpenShift Container Platform on IBM Z or IBM LinuxONE provides the following benefits:

- ▶ A clustered implementation approach that is aligned with Kubernetes.
- ▶ A virtualized environment with z/VM or KVM on IBM Z.
- ▶ The ability to use IBM Z attachments and capabilities.
- ▶ Vertical scalability versus horizontal scalability.
- ▶ Infrastructure by way of hypervisors, networks, and so on.

Depending on what your enterprise use case is, you can choose from different topologies and deployment options for Red Hat OpenShift Container Platform, including the following examples:

- ▶ Deploying Red Hat OpenShift Container Platform as the only cloud environment on IBM Z and IBM LinuxONE.
- ▶ Deploying Red Hat OpenShift Container Platform with another, noncontainerized workload that you also run as a back-end in a Linux environment on IBM Z and IBM LinuxONE hardware.
- ▶ Deploying Red Hat OpenShift Container Platform in colocation with a traditional operating and transactional environments and services, such as IBM z/OS, IBM z/VSE®, or z/TPF on IBM Z hardware.

1.4.1 Red Hat OpenShift Container Platform colocated with traditional workloads

Red Hat OpenShift Container Platform is available on IBM Z and IBM LinuxONE, and takes advantage of the underlying capabilities of each, including the following examples:

- ▶ Ability to scale to thousands of Linux guests and millions of containers.
- ▶ Nondisruptively grow vertically *and* horizontally to provide a platform with large scalability.
- ▶ Enables cloud native applications to easily integrate with data and applications on these platforms, which reduces latency by avoiding network delays.

1.4.2 IBM Cloud Paks for Data Overview

IBM Cloud Paks for Data represents a new way of software delivery for a Red Hat OpenShift environment. IBM is grouping and packaging multiple software solutions in different Cloud Paks. They are delivered in this format to enable the capability for easy deployment, integrated monitoring, and provide a common, consistent, and integrated experience for cloud-native workloads.

1.5 Use cases overview

To demonstrate throughout this book how you can build an Red Hat OpenShift environment on IBM Z or IBM LinuxONE, we provide the following use cases:

- ▶ “Use Case: Cloud Pak for Data platform that is running containerized Db2 service” on page 77
- ▶ “Use Case: A Red Hat OpenShift environment that is colocated with z/OS transactional services and DB2 data” on page 89



Architectural overview

In this chapter, we introduce the architectural topologies and components that are relevant when building a Red Hat OpenShift Container Platform on IBM Z and IBM LinuxONE.

We start with the planning considerations for a Red Hat OpenShift Container Platform environment. Then, we describe the essential characteristics and the options to build and operate a Red Hat OpenShift Container Platform environment on IBM Z and IBM LinuxONE.

This chapter includes the following topics:

- ▶ 2.1, “Topology planning” on page 22
- ▶ 2.2, “Environment considerations” on page 23
- ▶ 2.3, “Sizing” on page 36
- ▶ 2.4, “Continuous availability for applications” on page 46
- ▶ 2.5, “Planning for HA and DR infrastructure” on page 49

2.1 Topology planning

Red Hat OpenShift Container Platform is designed to facilitate the orchestration, deployment, and management of running containerized applications. Therefore, the key architectural decisions for your deployment are to consider how you can ensure that your applications adhere to best practices of a cloud native container workload and fulfill your business requirements.

Several characteristics guide you through the planning and decisions to build a Red Hat OpenShift Container Platform environment on IBM Z or IBM LinuxONE. The topology of a Red Hat OpenShift Container Platform environment on IBM Z and IBM LinuxONE depends on the use cases that you plan to implement.

You can choose from the following topologies and deployment options for Red Hat OpenShift Container Platform:

- ▶ Deploying Red Hat OpenShift Container Platform as a stand-alone cloud in a box environment on IBM Z and IBM LinuxONE.
- ▶ Deploying Red Hat OpenShift Container Platform in a hybrid cloud solution with a noncontainerized workload that you run as back-end data or a service in a Linux on IBM Z and IBM LinuxONE hardware.
- ▶ Deploying Red Hat OpenShift Container Platform in a hybrid colocation scenario with a traditional operating environment to access data and transactional services from z/OS, z/VSE, or z/TPF on IBM Z hardware.

You can deploy hybrid workloads and build hybrid multi-cloud services on the same platform. As a result, you can take advantage of the flexibility of the cloud services that are deployed in the Red Hat OpenShift Container platform side by side with the secure and established core systems, as needed.

2.2 Environment considerations

When planning to deploy a Red Hat OpenShift Container Platform cluster, it is important to consider the IBM Z and IBM LinuxONE architecture and decide whether you are planning for a Proof of Concept (PoC) or Proof of Technology (PoT) environment or a production-like environment for a specific workload.

For example, Figure 2-1 shows an architecture for a Red Hat OpenShift Container Platform with hybrid workloads.

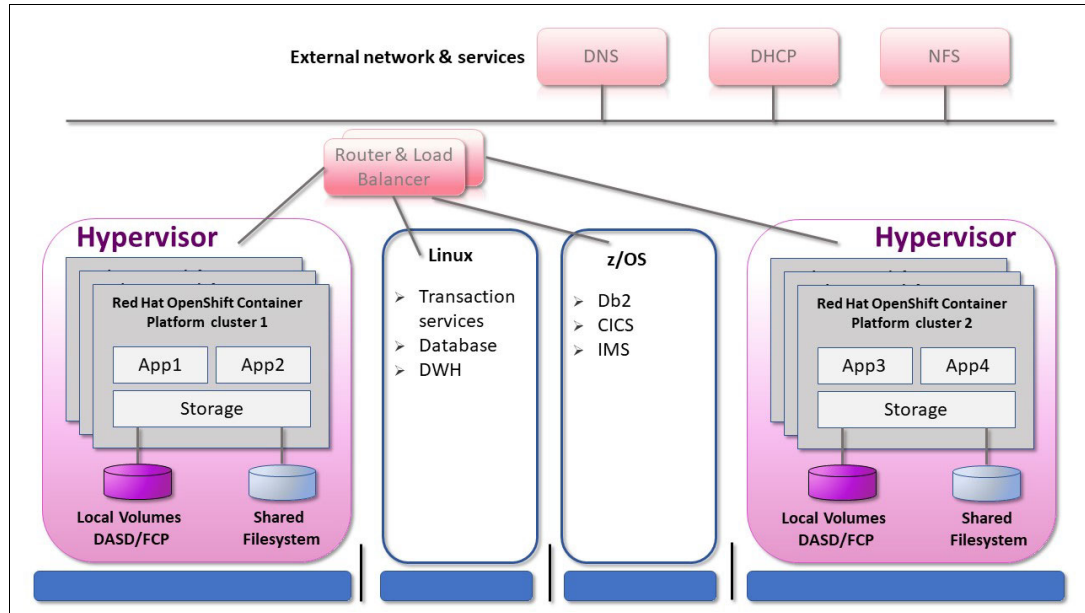


Figure 2-1 Red Hat OpenShift Container Platform hybrid workloads

In this section, we discuss the components that you must consider when planning to deploy a Red Hat OpenShift Container Platform environment.

2.2.1 Virtualization options

A Red Hat OpenShift Container Platform environment on IBM Z and LinuxONE is deployed in a virtualized environment. The following virtualized environment options are available:

- ▶ z/VM 7.2 or later
- ▶ RHEL KVM 8.3

Hypervisors can fully use the IBM Z and IBM LinuxONE features of scalability, security, and reliability. They also can virtualize memory, computing capacity, network, and storage attachments to shift resources where they are needed most or based on predefined priorities.

In addition, they can handle effectively the workload in environments with overcommitment of compute, power, and memory.

2.2.2 Hardware topology options

For a Red Hat OpenShift Container Platform environment on IBM Z and IBM LinuxONE, the machine must be partitioned first, which represents a hardware level virtualization with which you can share resources across those partitions (such as processor or core capacity, I/O and network) but remain multi-tenant because of the highest isolation certification in the industry with an Evaluation Assurance Level 5+ (EAL5+).

The resulting logical partitions (LPARs) are the basis for building different workload environments that can run side by side on the same hardware machine. Each cluster can exist on a single or multiple LPARs, which can be on one or multiple physical machines.

Therefore, the next architectural decision is to define the number of LPARs and physical IBM Z or LinuxONE machines where your cluster is located. In each LPAR, the hypervisor enables the capability to run the Red Hat OpenShift Container Platform Control Plane nodes (formerly known as *Master nodes*), the Compute nodes (formerly known as *Worker nodes*) and Infrastructure nodes.

With the introduction of a crypto accelerator chip (CPACF) on every core, billions of encrypted web requests can be handled with speed daily.

2.2.3 Red Hat OpenShift Container Platform cluster topology

The concept of an Red Hat OpenShift Container Platform cluster is to provide continuous operation and high availability (HA) to software services; therefore, the Red Hat OpenShift Container Platform cluster must be deployed with three control plane nodes and several Compute nodes, depending on the planned workloads and container applications. A minimum of two Compute nodes must be used (Compute nodes can be added to the cluster as required).

Day 2 operations are the “housekeeping” stage of a software’s life cycle. That is, the “care and feeding” of the software, and maintaining the overall stability and health of your software in production. These infrastructure nodes contain the cluster monitoring components and container registry access and free up resources in the Compute nodes for the main workload.

You can install a Red Hat OpenShift Container Platform cluster with five nodes, as shown in Figure 2-2. With Red Hat OpenShift Container Platform version 4.8, you can use a 3-node cluster where the Control Plane and Compute nodes are converged into one mixed node. This configuration can be effective for a fast start or development and test environments.

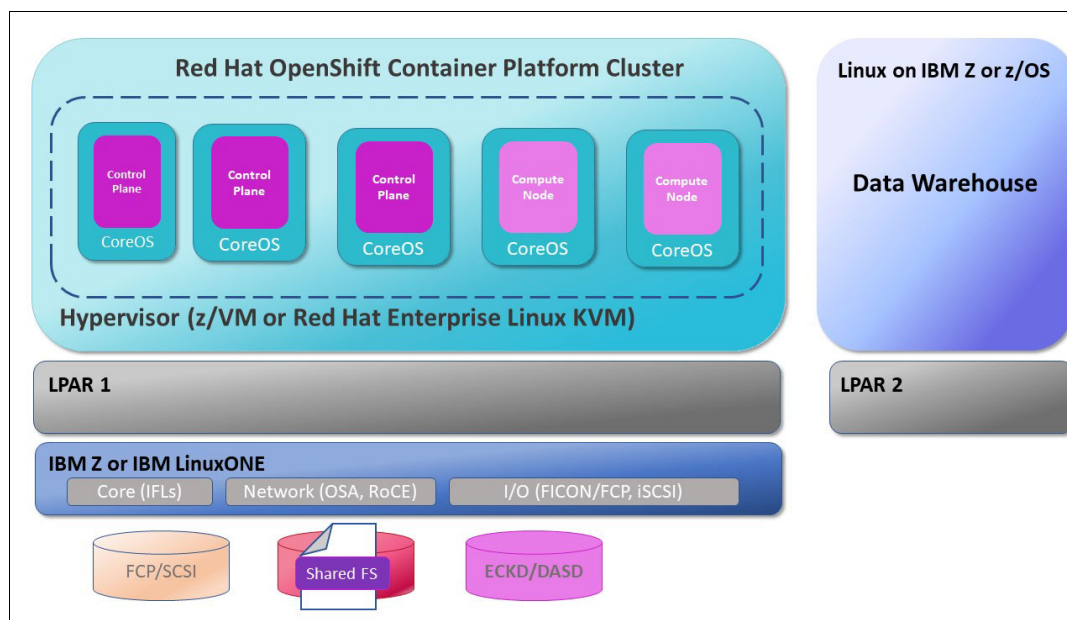


Figure 2-2 Red Hat OpenShift Container Platform-single LPAR cluster

For production environments, plan for at least three LPAR clusters for HA. The LPARs can be spread across multiple hardware machines as well.

2.2.4 Network topology

A Red Hat OpenShift Container Platform cluster can coexist with two separate networks:

- ▶ The internal network that is responsible for the Red Hat OpenShift Container Platform internode communication
- ▶ The second network, which is the external network for communication with external services

For the Red Hat OpenShift Container Platform internal network, the nodes within the same hypervisor and LPAR can use virtual switch (vSWITCH) capabilities to communicate with shared ports on the network cards.

Red Hat OpenShift Container Platform supports the following network interface cards (NICs):

- ▶ Direct-attached OSA Express
- ▶ RoCE

Although OSA is considered the native network card for an IBM Z system, RoCE cards can fit some use cases that demand lower latency and better throughput. Table 2-1 lists the main characteristics of each NIC.

Table 2-1 Main characteristics of OSA and RoCE cards

OSA Express	RoCE
Base-T and Fiber ¹	Fiber only
All Operating Systems on IBM Z	Linux only
Layer 2 (default, recommended) and Layer 3	Layer 3 only
Does not support SMC-R	Supports SMC-R ²
Supported by virtualization (vSWITCH and Open vSWITCH)	Not supported by vSWITCH and Open vSWITCH
<p>1 For 10 and 25 GB fiber only.</p> <p>2 Shared Memory Communication (SMC) is not supported by Red Hat OpenShift Container Platform.</p>	

With the OSA and RoCE network cards, you have two options for networking. With the internal HiperSockets network, you have another option with network-in-the-box capability between LPARs. Depending on the use case and hypervisor, you can decide to use HiperSockets or network cards.

Virtualization network in the hypervisor layer

To achieve HA in the network layer, a minimum of two cards is required; however, OSA cards can take advantage of network virtualization.

z/VM

The z/VM operating system provides a virtual switch (vSWITCH) to use a virtual LAN to the guests. IBM Think® of a vSWITCH as a physical switch, but virtualized by the system.

Each guest (node) sees the interfaces as a regular OSA and the vSWITCH connects to an external LAN through one or more OSAs. It also has several capabilities, similar to physical switches, such as VLAN and link aggregation. It also supports Layer 2 or Layer 3 of the OSI model.

The following benefits can be achieved by using a vSWITCH:

- Scalability: Easier and cheaper to scale because no extra physical ports in a switch are required.
- Simplicity: Easier to maintain and manage.
- HA: A VSWITCH can support up to eight NICs.
- Resiliency: NICs can be attached to different physical switches.

KVM

By using KVM as a host, the network must be set as a bridged networking in libvirt or MacVTap. By assigning a MAC address to the guests interface, both provide directly network connectivity to the guests by an external network.

For more information about a comparison between the possible network modes and performance data, see this IBM Documentation [web page](#).

Software defined networking

Software-defined networking (SDN) is an abstraction of the network layer and enables a network to be controlled and programmed as an application programming interface (API) that provides a virtual network to the cluster, which enables the communication between the pods and allows an operator to define its level of isolation and control by using APIs.

A container network interface (CNI) is a set of specifications for writing network plug-ins to containers on Linux. Any provider that wants to create their own plug-in must follow these specifications.

In the Kubernetes world, many plug-ins are available, each with its own characteristics. However, for a Red Hat OpenShift Cluster Platform on Z system, only two plug-ins are made available:

- ▶ Red Hat OpenShift SDN (default)
- ▶ OVN-Kubernetes

Table 2-2 lists the main features that are supported by each plug-in.

Table 2-2 CNI plug-in feature support

Feature	Red Hat OpenShift SDN	OVN-Kubernetes
Egress IPs	Supported	Supported
Egress firewall	Supported	Supported
Egress router	Supported	Supported
IPsec encryption	Not supported	Supported
IPv6	Not supported	Supported
Kubernetes network policy	Partially supported	Supported
Kubernetes network policy logs	Not supported	Supported
Multicast	Supported	Supported

With ez/VM or KVM hypervisors, you can also use direct attachments to the network cards without virtual switches. However, you must consider that this option is not as flexible for changing the topology of the cluster by adding Red Hat OpenShift Container Platform nodes in the same hypervisor.

With z/VM as the hypervisor, you can use OSA cards and a z/VM vSWITCH (see Figure 2-3) as a flexible way to establish an internal network for the cluster and use the card for external communications.

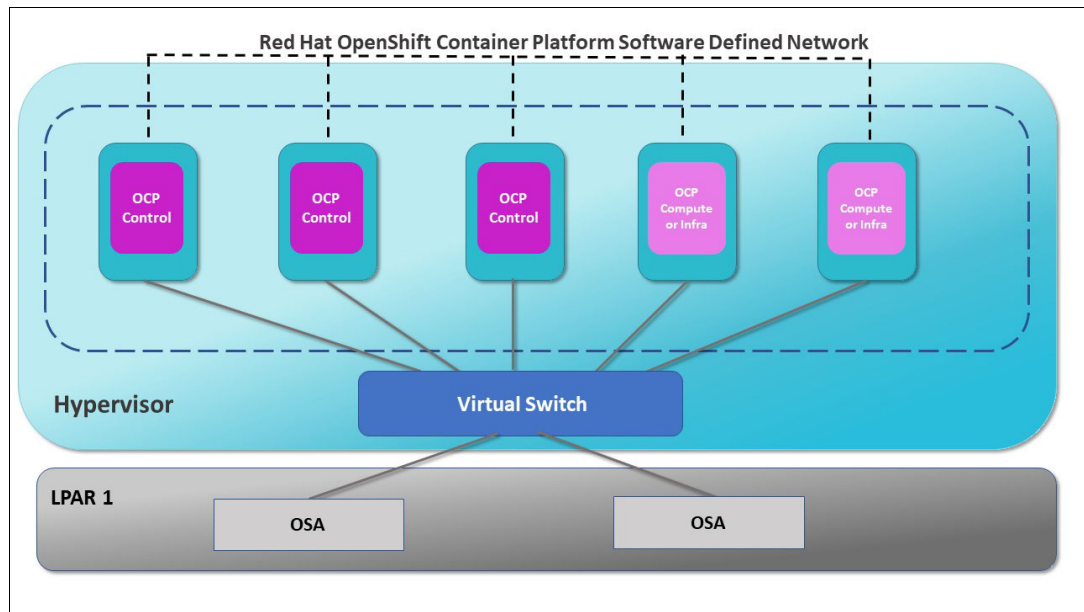


Figure 2-3 Network vSwitch

With KVM as the hypervisor, you can use RoCE cards or OSA cards (see Figure 2-4) and use MacVTap to build an internal network.

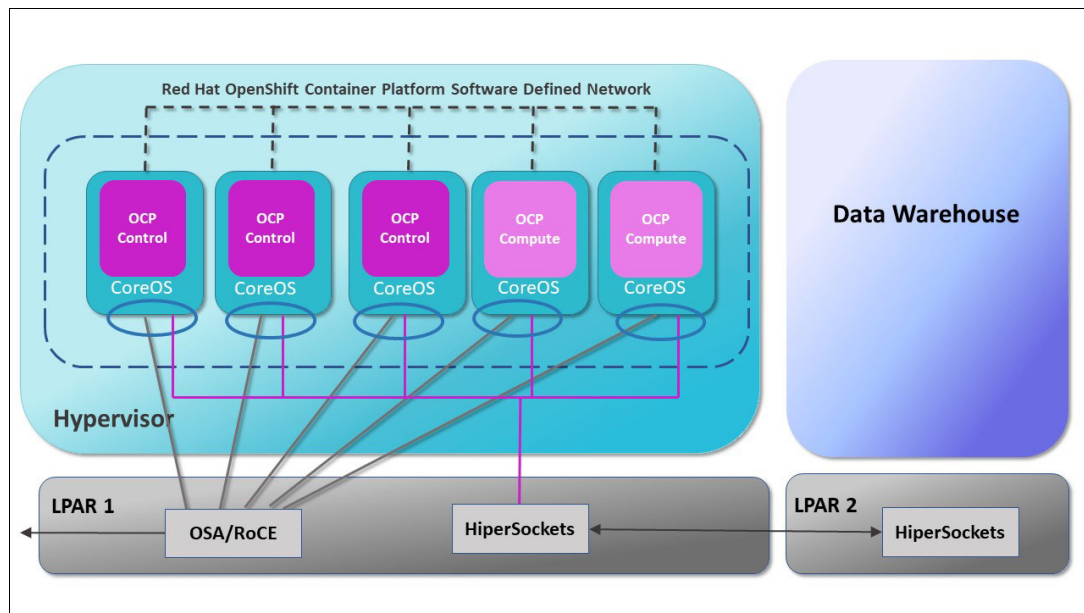


Figure 2-4 KVM topology

In the case of a colocation implementation, you can take advantage of multiple network interfaces by way of the Red Hat OpenShift Container Platform node. You also can separate the networks (for example, the network to an internal data or transactional service) from external communication.

Each Red Hat OpenShift Container Platform node requires at least one network interface, but can use multiple network interfaces for traffic isolation.

Depending on the use case with more or less external network traffic, the topology can be aligned to reach the best overall performance.

HiperSockets

HiperSockets is a hardware feature that provides high-speed LPAR-to-LPAR communications within the same server or Central Processor Complex (CPC) through the system memory of the processor by leveraging an internal LAN.

Depending on the workload and topology, it brings benefits in terms of performance as the communication goes at memory speed by bypassing all the network overhead and delays. A good example is an z/OS workload that uses a SpringBoot API that is running in the Red Hat OpenShift Cluster Platform.

However, HiperSockets has no external components and does not communicate outside of the CPC. Also, HiperSockets is not configured directly in the Red Hat OpenShift Container Platform.

To communicate outside of the CPC, some form of gateway must be in place for Red Hat OpenShift Container Platform. The following approaches can be used to achieve that goal:

- Bridging a HiperSockets LAN with a z/VM vSWITCH

If the cluster is deployed to a z/VM LPAR, vSWITCH can be used to bridge HiperSockets communication, which makes it transparent to the guests. The vSWITCH identifies itself to the host as part of a HiperSockets LAN or outside the CPC and sends out traffic without the guests knowing about the HiperSockets.

For more information, see this IBM Documentation [web page](#).

- HiperSockets gateway

To directly connect a guest to the HiperSockets, a gateway between the cluster, HiperSockets LAN, and the cluster running Red Hat Enterprise Linux is required. This gateway is needed because no mechanism is available to Red Hat OpenShift Container Platform to set up more HiperSockets interfaces in the CoreOS guests in the cluster.

Such an approach also demands the load balancer be in front of the cluster to be deployed as software inside the IBM Z system LPAR.

2.2.5 Storage topology

To install a Red Hat OpenShift Container Platform cluster, you can use your storage servers that are attached to IBM Z or LinuxONE. The options are to use the IBM FICON® attached ECKD type storage servers or the FCP/SCSI attached SCSI devices.

You also can combine the storage topology as required. The storage devices must be defined in your hypervisor and allocated to the defined guests that are to be used to install the Red Hat OpenShift nodes. The type of storage is known as *hypervisor storage* (see Figure 2-5).

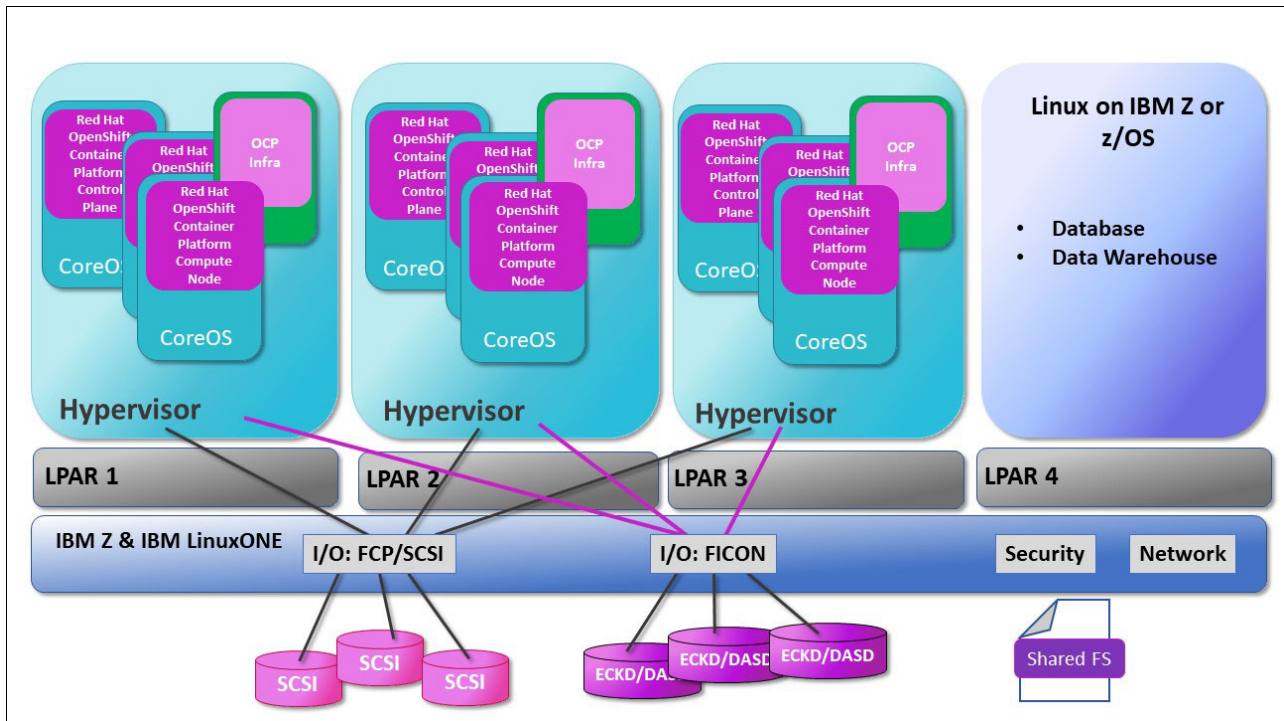


Figure 2-5 Hypervisor storage

After installing Red Hat OpenShift Container Platform and starting the compute nodes, you can deploy container workloads. Containers must use persistent storage so that the data used remains persistent. This type of storage is known as *container storage* (see Figure 2-6).

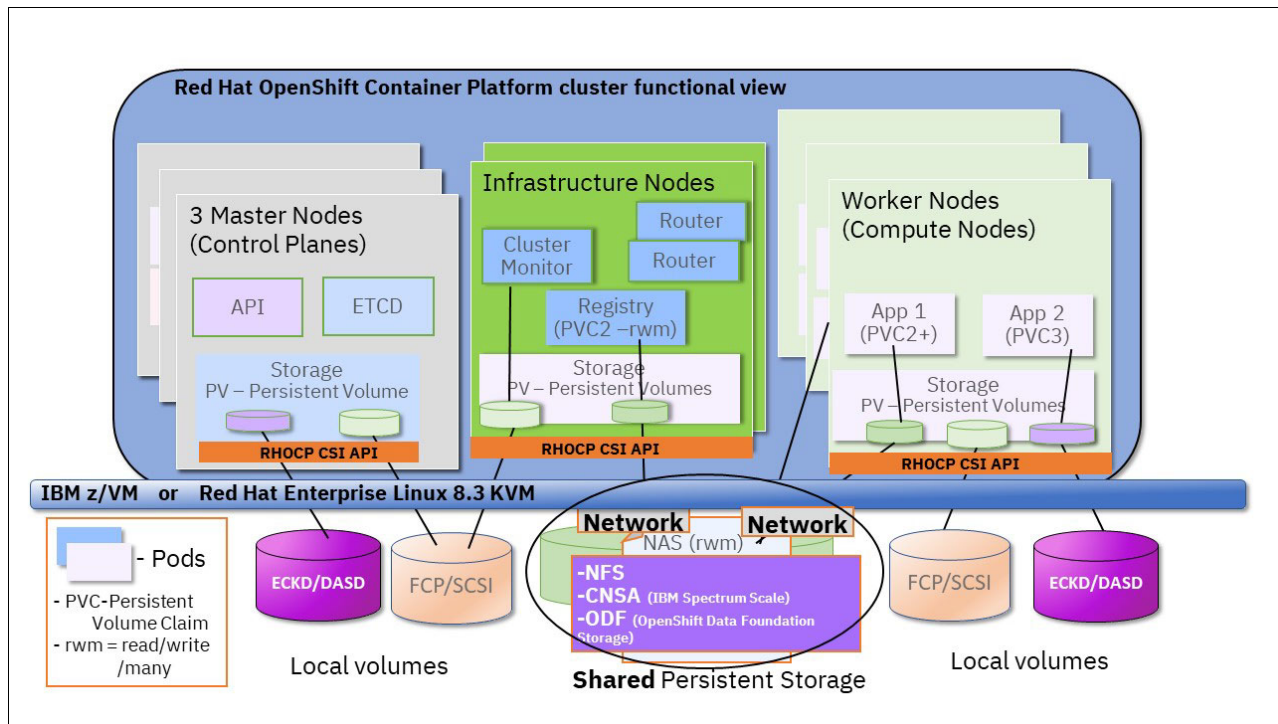


Figure 2-6 Container storage

For storage that hosts container data, you can use locally attached storage or software defined storage to share the storage among different nodes and container pods.

The following options are available for storage:

- ▶ NFS storage: Effective for development and tests, not recommended for production.
- ▶ IBM Spectrum Scale Container Native Attached Storage (CNSA): An effective IBM software-defined storage solution (see Figure 2-7).

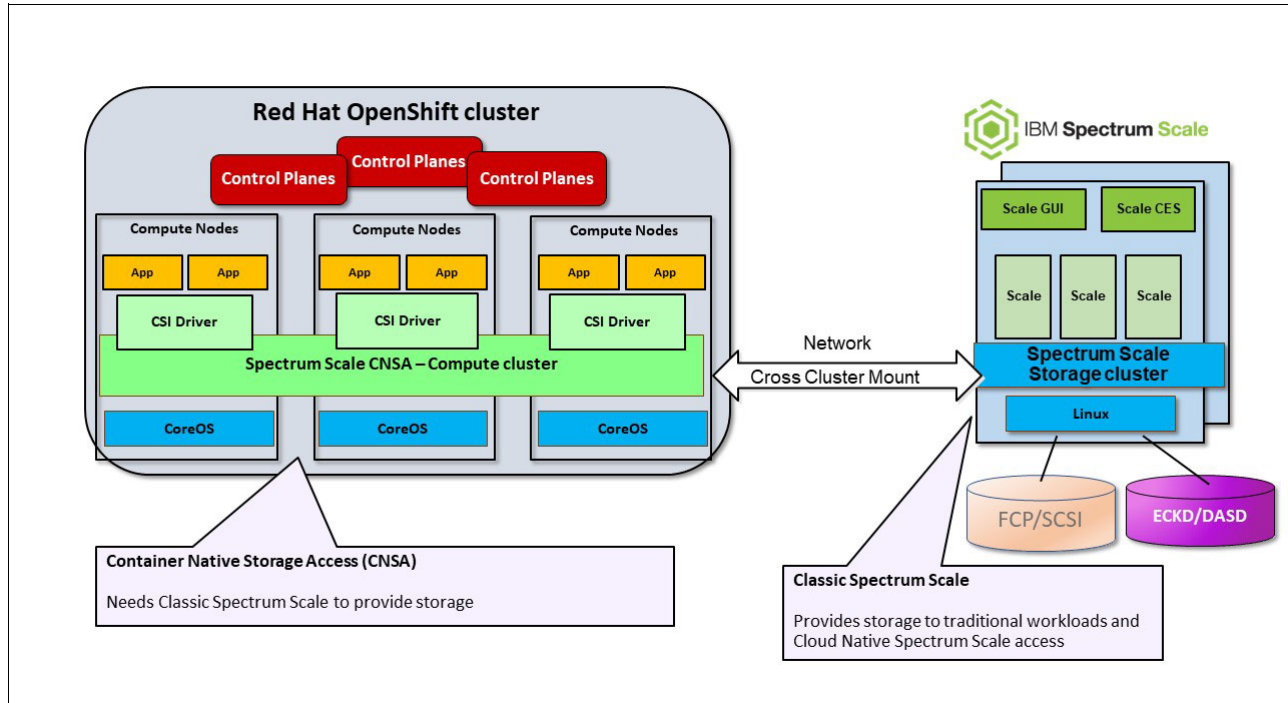


Figure 2-7 IBM Spectrum Scale Container Native Attached Storage

- Red Hat OpenShift Data Foundation (ODF) storage (formerly known as *Red Hat OpenShift Container Storage* [OCS]): The newest Red Hat software-defined storage solution for Red Hat OpenShift (see Figure 2-8).

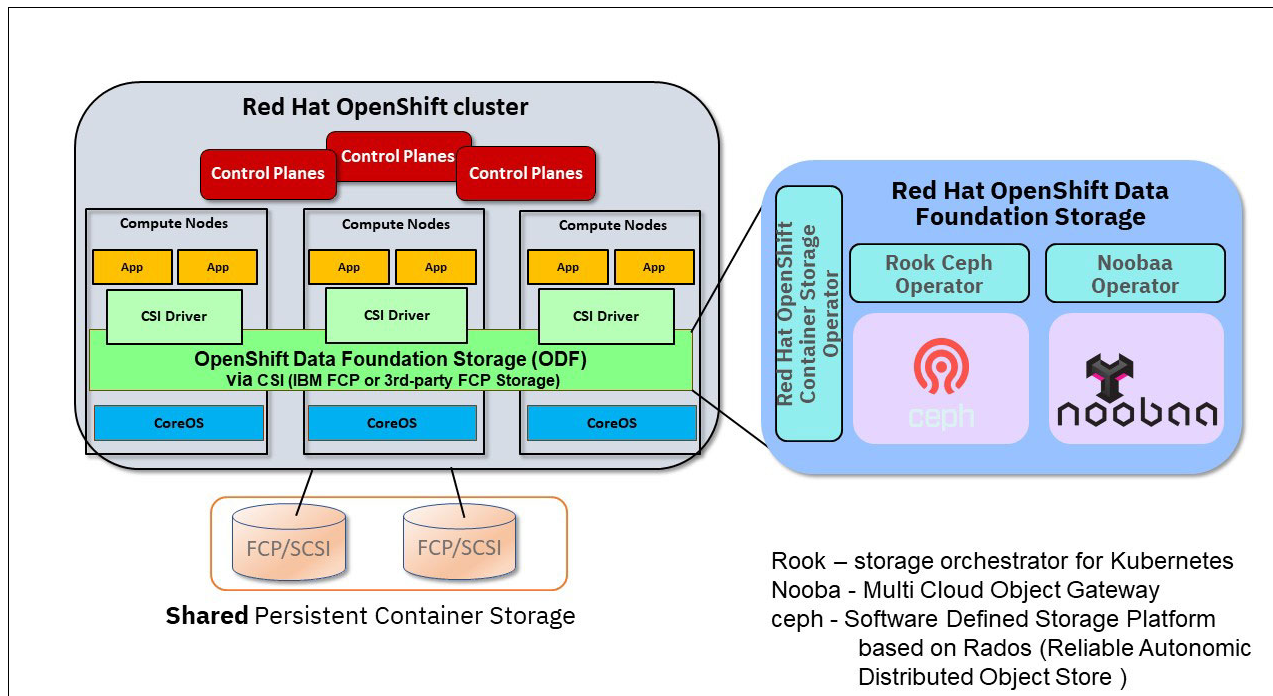


Figure 2-8 Red Hat OpenShift Data Foundation storage

2.2.6 Capacity cores and memory

The processing capacity (IFL cores) and main memory (RAM) that are needed to run your containerized workloads depends on the environment type and scope. Start with three cores per cluster, but you might need to examine your software stack and its requirements to determine whether you need more.

The advantage of running Red Hat OpenShift Container Platform on IBM Z and LinuxONE is that it enables the sharing of the processing capacity; therefore, it is economical. Because of the most effective vertical scalability and highest granular sharing with the hypervisors, you can define LPARs that share capacity and can nondisruptively scale up and down as needed for cloud services.

With the dedicated I/O subsystem on IBM Z and LinuxONE, Red Hat OpenShift Container Platform can take advantage of more capabilities for I/O intensive workloads.

It is helpful to start with your use case and plan the Red Hat OpenShift Container Platform environment that is based on the requirements of that use case. It is important to consider that the environment is a User Provisioned Infrastructure (UPI), which means that Red Hat OpenShift manages the HA of the container applications and the user or administrator is responsible for the infrastructure availability and management.

For a use case that implements an Red Hat OpenShift environment that is colocated with IBM Db2® or services on z/OS, you must plan for a network-intensive workload and an effective fast network topology between Red Hat OpenShift Container Platform and z/OS.

For a more CPU intensive workload in which web services are implemented in Red Hat OpenShift, the external network must be planned with scalability for capacity as the focus.

2.2.7 Security

Security in Red Hat OpenShift Container Platform running on IBM Z is a broad subject because it embraces several layers of security, from hardware to container level.

Although Red Hat OpenShift Container Platform includes some immediately available security, such as the requirement of nonroot application containers, and the fact that the hypervisor level provides another layer of security as does the operating system of the guests that do not have direct access to the hardware, the following extra security aspects must be considered:

- ▶ Container security: Containers share underlying operating system kernels; however, package libraries and binaries must be scanned for vulnerabilities.
- ▶ Application security: Scanning of malicious code and application libraries.
- ▶ Images source: Ensure that the base images are from a reliable source.
- ▶ Image registry: Build a reliable source of images of applications.
- ▶ Securing network: Use NetworkPolicy to isolate pods and namespaces. Also, ingress and egress to control traffic coming in and out of the cluster.
- ▶ Hardening kernel security: Custom kernel security parameters can be set in the nodes through MachineConfig objects.
- ▶ Securing Red Hat OpenShift Container Platform: By applying suitable role-based access control (RBAC) to namespaces or replacing cluster API certificates by a custom certificate authority, cluster security can be hardened.

This section is not meant to be an extensive discussion of the subject because many aspects can be included. For more information about relevant security aspects of Red Hat OpenShift Container Platform on IBM Z, see this Red Hat OpenShift [web page](#).

Important: FIPS and encrypted etcd are not yet supported on Red Hat OpenShift Container Platform on IBM Z.

Container security

A reliable and trusted image source is the base for container security. Red Hat and IBM provide a trusted repository of base and build images and containerized software for IBM Z. These components are available at the following web pages:

- ▶ [Red Hat Ecosystem Catalog](#)
- ▶ [IBM Z and LinuxONE Container Registry](#) (log in required)

Although having a secure and trusted image to build your application is an important aspect, it is not the only consideration. The build pipeline must ensure that no malicious code or libraries are injected into your container. Therefore, strategies to mirror artifacts and the ability to scan them are essential pieces of the security puzzle.

Following the container lifecycle, containers must be scanned for vulnerabilities and the vulnerable images must be replaced after the containers are deployed. It is important to remember that containers are ephemeral (lasting only a short time); therefore, a container is not patched. Instead, the image is replaced and the application is built again by using the suitable nonvulnerable images.

OpenSCAP is an auditing tool that uses the Extensible Checklist Description Format (XCCDF) and can be used to scan images and containers for s390x architecture. It checks for vulnerabilities and can be used for security specifications, such as Payment Card Industry Data Security Standard (PCI DSS).

Figure 2-9 shows a sample of an image vulnerability report that was generated by OpenSCAP.

OVAL Results Generator Information

Schema Version	Product Name	Product Version	Date	Time
5.10	cpe:/a:open-scap:oscap	1.3.3	2021-07-21	18:33:49
#X	#V	#Error	#Unknown	#Other
0	619	0	0	0

System Information

Host Name

localhost.localdomain

Operating System

Red Hat Enterprise Linux

Operating System Version

8.4 (Ootpa)

Architecture

Unknown

Interfaces

OVAL System Characteristics Generator Information

Schema Version	Product Name	Product Version	Date	Time
5.10	cpe:/a:open-scap:oscap	3	2021-07-21	18:33:49

OVAL Definition Results

X

V

Error

Unknown

Other

OVAL Definition Generator Information

Schema Version	Product Name	Product Version	Date	Time
5.10	Red Hat OVAL Patch Definition Merger	3	2021-07-21	15:05:52
#Definitions	#Tests	#Objects	#States	#Variables
619 Total	9903	2318	2303	1
00006190				

Figure 2-9 Image vulnerability report

The IBM Z system is growing fast; however, a gap still exists in tools for image and container security on s390x architecture.

Network

Because HiperSockets uses an internal LAN for LPAR-to-LPAR, memory-to-memory communication inside the same CEC, it provides another layer of security because no wiring or traffic exists outside of this CEC; that is, no traffic intercept occurs between these LPARs.

Therefore, whether the use case permits, enabling HiperSockets bring benefits to performance and security.

2.3 Sizing

Many aspects must be considered when you are sizing a Red Hat OpenShift cluster, such as the number of control planes and compute nodes, memory and cores that are assigned to each node, storage, and the kind of workload to be deployed.

It is equally important to plan ahead for the number of operators and IBM Cloud Paks (if used) because any software that is added to the cluster might incur more IFLs; therefore, they can reflect on the number of licenses that are based on IFL count.

In this section, we guide you in determining sizing.

2.3.1 Workload characteristics

One of the main aspects to consider while sizing a new cluster is to understand the workload characteristics because they can influence other sizing aspects.

Whether a workload is stateless and does not require data persistence or it is a stateful application (such as a database that demands data persistence) directly affects storage requirements. For example, a Java application, which can be memory intensive and have large heap requirements, also affects memory requirements, while a Go application is more CPU-intensive and demands more cores that are allocated to a pod.

Nonfunctional requirements, such as Service Level Agreements (SLAs) can determine that you must define pod requests equal to pod limits to avoid the Scheduler from bringing down pods because of capacity starvation. Such a configuration might avoid or limit over-committing in the cluster level.

Therefore, getting to know how an application behaves is an important task and must be avoided. If such information is not available, it is worth the effort to perform tests, and capture enough metrics to understand such characteristics.

As we explore the next sizing topics, the relevance of understanding the workload characteristics becomes clearer.

2.3.2 Consolidation ratio or factor

Containerized workloads that are headed toward the cloud can be densely paced to lower software, data center costs, and administrative overhead. Workload characteristics, such as application workload requirements, influence consolidation. Consolidating workloads onto denser, centralized computing platforms is an effective way to decrease IT expense.

Complete the following steps to size your workload when migrating from a distributed environment to IBM Z or IBM LinuxONE. Assume a consolidation factor for workloads (usually 4:1 - 10:1). Some workloads, such as ELK or Cassandra, do not consolidate well and are 1:1, but that is rare:

1. Find the average utilization on the distributed side per workloads (typical range is 5 - 20%, but 15% is an adequate starting place if this information is not readily available, especially for on-premises Intel servers). Differentiate between bare-metal servers versus virtualized environments, as the environments typically have higher utilization.

If the workload is CPU intensive, the consolidation factor is on the lower end. If the workload is I/O intensive (that is, storage or network), the consolidation factor often is at the higher end because of the I/O offloading capabilities of the processor.

2. Find throughput and performance studies for the workload. For example, Java performance on IBM Z is 50% - 2x higher per core than on Intel. Workloads, such as Java, have special hardware accelerated instructions that often improve the consolidation factor.
3. Combine all the preceding information into a range of 2:1 - 10:1 and size for the best and worst case. Work with your hardware team to get a range of how many Integrated Facility for Linux (IFLs), central electronic complexes (CECs), and so on, that are needed for best and worst case scenarios to ensure that adjustments can be made during the project smoothly.
4. Always size for contingency (typically 20 - 30%).
5. Ideally, a Proof of Concept must be performed with a subset of workload to get an accurate assessment and to tune the consolidation factor.

Limits and requests

Limits and requests are the mechanisms to control resource usage in a cluster and are defined at the container level. The pod total request is a sum of all containers requests.

Limit is the maximum resource that a pod can use in a node and does not go beyond that limit; for example, a pod with a two-core limit cannot use more than that limit.

However, a *request* allocates an amount of resources to that specific pod and this resource is ensured. Therefore, a pod that requests four cores in a 2 CPU node is never scheduled.

Limits and requests can be expressed as millicore or cores for CPU and as integer or fixed point for memory. For more information, see the Kubernetes Official documentation about limits and requests at this [web page](#).

Example 2-1 shows the pod specification in a yaml file. In this example, the container inside a pod is limited to the use of up to 1 GB of memory and up to 1 core (1000 millicores) and requests (allocates) 4 MB of memory and 100 millicores (0.1 core). Therefore, the nodes that are available to schedule this pod must have at least this amount of memory and cores; otherwise, the pod is not scheduled.

Example 2-1 Pod specification yaml file

```
containers:
  - name: appl
    image: busybox
    resources:
      limits:
        memory: "1GB"
        cpu: "1000m"
      requests:
        memory: "4Mi"
        cpu: "100m"
```

This example is a usual pod specification and an effective way to control resources. This example allocates enough CPU and memory for that specific application to start and limits its usage to a specific amount of resources.

However, in a node pressure situation where the node is running out of memory or CPU, a kubelet begins to look for users' pods to evict in an attempt to reclaim resources. Its algorithm ranks the pods to be evicted in a specific order and one of the criteria is a pod that uses more than its request.

Considering Example 2-1, which requests 100 millicores but the application uses 500 millicores, this pod can be eligible for eviction in a node pressure situation. For more information about the eviction algorithm, see this [web page](#).

Depending on the application SLA requirements, you might decide to set requests that are equal to limits to avoid evicting a pod in such a situation; however, doing so limits the level of overcommitting at the cluster level, which reduces the number of pods that can be deployed per node.

Assuming an application node with 8 CPUs and pods requesting two cores each, this specific node accommodates only for of these pods because requests allocate that amount of resources whether the application uses it.

This decision directly reflects on the number of application nodes in a cluster. In fact, a cluster can have different application node sizes to accommodate different types of workloads; therefore, consider this issue as you review other sizing aspects.

Important: Other node configurations affect the number of pods that can be used per compute node, such as `podsWithCore` and `maxPods`. For more information, see the Red Hat official documentation at this [web page](#).

2.3.3 Multi-tenant or single-tenant cluster

A multi-tenancy design implements a shared environment, which is used and operated by different application teams (tenants). A single-tenancy design implements a dedicated environment for a specific tenant.

Because this subject includes several different aspects, such as logical isolation, network, security and many others, we focus only on shared resources, such as cores and memory.

A multi-tenant cluster seems to be a natural choice in a microservices world with several tiny containers running and sharing resources; having logical resource isolation by namespaces; and using resource quotas, limits, and requests. A multi-tenant cluster can be achieved with a good level of isolation and avoids applications “stealing” resources from each other.

However, use cases exist that justify a single-tenant cluster; that is, an entire cluster for a single specific application. This cluster can be a resource-intensive application or an application with several microservices components that demand dedicated resources.

In either case, this decision reflects on the cluster sizing because several tiny clusters require more control nodes, which demand more cores. However, a single cluster that is shared by hundreds of application can require a higher level of overcommitting and increase the complexity to manage the resources.

2.3.4 Sizing control planes, compute nodes, and infrastructure nodes

In this section, we examine the sizing of control planes, compute nodes, and infrastructure nodes.

Control plane

According to [Red Hat documentation](#), a minimum of three control plane nodes is needed. The odd number of control planes is for the consensus algorithm that is required by etcd. The use of three control planes instead of a single control plane is to give the cluster a level of HA. The use of more than three is not recommended because it can increase latency because of etcd performing more unnecessary checking for consensus.

Each compute node brings another overhead to the control plane nodes; therefore, estimating how many applications and resources are required by each of them in terms of memory and cores becomes important in determining how many compute nodes the cluster requires and then how much resources to assign to the three control planes. For more information, see “Limits and requests” on page 37.

Table 2-3 lists the suggested number of cores and memory to allocate to each control plane node in relation to the number of compute nodes.

Table 2-3 Suggested control plane capacity per number of compute nodes

Number of compute nodes	Cluster load (namespaces)	CPU cores	Memory (GB)
25	500	6	16
100	1000	8	32
250	4000	16	96

Important: Use Table 2-3 as a reference and suggested values because many other factors reflect in control plane resource consumption, such as frequency of deployment and the number of applications that is deployed.

A user-provisioned infrastructure installation allows for modifying the control plane size of a running Red Hat OpenShift Container Platform 4.7 cluster.

Another consideration is to keep a specific level of usage in the control nodes to provide room for spikes and in situations where one of the nodes goes down and the remaining nodes must handle the workload. The same situation also can occur during other cluster operations, such as an upgrade or modifying machine configuration objects, which drains the nodes individually and restarts.

Compute nodes

Sizing the compute nodes features a straightforward relationship with the workload characteristics and planned resource management and the z/VM LPAR size in terms of IFLs.

The number of cores (IFL) available or required for the LPAR determines the maximum number of virtual cores that can be assigned to each compute node. Do not have more virtual cores (vCPUs) assigned to a single guest (node) than the number of Logical Processors available in the LPAR. Because Red Hat recommends a minimum of six IFLs that are required by a cluster with SMT-2 enabled, a total of 12 logical processors are used, which limits each node to have assigned a maximum number of 12 vCPUs.

Important: Even when a cluster is spread over two or more LPARs, each LPAR that is hosting a compute node with eight vCPUs needs at least four IFLs with SMT-2 enabled to provide the required eight Logical Processors.

2.3.5 Sample scenario

To demonstrate how to size control planes, compute nodes, and infrastructure nodes, assume the following cluster setup (see Figure 2-10):

- ▶ A single z/VM LPAR with six IFLs
- ▶ Three control plane nodes with four vCPUs and 16 GB RAM
- ▶ Two compute nodes with eight vCPUs and 32 GB RAM
- ▶ A Java application that uses two vCPUs and 4 GB RAM during peak load time

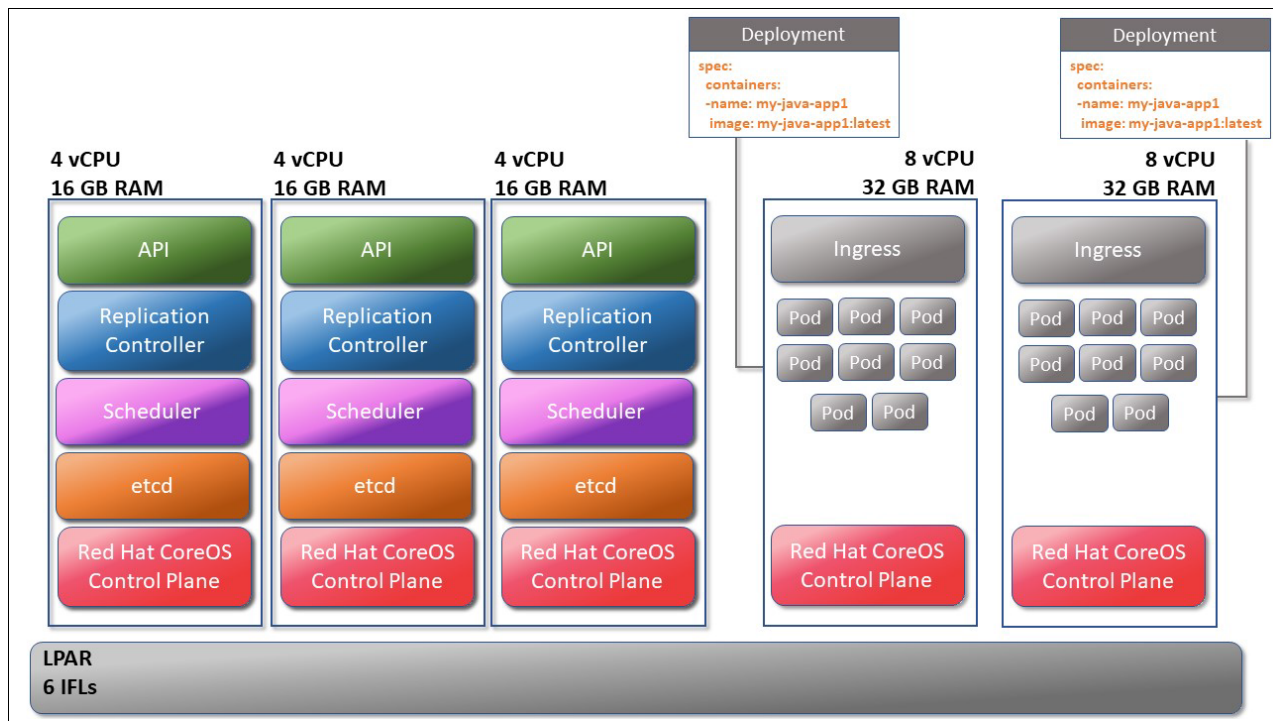


Figure 2-10 Initial cluster configuration and size

Considering the cluster that is shown in Figure 2-10, the exercise that is described next analyzes a few different scenarios.

Suppose that an application team agrees to a specific level of overcommit on a vCPU because they are not sure that all the pods reach their peak at the same time.

Take the total amount of memory in the compute node and divide that amount by the application memory requirements (because memory cannot be overcommitted). Therefore, memory is the constraint in this scenario. Example 2-2 shows the calculations per node.

Example 2-2 Memory constraint

$$32 \text{ GB} / 4 \text{ GB} = 8 \text{ Pods}$$

You multiply the eight pods by the application core requirements to understand the level of overcommit, as shown in the following equation:

$$2 \text{ vCPU} * 8 \text{ Pods} = 16 \text{ vCPU}$$

If all pods reach their peak at the same time, the workload attempts to use more vCPU than is available in the compute nodes and some threads must wait for an available core.

However, this application can be left with that level of overcommit because not all pods are active and reach the peak at the same time. Therefore, how many pods like that can be deployed in a cluster of this size? To calculate this figure, you multiply the number of pods by the number of vCPU, as shown in Example 2-3.

Example 2-3 Calculating the number of pods

$$8 \text{ Pods} * 2 \text{ compute nodes} = 16 \text{ Pods}$$

By using this configuration, 16 pods of that same size can be deployed, which allows a specific level of overcommit on the vCPU.

However, the application team has a new requirement and now, they want to allocate that amount of vCPU that was identified during the peaks to avoid a lower throughput because of pods competing by cores.

Therefore, a request is set to allocate the required amount of vCPU to the container, even if not used during a specific time. The number of cores is guaranteed, so the vCPU starts to be constrained now instead of memory:

$$8 \text{ vCPU} / 2 \text{ vCPU} = 4 \text{ Pods}$$

To calculate how many similar pods can be deployed in this same cluster, use the following calculation:

$$4 \text{ Pods} * 2 \text{ compute nodes} = 8 \text{ Pods}$$

By avoiding overcommitting, it reduces the cluster capacity by half in terms of pod density. However, the application team now requires that at least 16 pods of that application running in the cluster.

This goal can be accomplished by scaling the node capacity vertically, but the number of available Logical Processors in this LPAR is limited. Therefore, we decide to scale horizontally by increasing the number of compute nodes.

Per the previous results, setting the request to two vCPU, each cluster can take four pods. To give enough capacity for 16 pods, calculate how many compute nodes are required by using the following formula:

$$16 \text{ Pods} / 4 \text{ Pods per compute node} = 4 \text{ compute nodes}$$

With four compute nodes, the cluster handles the load of 16 pods of that application, which avoids overcommitting on cores.

Application requirements were fulfilled and the cluster handled the load; however, the operations team noticed that during a cluster upgrade, some pods were evicted.

What is happening? Notice each compute node is running with their maximum capacity allocated ($4 \text{ pods} * 2 \text{ vCPU} = 8 \text{ vCPU}$) without leaving any room for extra load.

During a cluster upgrade (or some other operations activity, such as modifying machine configuration objects), each node is drained individually and the remaining nodes likely can handle the load of the drained node, which is not the situation of the compute nodes that is described in our example. All have their resources allocated because the current applications are setting the request at the container level. So, the cluster scheduler cannot find available capacity to schedule the extra load from drained node.

With this information, it is important to define a level of usage in the cluster and nodes, one that better fits the requirements and workload characteristics. For the sake of simplicity, our example takes 50% of usage level in the compute nodes. This value is not recommended; it is used to keep round numbers in our example:

$$8 \text{ vCPU} * 50\% \text{ of usage} = 4 \text{ vCPU}$$

Therefore, because the operations team wants to keep a level of 50% of usage in the compute nodes, the pods that are deployed cannot exceed the four vCPU of usage or allocated resources (requests). For our example, adding two compute nodes with the same capacity is enough to handle the extra load from drained nodes.

Important: Our example is a rough estimation because it does not consider the allocated resources for the Red Hat OpenShift Container Platform components, such as the kubelet and monitoring.

Infrastructure nodes

An infrastructure node is nothing more than a regular compute node that is labeled as `infra`. The purpose of this kind of node is to schedule infrastructure components, such as Prometheus, Registry, and Router (Ingress) to these specific nodes, which releases capacity of compute nodes to application workloads.

This operation spans two days because it does not include a regular Red Hat OpenShift cluster installation. It also must be decided whether infrastructure nodes are required and which infrastructure components are moved over to these dedicated nodes.

Although it is essentially a regular compute node, some other factors must be considered to size them. The main factor to consider is the size of the cluster in terms of nodes, the more nodes a cluster has, and the more metrics it generates.

Table 2-4 lists recommendations per number of compute nodes.

Table 2-4 Recommended sizing for infrastructure nodes per compute nodes in a cluster

Compute nodes	CPU cores	Memory (GB)
25	6	16
100	8	32
250	16	128
500	32	128

Each of the following infrastructure components can affect sizing:

- Prometheus

Prometheus is memory that lack unity and its resource usage depends on several aspects of the cluster, such as number of nodes, cluster objects, and scraping interval. Disk usage also is related to retention period, which can be customized.

Scraping interval also can be increased to reduce the periodicity to gather metrics from cluster objects (such as pods) and consequently reduce memory and CPU usage.

The number of custom label attributes for metrics directly affects Prometheus resource consumption and performance because each assigned key-value pair generates its own time series.

- Registry

Red Hat OpenShift Container Platform provides an immediately available image registry to be used internally or externally. The disk that is used depends on the number of images and image size. CPU consumption depends on deployment strategies and frequency, containers lifecycle, and build strategies. Although it is not a required component, some other objects depend on it, such as ImageStream.

- Router (Ingress)

A Router (or Ingress) is the entering point for any consumer that is outside of the cluster, including consumers of infrastructure components, such as Console and cluster API. Therefore, it depends on the number of deployed routes, network traffic, and latency between the infrastructure nodes and compute nodes.

Its consumption also is affected by application characteristics (for example, static or dynamic content), TLS (as encryption activities is CPU intensive), HTTP keep-alive, the number of concurrent connections, and back-end response data size. Because of these variants, it is difficult to define how many routes a Router can take without knowing the workload characteristics.

The initial deployment of a cluster spins-up two replicas of a Router, each configured with four threads. Some Router configurations can be defined through labels in Ingress Controller Operator, but it is limited and not all Router configurations can be customized. The Router can be scaled vertically by adding resources to infrastructure nodes, or horizontally by creating infrastructure nodes and scaling the number of replicas.

Important: For more information about other components that can be deployed into an infrastructure node see this Red Hat OpenShift [web page](#).

2.3.6 Cores, memory, and storage

Now it is time to put all of this information together and determine how many physical resources a cluster requires. This process is not necessarily simple process because too many variants exist, such as workloads characteristics, some CPU intensive, others the use large heaps, databases needing persistence, and stateless applications with no storage requirements.

Consider the scenario that is described in “Sample scenario” on page 40. The cluster requirements are listed in Table 2-5.

Table 2-5 Cluster requirements

Machine	Quantity	vCPU	Memory	Disk
Control Plane	3	4	16 GB	120 GB
Compute	4	8	16 GB	120 GB

Important: A temporary node that is called a *bootstrap* also is required.

Cores

Throughout the sizing exercise, several times we refer to cores, virtual CPU, and logical processors; therefore, consider the following definitions:

- ▶ IFL
Integrated Facility for Linux. This physical core is in a IBM Z environment, which is dedicated to Linux workloads.
- ▶ Logical Processors
These processors represent the share of IFLs that is assigned to an LPAR. This amount is the quantity that an LPAR recognizes as available processors.
- ▶ SMT
Simultaneous multithreading (SMT) enables separate threads to run concurrently in the same processors. When SMT-2 is enabled in an LPAR with four Logical Processors, each guest within that LPAR sees it as eight available Logical Processors.

In our scenario, we have a single LPAR with six IFLs, which is the minimum that is required according to Red Hat documentation. With SMT-2 enabled, cluster has 12 Logical Processors. This amount initially is enough because each compute node has eight vCPUs, which respects the recommendation to not define more vCPUs in a single node than the available Logical Processors in the LPAR. Diving a bit deeper, how many virtual cores (or vCPUs) are allocated to the cluster in total? Consider the following example:

```
3 Control planes * 4 vCPU = 12 vCPU
4 Compute nodes * 8 vCPU = 32 vCPU
Total vCPU = 44 vCPU
```

Considering the 12 Logical Processors that are available to this LPAR, determine the vCPU versus Logical Processors ratio by using the following formula:

```
44 vCPU / 12 Logical Processors = 3.6:1
```

One Logical Processor is available for each 3.6 vCPU (or virtual cores). Although this metric is important to follow, no rule or recommended magic number exists because it depends on the workload characteristics (for example, whether all of the vCPU that is fully used during most of time) and the cluster environment (for example, is it a test, development or production environment?). Therefore, closely monitor this metric, pay attention to the workload behavior, and fine-tune it to your needs to understand the best ratio for you.

Memory

Sizing the memory for the cluster is a bit more straight forward, although a few specific considerations are important. Swap memory does not exist on disk at the guest level (in the node); therefore, what you assign for RAM is what you get.

Also, in contrast to vCPU, if your node runs out of vCPU, the application might delay in responding, but is (most likely) kept up. However, if you run out of memory, the pods are evicted immediately if no resource is available in any of the nodes.

Referencing our scenario, how much memory is required to handle that cluster? Consider the following example:

```
3 Control planes * 16 GB RAM = 48 GB RAM
4 Compute nodes * 16 GB RAM = 64 GB RAM
Total GB of RAM = 112 GB of RAM
```

Therefore, the LPAR must make available to the guest a total of 112 GB of RAM. Depending on how the requests and limits are used in a pod specification, it is possible to overcommit on memory in the node level. It also might be acceptable if it is known that not all of the applications are fully using all of the memory resources during the same time but use it carefully.

Important: In relation to the guests (nodes) of the hypervisor, a specific level of overcommit on memory might be acceptable in a test or development environment; however, it is not recommended in production environments.

Storage

The local storage that is attached to each guest is intended for the operational system usage and the containers and images for the applications that are running in that specific node. Although this local storage can be used as a Persistent Volume (directly or through the Local Storage Operator), such an approach might not be suitable for all applications because it relies on the underlying node availability.

To suit more complex application storage requirements, consider the use of a clustered storage solution, such as Spectrum Scale and Red Hat OpenShift Container Storage. For more information, see “Storage topology” on page 30.

In our example, which was sized with the minimum amount of local storage, the cluster has the following storage needs:

```
(3 Control Plane + 4 Compute nodes) * 120 GB = 840 GB of disk
```

A total of 840 GB of disk storage must be made available to the LPAR through DASD or FCP.

2.4 Continuous availability for applications

With a Red Hat OpenShift Container Platform environment comes the ability to enable continuous availability of the application services.

The implementation of the Red Hat OpenShift Container Platform components and the underlying Kubernetes services are deployed for continuous operation. Their dependencies are the hypervisor and infrastructure services. HA and DR must be part of the continuous availability planning and are typical requirements for production deployments.

For planning continuous availability, the main considerations are:

- ▶ Removing single points of failure on the various areas of the Red Hat OpenShift Container Platform environment.
- ▶ Application deployment and DevOps integration (to avoid downtime with rolling updates).
- ▶ The use of HA and DR concepts for the infrastructure.

The goal is to ensure continuous operation and avoid planned and unplanned outages of application services. Figure 2-11 shows the components and the access path to an application that runs with HA.

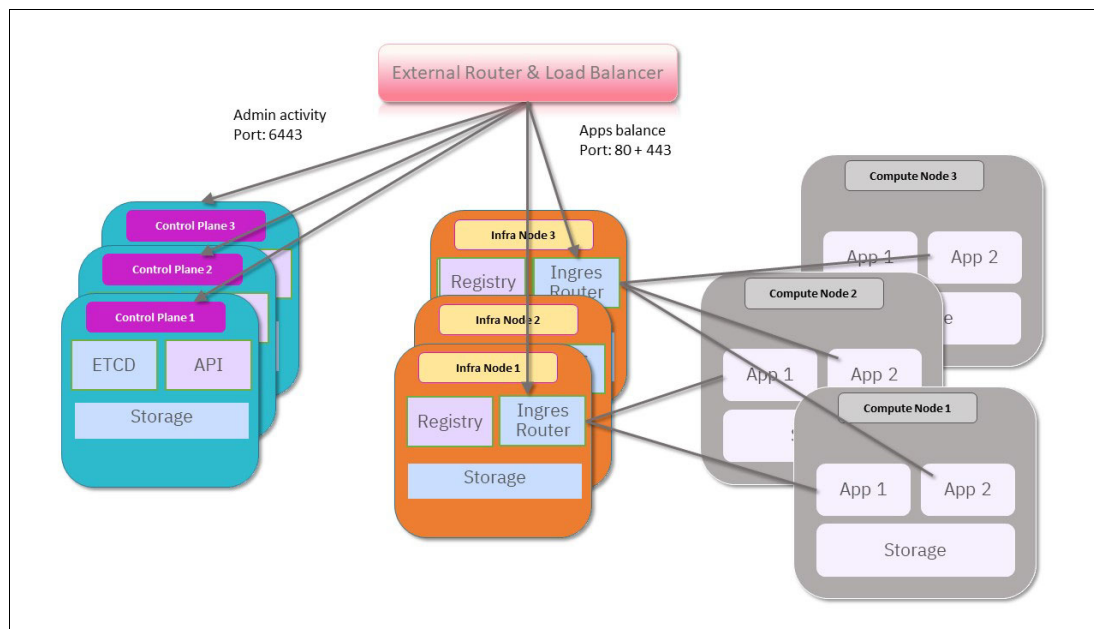


Figure 2-11 Red Hat OpenShift Container Platform application with HA

2.4.1 Avoiding single points of failure

A Red Hat OpenShift Container Platform environment is based on a clustered implementation. For continuous availability, it must meet the following minimum requirements:

- ▶ Three Control plane nodes (formerly known as *Master nodes*)
- ▶ Two Compute nodes (formerly known as *Worker nodes*)
- ▶ Two or three Infrastructure nodes

All of these nodes are virtual machines that are running under the control of a hypervisor that can be IBM z/VM or Red Hat Enterprise Linux KVM. The hypervisors can run in one or multiple LPARs in one or multiple physical IBM Z or LinuxONE machines.

With the three Control Planes, the implementation of Red Hat OpenShift Container Platform ensures that the cluster is manageable if the quorum for those nodes exists. That means that at least two of the three Control Planes are operational. This configuration ensures that the entire cluster and the workload in the cluster is operable.

The Control Planes include container workloads that are running in their pods, which are responsible for supervising the behavior of the pods and nodes and the deployed applications in the entire cluster. They also ensure the management of the nodes in the cluster and keep the control over the number of pods and replica sets of an application to ensure continuous operation. This status of the applications and cluster behavior is stored in the etcd database, which has a replica in each Control Plane.

If one Control Plane is malfunctioning, the Red Hat OpenShift Container Platform cluster is still fully functional and operable because the quorum is still viable for the Control Planes.

Note: Whenever more than one Control Plane is malfunctioning, a quorum is not possible and an action must be started automatically or manually to restart or rebuild the failed Control Planes. A timely restart of a Control Plane can regain the quorum and the Red Hat OpenShift Container Platform senses that change and makes the cluster operational and manageable again.

Even without a quorum, the cluster is not stopping its applications in the Compute Nodes. Although they still run, they no longer can scale and even an administrator cannot change the cluster until quorum is reestablished.

It is the same for Compute Nodes and Infrastructure Nodes, only at the application level. The nodes operate individually and are part of the overall Red Hat OpenShift Container Platform compute cluster. These nodes host the application pods and the monitoring, container registry pods, and other logic, such as routes and ingress pods.

The Control Planes manage the deployment of applications and ensures that they are deployed in multiple nodes and pods to ensure continuous availability of an application service.

Note: In the planning stage of the deployments, the administrator must configure how many replicas are to run simultaneously for a specific application logic. The ReplicaSet number automatically starts a specific number of pods for an application. If a Compute Node is malfunctioning, the application pod is automatically redeployed in another functional node and becomes highly available again.

A Red Hat OpenShift Container Platform cluster uses its automation to supervise, restart, or repartition the application to ensure continuous operation of the application service. The routers and routes in the Red Hat OpenShift Container Platform cluster ensure that access to an application is always given and is maintained if some components are malfunctioning.

The infrastructure environment is the responsibility of the IT Administrator because IBM Z and IBM LinuxONE features a User Provisioned Infrastructure (UPI) deployment. Also, Red Hat OpenShift Container Platform can manage only the continuous operation from a pod at the application level.

Therefore, for continuous availability of the infrastructure, you must plan for avoiding single points of failure on the storage, network, hypervisor, and Red Hat OpenShift Container Platform nodes level.

For more information about the options that are available to achieve HA, see “High availability considerations” on page 49.

2.4.2 Deploying applications and integrating DevOps

To ensure continuous operation in a continuously available environment, the integration of a lifecycle concept for the operating systems and the base environments is required. Red Hat OpenShift Container Platform is integrating several capabilities per design, including the following examples:

- ▶ Complete Red Hat OpenShift Container Platform cluster updates in place
- ▶ CoreOS lifecycle and updates
- ▶ Application lifecycle and roll-out

The management and service lifecycle of the environment consists of servicing and updating the CoreOS operating system. It also includes updating the applications and operational components without interrupting the application services.

It also enables the deployment of new services without downtime, but with rolling updates. Red Hat OpenShift Container Platform upgrades the entire environment to a newer level with a single click.

2.5 Planning for HA and DR infrastructure

For an environment with continuously available active services, the following considerations about HA and DR are important:

- ▶ All layers of the environment (hardware machines, disks, network, virtualization/hypervisor, Red Hat OpenShift Container Platform cluster).
- ▶ A solution architecture with HA and DR concepts.
- ▶ The operational procedures regarding continuous availability based on HA and DR.

When considering HA, also consider a multiplication of all the components and services that are needed for an operational environment.

For DR, the considerations are on a larger scale. You also must consider hardware failures of an entire machine or storage server or a data center or environmental collapse.

2.5.1 High availability considerations

Figure 2-12 shows an example of a highly available Red Hat OpenShift Container Platform deployment.

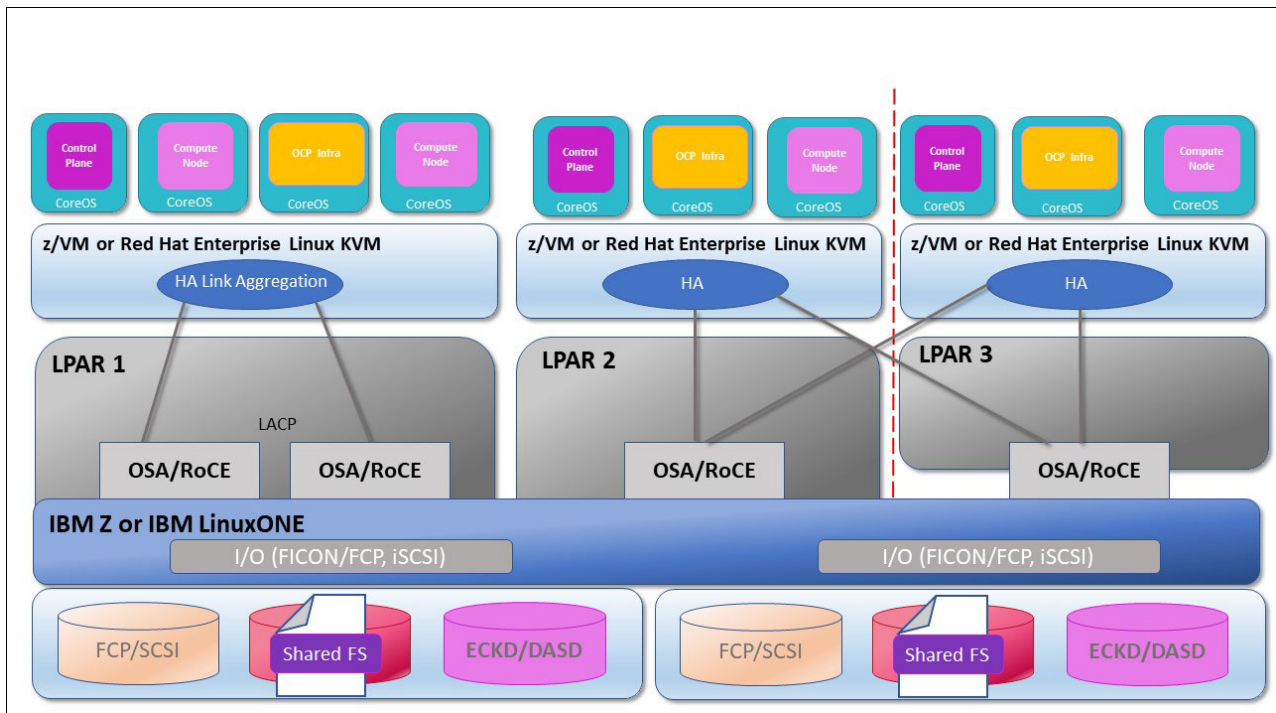


Figure 2-12 Red Hat OpenShift Container Platform cluster HA

For HA, the entities that ensure that the environment is operational can be in a single hardware machine, in different LPARs (as shown in Figure 2-12), or they can be spread across two or more (ideally three) hardware machines.

The three machines amount is chosen for redundancy because of the HA definitions for a quorum. The layers for HA are the same and independent of the numbers of hardware machines, only the rate at which the guarantee for a highly available infrastructure increases with the number of physical machines and sites.

Storage high availability

For storage HA, plan the HA for the hypervisor storage first, where the hypervisor runs and its guests that represent the Red Hat OpenShift Container Platform Nodes. For the storage HA, plan for a solution with storage replication for the Hypervisor storage.

The storage that is used for container workloads can be replicated if local attached storage is used, or it can use software-defined replication. Especially with Spectrum Scale CNSA, you can use a stretched cluster across Metro sites, which is managed by the software-defined storage components in the IBM Spectrum Scale storage servers.

IBM Z and IBM LinuxONE

The hardware and LPAR virtualization of the IBM Z and IBM LinuxONE platform offers characteristics that make them a perfect match for HA (for example, redundancy in the machine by way of hardware design) and highest isolation in an LPAR. Therefore, many installations rely on the characteristic of decades of Mean Time Before Failure (MTBF) of IBM Z hardware.

Planning network topology

The network topology can be planned to use multiple network cards and cross configure them to avoid a single point of failure in a card. With bonding, you can solve two challenges: one regarding HA, and the other the enlargement of bandwidth by bonding multiple cards by using Link Aggregation Control Protocol (LACP) or the BOND device in KVM.

Virtualization layer

The hypervisor in the virtualization layer is the last layer bottom-up that you, as a user, need to manage for HA. You can plan to use the z/VM Single System Image feature to build a cluster with up to four z/VM nodes that can be managed from every node. In doing so, a total collapse of the virtualization layer is avoided in an Red Hat OpenShift Container Platform cluster.

High availability nodes in the Red Hat OpenShift Container Platform

The nodes in a Red Hat OpenShift Container Platform environment on IBM Z and IBM LinuxONE are virtual guests that can run in a single LPAR that is under the control of z/VM or Red Hat Enterprise Linux KVM or can run in multiple LPARs. If running a Red Hat OpenShift Container Platform cluster in multiple LPARs, these LPARs can spread across multiple physical machines to reach the highest availability.

As described in this chapter, you can have different types of cluster nodes, such as Control Planes, Compute nodes, and Infrastructure nodes. If one of the nodes fails, Red Hat OpenShift Container Platform cannot automatically restart the node. Therefore, for HA of the nodes, an external procedure must be in place to ensure continuous operation of the cluster services.

For HA, it is required that more than half of the Control Plane nodes are active and running to have an operational cluster. Therefore, you can plan for special actions to recover a failed Control Plane node or any other nodes.

The following examples show some typical situations and their mitigation strategies:

- Control Plane nodes

Control Plane nodes track the status of operational and configuration changes in the entire cluster. If one Control Plane node is malfunctioning, you must restart the node. This process can be done manually or automatically by using tools, such as Red Hat Enterprise Linux HA and IBM GDPS, or other tools that can supervise the activity of a virtual guest and restart it whenever needed.

If the node is started in time and no hard-defined time limit exists, it starts the Red Hat OpenShift Container Platform components and Red Hat OpenShift reactivates its activities in the cluster. Therefore, the node is repopulated with the control pods to become fully operational.

Note: It is important to have the ability to restart the Control Plane nodes, especially if two out of three are failing because a loss of quorum in the cluster switches the cluster to a state that the applications run only in the status they are in at that time of failure without scaling or failover capabilities.

Also, the cluster is not stopping the applications, but the cluster cannot be operated until the quorum is reestablished, which means that at least two of the three Control Planes are running again.

If the Red Hat OpenShift Container Platform environment is running on multiple LPARs or hardware machines, the nodes can be started on any machine from the original or a copy of the storage disk where the failing node is stored; therefore, it can use the same credentials and security certificates in the cluster. In some situations, it also is a good idea to have “spare” nodes that are defined as virtual machines, which can be started instantly when a node fails. Red Hat OpenShift Container Platform then repopulates the node and continues normal operations.

- Compute and Infrastructure nodes

These nodes run independently, which means no quorum exists between Compute or Infrastructure nodes. Nevertheless, if one of the nodes crashes or multiple nodes malfunction, the applications that are running in the nodes might be affected and lose the quorum at the application level. Therefore, it is important to have these nodes watched as well from an activity perspective to restart them whenever they fail.

- Red Hat OpenShift Container Platform runs in a clustered mode and can fail over pods and their container workloads within the same cluster between the nodes.

In the simplest setup for HA, HA with DR can be unified. This feature can be planned in an active-active implementation with metro distance between the hardware machines.

2.5.2 Planning for disaster recovery

A DR scenario is a scenario in which the environment needs to fail over to another location, which can be in a Metro, Global, or long distance.

A DR scenario features the following implementations:

- Active-active

As shown in Figure 2-13 on page 52, this implementation features Metro distance and two or more physical sites, which all are active at the same time. The Red Hat OpenShift Container Platform environment can be spread across these sites. The distance is relevant because of the network latency and disk replication that it needs to conduct a synchronous replication.

The implementation of the Red Hat OpenShift Container Platform applications can be realized by considering the following points:

- Clusters on each site operate independently, which means that two Red Hat OpenShift Container Platform cluster sets are used that share storage. They also operate based on an external workload balancer, such as F5 or Datapower, which can direct the requests based on capacity, response time, or round robin if all sites operate normally.
- The application storage must be shared between the sites. You can use two shared Storage Servers and unify them logically with IBM Spectrum Scale in a stretched storage cluster. If one site is malfunctioning, the workload balance directs all requests to the healthy site. The advantage is that this configuration is a converged HA and DR implementation architecture and no restart or recovery is needed when switching data centers.

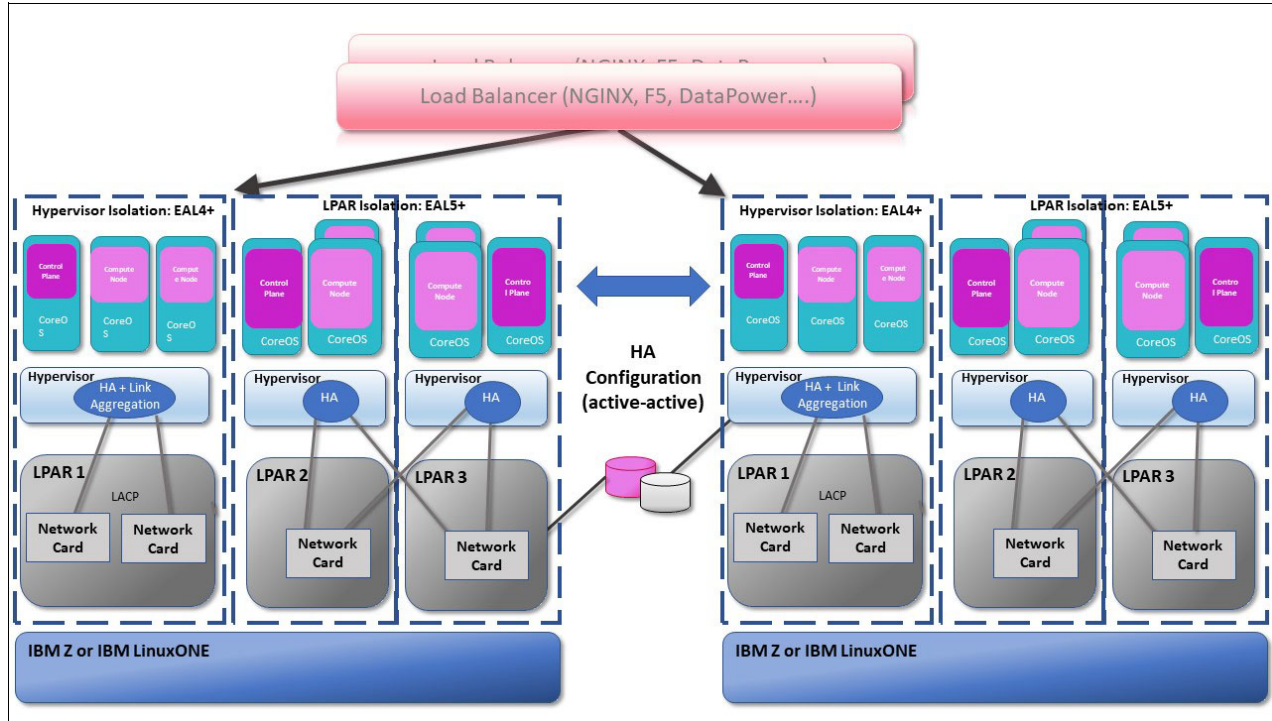


Figure 2-13 Red Hat OpenShift Container Platform DR active-active

- A single cluster with nodes spread across the two data centers.

In this case, the storage for the applications must be shared between the data centers. This sharing can be realized by using (for example) IBM Spectrum Scale CNSA and a stretched storage cluster, as shown in Figure 2-14. The operational model now is that the requests are routed by way of the Red Hat OpenShift Container Platform ingress routers to the applications in the main or DR data center.

In this situation, if one data center is not functioning, responses must be established to timely restart the Control Planes or the other nodes that are not functioning to reestablish the cluster integrity. It is also important to decide which data center is the main center and position the three control planes accordingly.

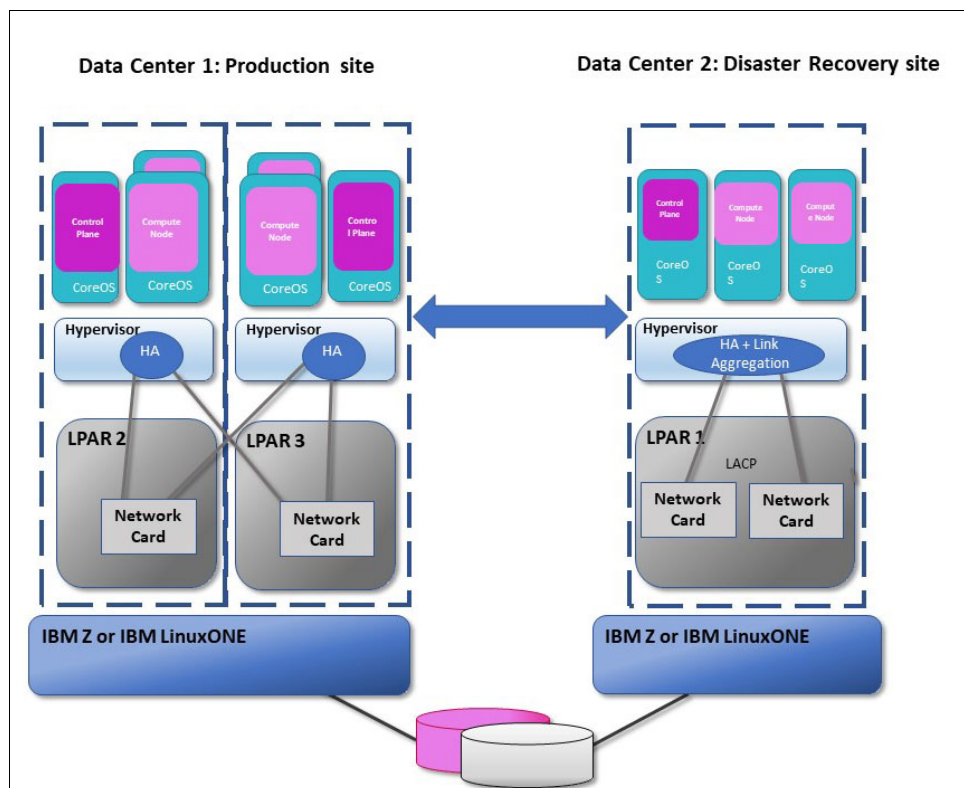


Figure 2-14 Red Hat OpenShift Container Platform DR active-active with 1 cluster

- An active-passive DR scenario (see Figure 2-15).

In this environment, an active site and a site are activated in a disaster or outage on the main site only. In preparation for DR scenarios, consider automation for the failover between data centers. Tools can automate the failover and help in HA, including IBM General Dispersed Parallel Sysplex (GDPS) or an implementation that uses Red Hat Enterprise Linux HA. These tool sets also can help to restart failed Red Hat OpenShift Container Platform nodes and reestablish the connection to storage servers without interruption, such as IBM HyperSwap® with IBM z/VM and GDPS with Extended Disaster Recovery (xDR).

Typically, storage replication is used. In a disaster case, the replicated storage is used to reestablish an identical environment as the main data center. This implementation often generates an outage, depending on the distance the tools that are used and the procedures that are in place to switch to the DR data center.

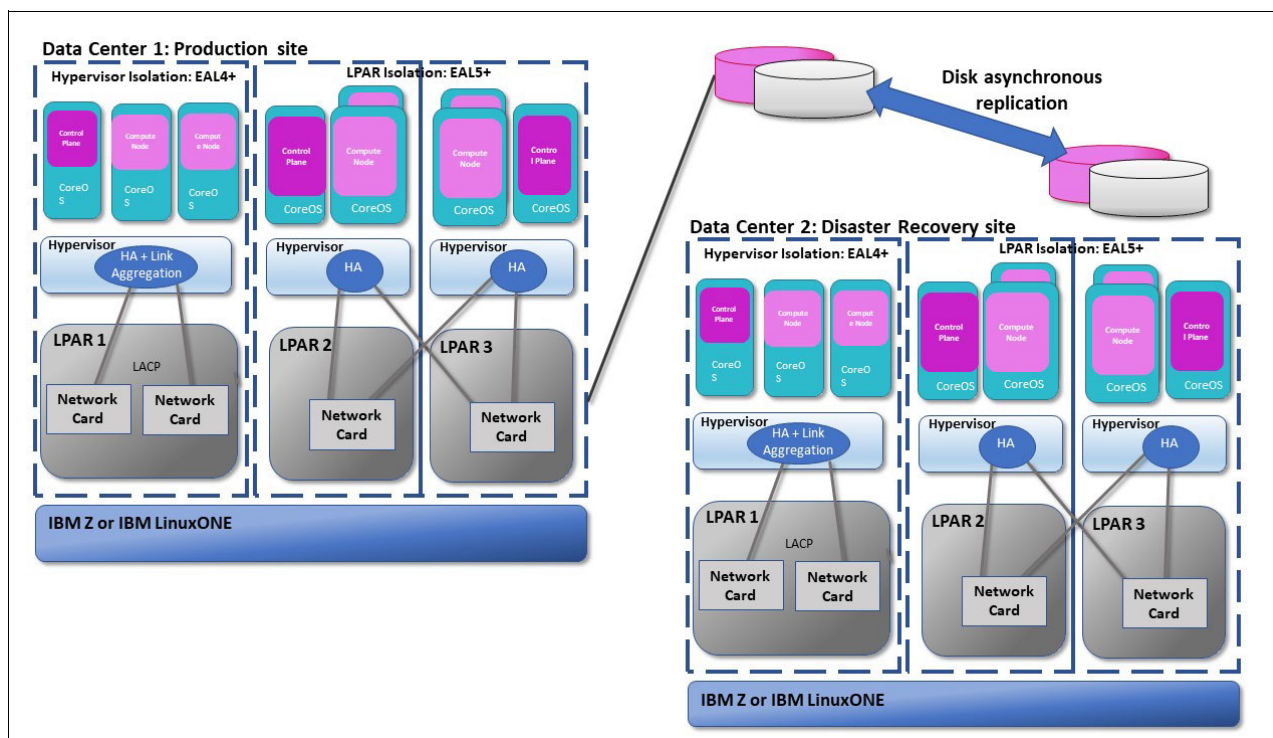


Figure 2-15 Red Hat OpenShift Container Platform DR active-passive



Red Hat OpenShift Container Platform installation for KVM on LinuxONE

This chapter provides Red Hat OpenShift installation steps, deployment, and configuration prerequisites for a successful cluster installation.

Although this chapter covers a Red Hat OpenShift Container Platform installation on KVM, most of the Red Hat OpenShift installation prerequisites and deployment steps are common for KVM *and* z/VM.

This chapter includes the following topics:

- ▶ 3.1, “Architectural patterns: Red Hat OpenShift cluster on KVM” on page 56
- ▶ 3.2, “High-level Red Hat OpenShift installation steps” on page 60
- ▶ 3.3, “Red Hat OpenShift Container Platform prerequisites” on page 61
- ▶ 3.4, “Deploying HAProxy” on page 68

3.1 Architectural patterns: Red Hat OpenShift cluster on KVM

Kernel-based Virtual Machine (KVM) is an open source hypervisor that is part of Linux. In fact, its modules are natively integrated into the Linux kernel. KVM also is used as a “building block” of many hybrid cloud implementations and represents the standard virtualization hypervisor.

KVM on IBM Z and LinuxONE uses all of the specific hardware features of IBM Z and LinuxONE architecture, such as the CPACF Crypto hardware facility and CryptoExpress cards on hosts and virtual machine (VM) guests. It also supports Java pause-less garbage collection, and can use FlashExpress as swap storage for KVM hosts and any kind of SAN (FICON or FC) and still use SAP processors to offload the I/O overhead from IFL CPs.

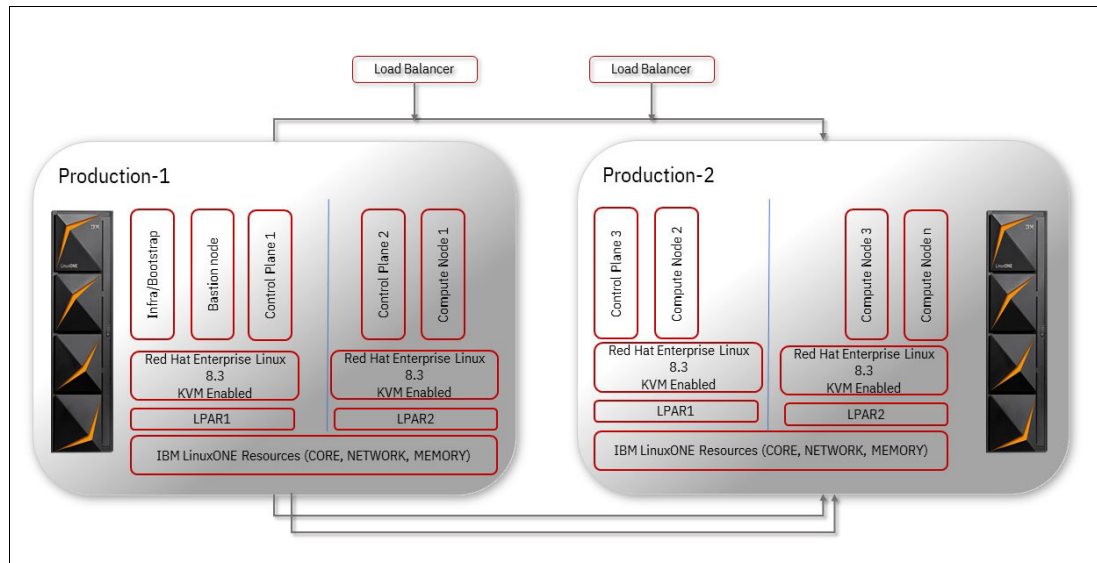


Figure 3-1 High-Level design for an KVM-based Red Hat OpenShift environment

Infrastructure design decisions must be considered before you can start the cluster deployment. For more information about design decisions that must be considered, see Chapter 5, “Use Case: A Red Hat OpenShift environment that is colocated with z/OS transactional services and DB2 data” on page 89.

In this chapter, we describe some of the design decisions we made while designing the cluster environment for this publication. We briefly discuss the following topics from the perspective of our environment:

- ▶ Storage Connectivity: Fibre Channel (FC) and FICON
- ▶ Network Connectivity
- ▶ Virtual Image Format

3.1.1 Storage connectivity

In our test environment, we strove to follow a production-like storage connectivity design. Therefore, out of the eight FC paths that were available for the environment, we assigned four paths to be shared with the host and control plane (Infra Node and the Master nodes) across multiple partitions.

The remaining four paths are shared with the bastion and worker nodes across multiple partitions on the LinuxONE server. Separating paths or ports between the control plan and the worker nodes provides the needed efficiency.

The storage path connectivity that was used in our test environment is shown in Figure 3-2.

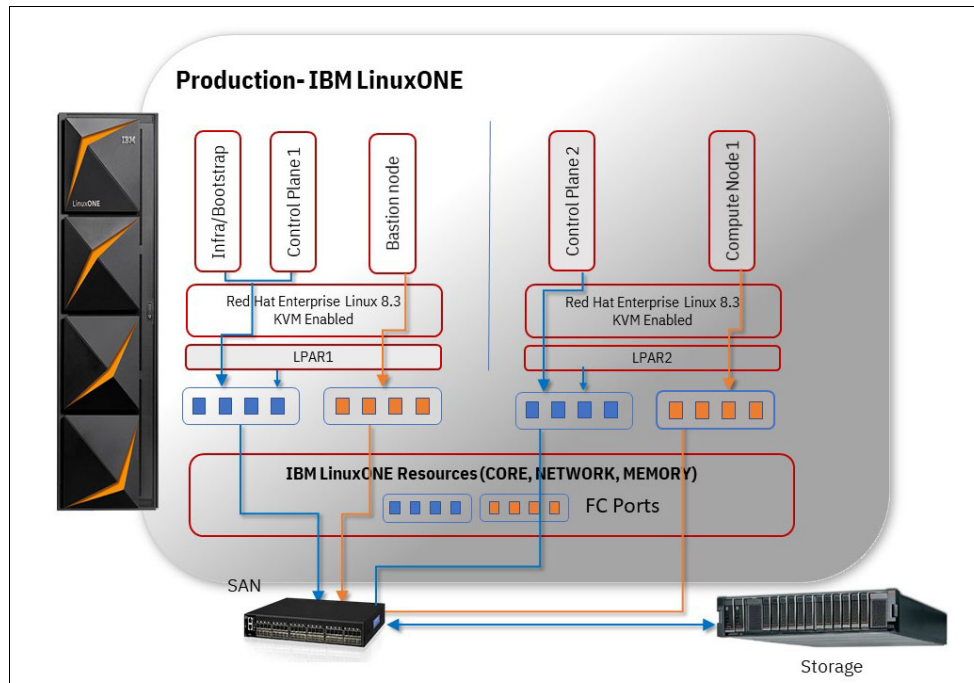


Figure 3-2 Storage path connectivity in our test environment

3.1.2 Network Connectivity

On IBM LinuxONE with KVM virtualization, you can use dedicated network ports or create a virtual switch internal to the virtualization layer. Red Hat OpenShift provides options to use Open-vswitch (OVS) or MacVTAP, depending on the network requirements.

Red Hat OpenShift Container Platform clusters requires logically separated network traffic for different purposes by using virtual local area networks (VLANs).

The network setup that was used in our test environment is shown in Figure 3-3.

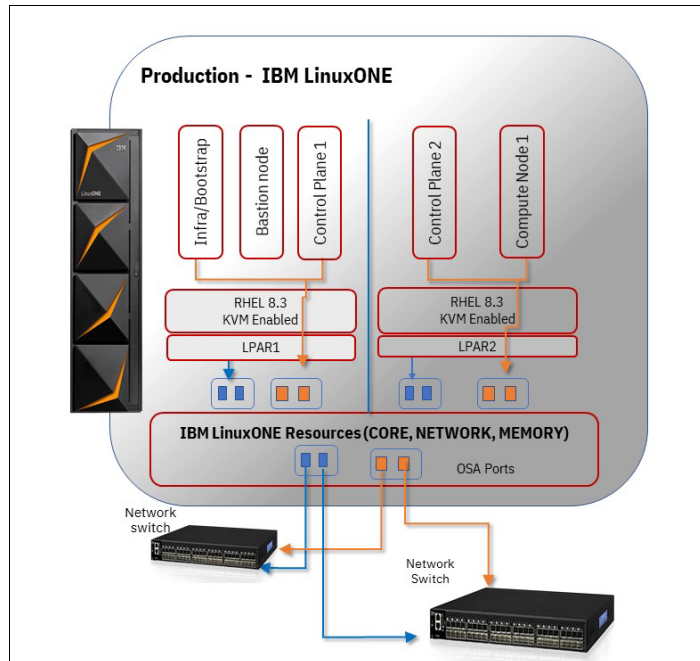


Figure 3-3 Network setup in our test environment

Note: For more information about network patterns, see this [web page](#).

With VLAN configurations, network connectivity can be scaled to meet cluster bandwidth demands and to provide further isolation for specific network services. For ease of use, two separate network ports are assigned in our test environment: one each for HOST and the other for the cluster nodes (control plane and workers).

The MacVTap with the VLAN configuration is shown in Figure 3-4.

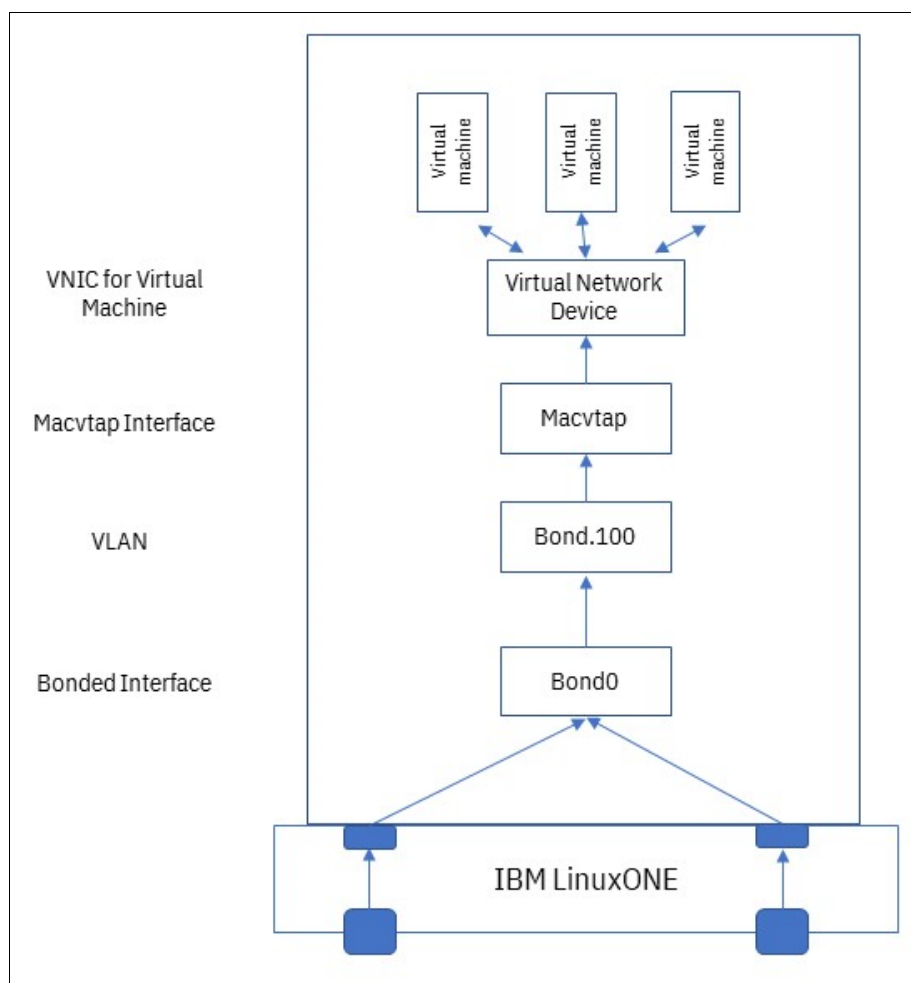


Figure 3-4 MacVTap with VLAN configuration

Note: For more information about the available KVM-based MacVTap networking options, see this IBM Documentation [web page](#).

In our environment, we used MacVTAP to create basic bridging capabilities for the host and cluster node.

3.1.3 Virtual image format

In KVM, VM images can be stored in different formats. The most straightforward option is to use the RAW format, where I/O requests to the virtual disk are served by way of simple block-to-block address mapping. Also, RAW format is faster because it has no associated metadata.

Another notable format is Quick Emulator, Copy On Write (QCOW2), in which a separate file is created to store all data blocks that were modified by the provisioned VM. QCOW2 features layers of indirection that must be crossed before it hits the data.

From a production cluster design, it is suggested to have RAW format from a performance and efficiency perspective. However, QCOW2 can be used for test and development environments in which agility and quick provisioning is a requirement. In our environment, we used a QCOW2-based virtual image.

3.2 High-level Red Hat OpenShift installation steps

An Red Hat OpenShift cluster consists of the following nodes:

- ▶ Master nodes form the control plane
- ▶ Infrastructure (bastion) nodes cater to a routing layer and other functions, such as logging and monitoring.
- ▶ Worker nodes are the nodes on which the application container workloads exist.

Red Hat OpenShift installer provides the following options for deploying the cluster on the underlying LinuxONE infrastructure:

- ▶ Installer-provisioned infrastructure (IPI)
- ▶ User-provisioned infrastructure (UPI)

We used the User-provisioned Infrastructure method for setting up our test environment for this publication (see Figure 3-5).

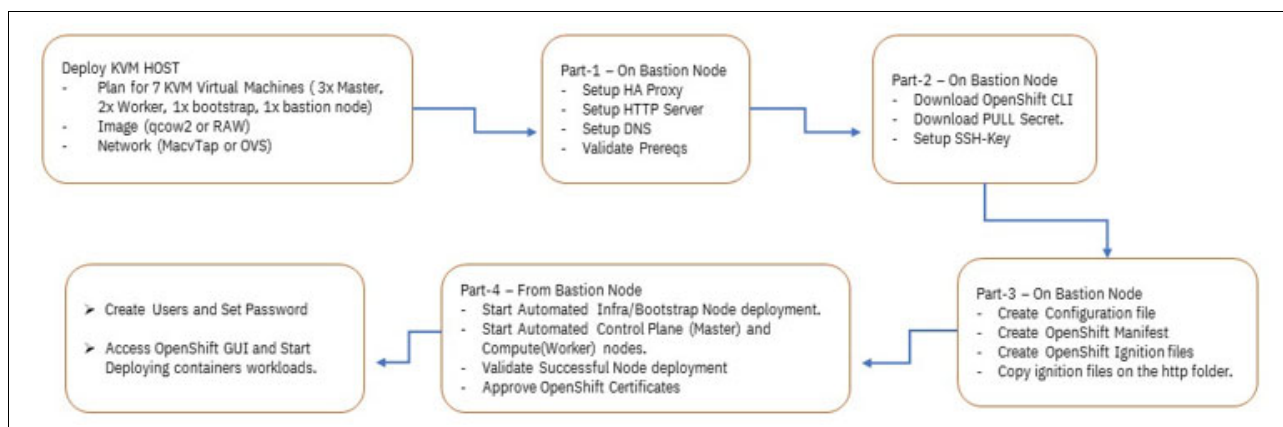


Figure 3-5 High-level Red Hat OpenShift installation steps

It is also recommended that you create a checklist (see Figure 3-6) to track all of the prerequisites cluster details for a setting up the host and cluster nodes. We found this practice to be useful while we set up the overall Red Hat OpenShift cluster, especially for HAProxy, DNS, and other network configurations.

KVM	Distro	IFL	Memory (GB)	Disk	Hostname	IP Address	Subnet	GW	Credentials	network interface	
HOST	RHEL 8.3			260GB	rhelvm.pbm.lhost.com	129.40.119.102	255.255.255.192	129.40.119.126	root	enc3617	
	HOSTNAME	vCPU	Memory (GB)	OSA(Shared 10GBe)	Disk	IP Address	Subnet	GW	Domain	FQDN	Clustername
	bootstrap			enc3610	/Images (in host 16GB)	129.40.119.95	255.255.255.192	129.40.119.126	test.ocpbm	bootstrap.ocp.test.ocpbm	ocp
	bastion			enc3610	/mpatha1	129.40.119.96	255.255.255.192	129.40.119.126	test.ocpbm	bastion.ocp.test.ocpbm	ocp
	master1			enc3610	140GB	129.40.119.97	255.255.255.192	129.40.119.126	test.ocpbm	master1.ocp.test.ocpbm	ocp
	master2			enc3610	Each 35GB	129.40.119.98	255.255.255.192	129.40.119.126	test.ocpbm	master2.ocp.test.ocpbm	ocp
	master3			enc3610	qcow2	129.40.119.99	255.255.255.192	129.40.119.126	test.ocpbm	master3.ocp.test.ocpbm	ocp
	worker1			enc3610	/mpathb1 (RAW)	129.40.119.100	255.255.255.192	129.40.119.126	test.ocpbm	worker1.ocp.test.ocpbm	ocp
	worker2			enc3610	/mpathc1 (RAW)	129.40.119.101	255.255.255.192	129.40.119.126	test.ocpbm	worker2.ocp.test.ocpbm	ocp

Figure 3-6 Cluster checklist

3.3 Red Hat OpenShift Container Platform prerequisites

In this section, we describe the perquisites for a successful deployment of a Red Hat OpenShift Container Platform on KVM.

3.3.1 KVM HOST configuration

For the KVM host, we installed Red Hat Enterprise Linux v8.3 with virtualization (KVM) packages enabled. Enabling virtualization packages to be deployed during installation is recommended because the overall dependencies for libvirt and its subsystems are automatically addressed.

We used QCOW2 for the Control Plane, Infra (Bastion) node, and the bootstrap nodes. The Compute nodes are placed in RAW format, directly on separate LUNs as RAW format provides better efficiency and performance.

Similarly, we assigned two network ports 3617 (for the KVM host) and 3610 (for the cluster nodes). The KVM host node (port 3617) features a static IP address, as shown in Figure 3-6.

3.3.2 MacVTap configuration

In the KVM host node of our lab environment, we assigned network port 3610 for cluster connectivity. Because this port is used as a base for the cluster node routing, we enabled the network port without any specific network routing or IP capability.

Example 3-1 shows our network configuration file.

Example 3-1 Network Device 3610 configuration file in host KVM

```
[root@kvmhost network-scripts]# cat ifcfg-enc3610
SUBCHANNELS=0.0.3610,0.0.3611,0.0.3612
NETTYPE=qeth
PORTNAME=DUMMY
TYPE=Ethernet
BOOTPROTO=static
```

```

IPADDR=
PREFIX=26
NAME=enc3610
UUID=a3060fe5-aa85-4660-9800-3ff90513ddce
DEVICE=enc3610
ONBOOT=yes
MULTI_CONNECT=1
OPTIONS="layer2=1 buffer_count=128"

```

Important: In the KVM host, the cluster nodes network configuration must not be assigned with GATEWAY, DNS, and IP address.

Libvirt automatically shows an interface when you define the network device. Example 3-2 shows the definition for the MacVTap bridge. The uuid is updated by libvirt when the port is enabled.

Example 3-2 MacVTap definition

```

[root@kvmhost ~]# cat ocp-macvtap-net
<network>
  <name>ocp-macvtap-net</name>
  <uuid>f7de70b2-2ea6-4161-ba6e-c68328b75afd</uuid>
  <forward dev='enc3610' mode='bridge'>
    <interface dev='enc3610' />
  </forward>
</network>

```

After the definition file is created, as shown in Example 3-2, MacVTap must be defined with the libvirt and enabled. Example 3-3 provides the commands that were used for defining and enabling MacVTap.

Example 3-3 Macvtap definition and enabling by using virsh

```

virsh # net-list
Name      State    Autostart  Persistent
-----
default   active   yes         yes

virsh # net-start ocp-macvtap-net

Network ocp-macvtap-net started

virsh # net-list
Name      State    Autostart  Persistent
-----
default   active   yes         yes
ocp-macvtap-net  active   no          yes

virsh # net-autostart ocp-macvtap-net
Network ocp-macvtap-net marked as autostarted

virsh # net-list
Name      State    Autostart  Persistent
-----
default   active   yes         yes
ocp-macvtap-net  active   yes         yes

```

With the configurations completed, we can move on to the next activity to get the Infrastructure (Bastion) node to be deployed.

3.3.3 Infrastructure (Bastion) Node deployment and prerequisites

This dedicated node serves as the main deployment and management server for the Red Hat OpenShift cluster. It is used as the log-on node for the cluster administrators to perform the system deployment and management operations. Because a test environment is used, we also decided to deploy the Infrastructure node to act as a web server, load balancer, and DNS routing node.

We used the **libvirt (virsh)** command line to define and install the infrastructure node with Red Hat Enterprise Linux 8.3 and its prerequisites. Example 3-4 shows the KVM definition file for the infrastructure node, which includes the MacVTap network definition.

Example 3-4 Infra node KVM Config definition

```
<domain type='kvm'>
  <name>rhelvm</name>
  <uuid>0736112f-1767-4062-bbef-c9373fbc0333</uuid>
  <memory unit='KiB'>8388608</memory>
  <currentMemory unit='KiB'>8388608</currentMemory>
  <vcpu placement='static'>2</vcpu>
  <iothreads>1</iothreads>
  <os>
    <type arch='s390x' machine='s390-ccw-virtio-rhel8.2.0'>hvm</type>
    <boot dev='hd'>/>
  </os>
  <cpu mode='host-model' check='partial'>/>
  <clock offset='utc'>/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>preserve</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' io='native' iotread='1'>/>
      <source file='/images/rhelvm.img'>/>
      <target dev='vda' bus='virtio'>/>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000'>/>
    </disk>
    <controller type='pci' index='0' model='pci-root'>/>
    <controller type='scsi' index='0' model='virtio-scsi'>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0002'>/>
    </controller>
    <interface type='network'>
      <mac address='52:54:00:84:1a:56'>/>
      <source network='ocp-macvtap-net'>/>
      <model type='virtio'>/>
      <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0001'>/>
    </interface>
    <console type='pty'>
      <target type='sclp' port='0'>/>
    </console>
    <memballoon model='none'>/>
  </devices>
</domain>
```

```
<panic model='s390' />
</devices>
</domain>
```

Note: The Infrastructure node acts as a DNS routing node for the Red Hat OpenShift cluster. Therefore, care must be taken when the correct gateway and DNS are identified, as shown in Example 3-5.

Example 3-5 infra node network and resolve configuration

```
[root@pbmbas network-scripts]# cat ifcfg-enc1
TYPE=Ethernet
BOOTPROTO=none
NAME=enc1
UUID=55e9b215-db26-4cfb-a4d4-eb047846bbfe
DEVICE=enc1
ONBOOT=yes
IPADDR=129.40.119.96
PREFIX=26
GATEWAY=129.40.119.126
MULTI_CONNECT=1
DEVTIMEOUT=60
```

```
[root@pbmbas etc]# cat resolv.conf
# Generated by NetworkManager
search ocp.test.ocpbm
nameserver 129.40.119.96
```

Validate the network configuration by pinging from the infrastructure node to the KVM host back and forth. After that process successfully completes, you can install the following packages:

- ▶ HTTP Server → Web Server
- ▶ haproxy → Load balancer
- ▶ bind utils → DNS Server

3.3.4 Setting up and configuring HAProxy

Red Hat OpenShift Container Platform requires specific DNS records for forward and reverse name resolution for a user provisioned infrastructure (UPI). Per the Red Hat OpenShift prerequisites, we created two DNS record files (forward and reverse) and included them in the `named.conf` file for resolution.

These two files must be created under the `/etc/named` directory and a copy of the same file must be placed under the `/var/named` directory. Example 3-6 shows the file with its entries for the forward DNS records.

Example 3-6 Forward zone DNS records

```
[root@pbmbas named]# cat ocp.db
$TTL 86400
@ IN SOA ns1.ocp.test.ocpbm. admin.ocp.test.ocpbm. (
                                2020021825 ;Serial
                                3600 ;Refresh
                                1800 ;Retry
```

```

        604800 ;Expire
        86400 ;Minimum TTL
)

```

```

;Name Server Information
@ IN NS ns1.ocp.test.ocpbm.

```

```

;IP Address for Name Server
ns1 IN A 129.40.119.96

```

```

;A Record for the following Host name

```

```

haproxy      IN  A   129.40.119.96
pbmbas       IN  A   129.40.119.96
bastion      IN  A   129.40.119.96

```

```

bootstrap    IN  A   129.40.119.95
master1      IN  A   129.40.119.97
master2      IN  A   129.40.119.98
master3      IN  A   129.40.119.99

```

```

worker1      IN  A   129.40.119.100
worker2      IN  A   129.40.119.101

```

```

;CNAME Record

```

```

api          IN CNAME haproxy.ocp.test.ocpbm.
api-int      IN CNAME haproxy.ocp.test.ocpbm.
*.apps       IN CNAME haproxy.ocp.test.ocpbm.

```

Reverse zone DNS records are also important because Red Hat OpenShift uses the reverse records to set the host name for all the nodes. Therefore, another file for reverse DNS was created (see Example 3-7).

Example 3-7 Reverse DNS Records

```

[root@pbmbas named]# cat ocp.rev
$TTL 86400
@ IN SOA ns1.ocp.test.ocpbm. admin.ocp.test.ocpbm. (
                                2020021827 ;Serial
                                3600 ;Refresh
                                1800 ;Retry
                                604800 ;Expire
                                86400 ;Minimum TTL
)
;Name Server Information
@ IN NS ns1.ocp.test.ocpbm.
ns1      IN      A       129.40.119.96

;Reverse lookup for Name Server
96 IN PTR ns1.ocp.test.ocpbm.

;PTR Record IP address to Hostname
96      IN      PTR     haproxy.ocp.test.ocpbm.

```

95	IN	PTR	bootstrap.ocp.test.ocpbm.
97	IN	PTR	master1.ocp.test.ocpbm.
98	IN	PTR	master2.ocp.test.ocpbm.
99	IN	PTR	master3.ocp.test.ocpbm.
100	IN	PTR	worker1.ocp.test.ocpbm.
101	IN	PTR	worker2.ocp.test.ocpbm.
96	IN	PTR	pbmbas.ocp.test.ocpbm.

Setting up the forward and reverse DNS records plays a crucial role during the Red Hat OpenShift installation because the Red Hat OpenShift bootstrap looks up the name resolution of the Control Plane and Compute nodes. Therefore, double check and validate the IP addresses and the domain name resolutions from your prepared cluster checklist, as shown in Figure 3-6 on page 61.

After the IP address is validated, the zone details must be included in the `named.conf` file. Also, some of the default parameters must be updated in the file, as shown in Figure 3-7.

```
[root@pbmbas etc]# cat named.conf

options {
    listen-on port 53 { 127.0.0.1; 129.40.119.96; };
    listen-on-v6 port 53 { ::1; };
    directory "/var/named";
    dump-file "/var/named/data/cache_dump.db";
    statistics-file "/var/named/data/named_stats.txt";
    memstatistics-file "/var/named/data/named_mem_stats.txt";
    secroots-file "/var/named/data/named.secroots";
    recursing-file "/var/named/data/named.recursing";
    allow-query { localhost; 129.40.119.0/26; };
    allow-recursion { any; };
    forwarders { 129.40.119.126; };
};

recursion yes;

dnssec-enable yes;
dnssec-validation yes;

:
:

//Added Openshift Entries
//forward zone
zone "ocp.test.ocpbm" IN {
    type master;
    file "/var/named/ocp.db";
    allow-update { none; };
    allow-query { any; };
};
//reverse zone
zone "119.40.129.in-addr.arpa" IN {
    type master;
    file "/var/named/ocp.rev";
    allow-update { none; };
    allow-query { any; };
};
```

Mention cluster IP and subnet

Mention Host nameserver IP as Openshift would connect to external site to pull packages.

Point to the right forward and reverse zone DNS records file.

Figure 3-7 Updated `named.conf` file

The following parameters must be updated with their descriptions:

- ▶ **allow-query:** This parameter allows clients to query the DNS for the name (URL) to IP translation. The cluster subnet details must be included.
- ▶ **forwarders:** List of valid IP addresses for name servers where requests must be forwarded for resolution. Provide the name server IP address of the KVM host for resolution and external connectivity.

Now that the `named.conf` file is updated with the changes and zones, we can start the DNS Server by using the commands that are shown in Example 3-8.

Example 3-8 Starting named services

```
[root@pbmbas ~]# systemctl start named.service
[root@pbmbas ~]# systemctl status named.service
? named.service - Berkeley Internet Name Domain (DNS)
   Loaded: loaded (/usr/lib/systemd/system/named.service; disabled; vendor preset:
disabled)
   Active: active (running) since Fri 2021-07-09 23:58:12 EDT; 3s ago
     Process: 1692 ExecStart=/usr/sbin/named -u named -c ${NAMEDCONF} $OPTIONS
(code=exited, status=0/SUCCESS)
    Process: 1690 ExecStartPre=/bin/bash -c if [ ! "$DISABLE_ZONE_CHECKING" == "yes"
]; then /usr/sbin/named-checkconf -z "$NAMEDCONF"; else echo "Checking of>
Main PID: 1694 (named)
   Tasks: 5 (limit: 50034)
  Memory: 63.9M
   CGroup: /system.slice/named.service
           +-1694 /usr/sbin/named -u named -c /etc/named.conf
```

After the `named` services are successfully started, validate that the DNS is working correctly by using the **nslookup** command to check the name resolution for any of the cluster node names, as shown in Example 3-9.

Example 3-9 Checking on DNS working properly

```
[root@pbmbas ~]# nslookup master1
Server:           129.40.119.96
Address:          129.40.119.96#53

Name:   master1.ocp.test.ocpbm
Address: 129.40.119.97
```

3.4 Deploying HAProxy

HAProxy is a popular Open Source load-balancing software, which also offers high availability and proxy function. We used HAProxy as our load balancer for the Red Hat OpenShift cluster.

Red Hat Enterprise Linux packaged HAProxy with the distribution; therefore, we installed HAProxy by using our NFS-based local repository. After HAProxy is successfully installed, you must update the `haproxy.conf` file under the `/etc/haproxy` directory by using the port bindings for Red Hat OpenShift cluster access points, such as `api`, `ingress-http`, `ingress-https`, and `api-init`.

Example 3-10 shows the entries that must be added to the `haproxy.conf` file.

Example 3-10 Entries that must be added to haproxy.conf

```
#-----
# -- Added Red Hat OpenShift Entries --
#-----

listen ingress-http

    bind *:80
    mode tcp

    server worker1 129.40.119.100:80 check
    server worker2 129.40.119.101:80 check

listen ingress-https

    bind *:443
    mode tcp

    server worker1 129.40.119.100:443 check
    server worker2 129.40.119.101:443 check

listen api

    bind *:6443
    mode tcp

    server bootstrap 129.40.119.95:6443 check
    server master1 129.40.119.97:6443 check
    server master2 129.40.119.98:6443 check
    server master3 129.40.119.99:6443 check

listen api-int

    bind *:22623
    mode tcp

    server bootstrap 129.40.119.95:22623 check
    server master1 129.40.119.97:22623 check
    server master2 129.40.119.98:22623 check
    server master3 129.40.119.99:22623 check
```

Start the HAProxy by using the **systemctl** command, as shown in Example 3-11.

Example 3-11 Starting HAProxy

```
[root@pbmbas ~]# systemctl start haproxy
[root@pbmbas ~]# systemctl status haproxy
? haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/usr/lib/systemd/system/haproxy.service; disabled; vendor
   preset: disabled)
   Active: active (running) since Fri 2021-07-09 23:56:46 EDT; 2s ago
     Process: 1408 ExecStartPre=/usr/sbin/haproxy -f $CONFIG -c -q $OPTIONS
   (code=exited, status=0/SUCCESS)
    Main PID: 1409 (haproxy)
       Tasks: 2 (limit: 50034)
      Memory: 5.0M
     CGroup: /system.slice/haproxy.service
            +-1409 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p
            /run/haproxy.pid
            +-1411 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p
            /run/haproxy.pid
Jul 09 23:56:46 pbmbas systemd[1]: Started HAProxy Load Balancer.
```

After it is confirmed that the HAProxy successfully started, validate the successful port bindings and that they are in listening mode. Use the **netstat** command (see Example 3-12) to verify successful port bindings with HAProxy.

Example 3-12 Using netstat to verify that the ports are in listen mode

```
[root@pbmbas ~]# netstat -apnt
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 129.40.119.96:53        0.0.0.0:*               LISTEN
1969/named
tcp        0      0 127.0.0.1:53            0.0.0.0:*               LISTEN
1969/named
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
900/sshd
tcp        0      0 127.0.0.1:953           0.0.0.0:*               LISTEN
1969/named
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
2015/haproxy
tcp        0      0 0.0.0.0:22623           0.0.0.0:*               LISTEN
2015/haproxy
tcp        0      0 0.0.0.0:6443            0.0.0.0:*               LISTEN
2015/haproxy
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
1/systemd
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
2015/haproxy
```

3.4.1 HTTP server configuration

Installing and configuring the HTTP server is straight forward. Because the HAPRoxy used port 80, the http servers listening port to other web server ports, such as 8080 or 8081, must be updated. Therefore, we updated the `httpd.conf` file to listen at port 8080 instead of the default port 80 and then, we started the http server.

We now completed the first part of meeting the Red Hat OpenShift prerequisites by configuring HAProxy, the DNS service, and the web server that is to be used during the installation of Red Hat OpenShift.

3.4.2 Red Hat OpenShift cluster CLI and certificate generation

On the Infrastructure node, download the Red Hat OpenShift command line interface (CLI), Red Hat OpenShift initial install files, and the pull secret files from the Red Hat OpenShift Container Platform [cluster manager web page](#) (log in required). After the files are downloaded, extract the Red Hat OpenShift CLI executables into a folder. In our example, we created a folder that is named `/install` and placed the CLI executables in it.

We also generated a ssh-key by using the `ssh-keygen -t ed25519 -f <folder>` command. A ssh-key (public) is generated that is passed on with the `install-config.yaml` file.

Also extract the initial installation file (`openshift-install-linux-4.7.18.tar.gz`) into the httpd access directory named `/var/www/http/`.

Note: Red Hat OpenShift initial installation files are available at this [web page](#).

3.4.3 Creating the installation configuration file

Next, you create an `install-config.yaml` file in which information is provided about the cluster, pull secret, and the ssh-key for generating the manifest and ignition files for Red Hat OpenShift cluster automated installation.

An example of our `install-config.yaml` file is shown in Figure 3-8.

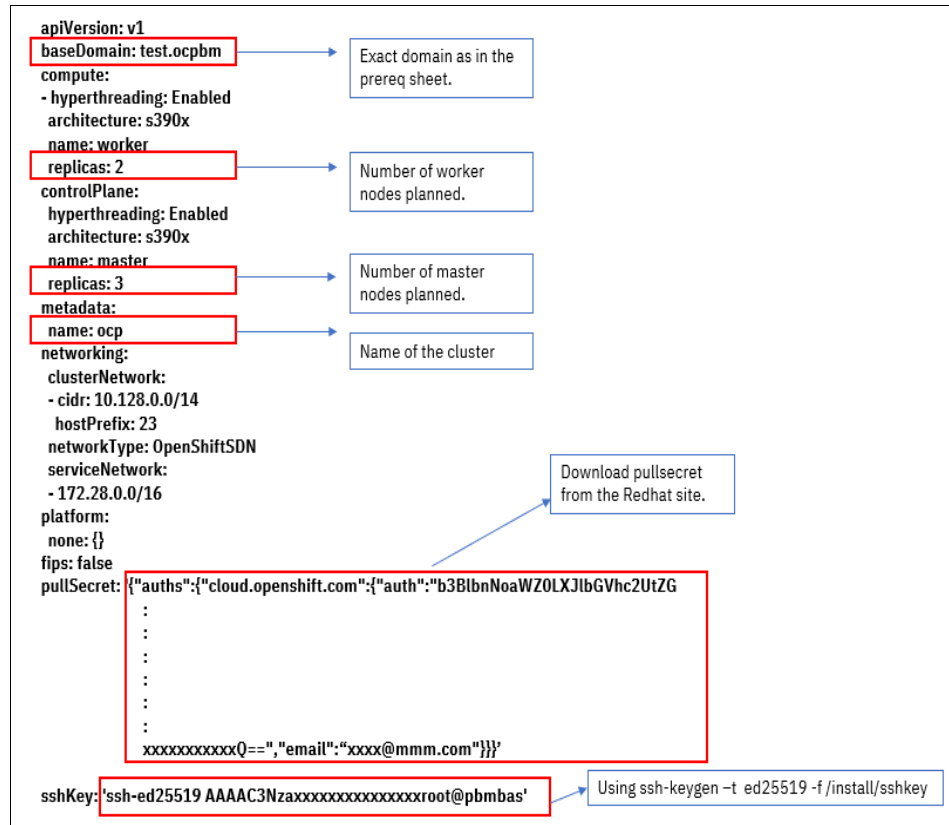


Figure 3-8 Sample `install-config.yaml`

Note: The `install-config.yaml` is removed automatically after the commands are issued to generate the manifest and edition files. It is recommended to back up the `install-config.yaml` file before completing the next steps.

3.4.4 Generating manifest and ignition files

Red Hat OpenShift CLI commands are issued to generate the manifest and the ignition files, as shown in Example 3-13.

Example 3-13 Creating manifest and ignition files

```
[root@pbmbas ]# openshift-install create manifests --dir=/ISO
INFO Consuming Install Config from target directory
INFO Manifests created in: /install/manifests and /install/openshift
[root@pbmbas ]#

[root@pbmbas ]# ./openshift-install create ignition-configs --dir=/install
INFO Consuming Red Hat OpenShift Manifests from target directory
INFO Consuming Common Manifests from target directory
INFO Consuming Worker Machines from target directory
INFO Consuming Red Hat OpenShift Install (Manifests) from target directory
INFO Consuming Master Machines from target directory
INFO Ignition-Configs created in: /install and /install/auth
[root@pbmbas ]#
```

Copy the bootstrap, master, and worker ignition files into the http folder path. Example 3-14 shows the files that must be placed in the http access folder named `/var/www/html`.

Example 3-14 Files under /var/www/html folder

```
[root@pbmbas html]# ls -la
total 816476
drwxr-xr-x. 2 root root      227 Jul 11 05:49 .
drwxr-xr-x. 4 root root       33 Jul  7 01:21 ..
-rwxrwxrwx. 1 root root 288928 Jul 11 05:49 bootstrap.ign
-rwxrwxrwx. 1 root root  1716 Jul 11 05:49 master.ign
-rwxrwxrwx. 1 root root 67327876 Jul  8 10:13
rhcos-4.7.13-s390x-live-initramfs.s390x.img
-rwxrwxrwx. 1 root root  6750693 Jul  8 10:48
rhcos-4.7.13-s390x-live-kernel-s390x
-rwxrwxrwx. 1 root root 761682944 Jul  8 10:13
rhcos-4.7.13-s390x-live-rootfs.s390x.img
-rwxrwxrwx. 1 root root    1716 Jul 11 05:49 worker.ign
```

3.4.5 Installing Red Hat OpenShift cluster

After the manifest and ignition files are generated, you can start deploying the cluster nodes. The first node that must be deployed is the bootstrap node. This node facilitates the successful installation of the Control Plane (master) and the worker (Compute) nodes.

From the KVM host, run the **virt-install** command with parameters that provide details about the network and the ignitions files. Example 3-15 shows the **virt-install** CLI command that was used to deploy the cluster.

Example 3-15 Bootstrap virt-install command

```
virt-install --connect qemu:///system --name bootstrap --vcpus 4 --memory 16384
--disk /mstdisk/bootstrap.qcow2,size=30 --network network=ocp-macvtap-net --boot
hd --location /install,
kernel=rhcos-4.7.13-s390x-live-kernel-s390x,initrd=rhcos-4.7.13-s390x-live-initram
fs.s390x.img --extra-args "rd.neednet=1 dflttcc=off coreos.inst=yes
coreos.inst.install_dev=vda
coreos.live.rootfs_url=http://129.40.119.96:8080/rhcos-4.7.13-s390x-live-rootfs.s3
90x.img ip=129.40.119.95::129.40.119.126:26:bootstrap:encl:none:1500
nameserver=129.40.119.96
coreos.inst.ignition_url=http://129.40.119.96:8080/bootstrap.ign" --noautoconsole
--wait
```

By running the command that is shown in Example 3-15, **virt-install** connects to the Bastion node and reads the ignition files as and the initial install files to deploy the bootstrap nodes. The installation is shown in Example 3-16.

Example 3-16 executing virt-install to deploy bootstrap node.

```
Starting install...
Retrieving file rhcos-4.7.13-s390x-live-kernel-s390x... | 6.4 MB 00:00:00
Retrieving file rhcos-4.7.13-s390x-live-initramfs.s390x.img... | 64 MB 00:00:00
Allocating 'bootstrap.qcow2' | 20 GB 00:00:00
Domain installation still in progress.
Waiting for installation to complete.
Domain has shutdown. Continuing.
Domain creation completed.
Restarting guest.
```

After the bootstrap node deployment completes successfully, run the **virt-install** commands for the Control Plane (master) and Compute (worker) nodes by changing the network and ignition files.

Example 3-17 shows a sample of the **virt-install** commands.

Example 3-17 Sample Control Plane (master) and Compute (worker) virt-install commands

```
virt-install --connect qemu:///system --name master1 --vcpus 4 --memory 16384
--disk /mstdisk/master1.qcow2,size=30 --network network=ocp-macvtap-net --boot hd
--location
/ISO,kernel=rhcos-4.7.13-s390x-live-kernel-s390x,initrd=rhcos-4.7.13-s390x-live-in
itramfs.s390x.img --extra-args "rd.neednet=1 dflttcc=off coreos.inst=yes
coreos.inst.install_dev=vda
coreos.live.rootfs_url=http://129.40.119.96:8080/rhcos-4.7.13-s390x-live-rootfs.s3
90x.img ip=129.40.119.97::129.40.119.126:26:master1:encl:none:1500
nameserver=129.40.119.96
```

```
coreos.inst.ignition_url=http://129.40.119.96:8080/master.ign" --noautoconsole
--wait

virt-install --connect qemu:///system --name worker1 --vcpus 4 --memory 16384
--disk /wrkdisk1/worker1.qcow2,size=30 --network network=ocp-macvtap-net --boot hd
--location
/ISO,kernel=rhcos-4.7.13-s390x-live-kernel-s390x,initrd=rhcos-4.7.13-s390x-live-in
itramfs.s390x.img --extra-args "rd.neednet=1 dfltcc=off coreos.inst=yes
coreos.inst.install_dev=vda
coreos.live.rootfs_url=http://129.40.119.96:8080/rhcos-4.7.13-s390x-live-rootfs.s3
90x.img ip=129.40.119.100::129.40.119.126:26:worker1:encl:none:1500
nameserver=129.40.119.96
coreos.inst.ignition_url=http://129.40.119.96:8080/worker.ign" --noautoconsole
--wait
```

3.4.6 Validating Red Hat OpenShift cluster deployment

After all of the nodes are successfully deployed, you must validate whether all of the nodes are configured correctly. Complete the following steps:

1. Start with the host KVM and check to see if all of the Red Hat OpenShift cluster VMs and nodes are deployed (see Example 3-18).

Example 3-18 KVM host virtual images

```
virsh # list
Id   Name      State
-----
1    rhelvm    running
57   bootstrap running
59   master1   running
61   master2   running
63   master3   running
65   worker1   running
67   worker2   running
```

2. From the Infrastructure node, run the **openshift-install** command (see Example 3-19) to validate that your cluster deployment is successful. This step most important because it is here where you determine whether the cluster is deployed and cluster APIs are responding.

Example 3-19 Openshift-install command to validate the cluster deployment.

```
[root@pbmbas ~]# openshift-install --dir=/install wait-for bootstrap-complete
--log-level=info
INFO Waiting up to 20m0s for the Kubernetes API at
https://api.ocp.test.ocpbm:6443...
INFO API v1.20.0+87cc9a4 up
INFO Waiting up to 30m0s for bootstrapping to complete...
INFO It is now safe to remove the bootstrap resources
INFO Time elapsed: 0s
[root@pbmbas ~]#
```

After you verify a successful deployment, remove the bootstrap node that is shown in the output in Example 3-19. We commented the bootstrap entry from the HAProxy so that nothing is routed to the bootstrap nodes. It is also important to check whether the nodes are operating when the Red Hat OpenShift CLI is used by running the command that is shown in Example 3-20.

Example 3-20 Red Hat OpenShift CLI to check nodes status

```
[root@pbmbas ~]# oc get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master1	Ready	master	94m	v1.20.0+87cc9a4
master2	Ready	master	80m	v1.20.0+87cc9a4
master3	Ready	master	76m	v1.20.0+87cc9a4
worker1	Ready	worker	33m	v1.20.0+87cc9a4
worker2	Ready	worker	33m	v1.20.0+87cc9a4

3.4.7 Approving Red Hat OpenShift certificates

When you run the **openshift-install** command to validate the deployment status, start approving the certificates from another infrastructure node **ssh** session. The certificates can be approved by using the command that is shown in Example 3-19 on page 74.

By using the Red Hat OpenShift command **oc get csr**, all of the pending certificates that must be approved are listed. Until all of the certificates are approved, Red Hat OpenShift cluster objects are *not* listed as available.

Example 3-21 Check approval of certificates

```
oc get csr --no-headers | awk '{print $1}' | xargs oc adm certificate approve
```

To verify whether all of the cluster objects certificates are approved and are in available status, use the command `oc get clusteroperators` as shown in Figure 3-9.

```
[root@pbmbas ~]# oc get clusteroperators
```

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
authentication	4.7.18	True	False	False	3d6h
baremetal	4.7.18	True	False	False	10d
cloud-credential	4.7.18	True	False	False	10d
cluster-autoscaler	4.7.18	True	False	False	10d
config-operator	4.7.18	True	False	False	10d
console	4.7.18	True	False	False	10d
csi-snapshot-controller	4.7.18	True	False	False	10d
dns	4.7.18	True	False	False	10d
etcd	4.7.18	True	False	False	10d
image-registry	4.7.18	True	False	False	10d
ingress	4.7.18	True	False	False	10d
insights	4.7.18	True	False	True	10d
kube-apiserver	4.7.18	True	False	False	10d
kube-controller-manager	4.7.18	True	False	False	10d
kube-scheduler	4.7.18	True	False	False	10d
kube-storage-version-migrator	4.7.18	False	False	False	16h
machine-api	4.7.18	True	False	False	10d
machine-approver	4.7.18	True	False	False	10d
machine-config	4.7.18	True	False	False	3d6h
marketplace	4.7.18	True	False	False	10d
monitoring	4.7.18	True	False	False	11h
network	4.7.18	True	True	False	10d
node-tuning	4.7.18	True	False	False	10d
openshift-apiserver	4.7.18	True	False	False	3d6h
openshift-controller-manager	4.7.18	True	False	False	3d6h
openshift-samples	4.7.18	True	False	False	10d
operator-lifecycle-manager	4.7.18	True	False	False	10d
operator-lifecycle-manager-catalog	4.7.18	True	False	False	10d
operator-lifecycle-manager-packageserver	4.7.18	True	False	False	3d6h
service-ca	4.7.18	True	False	False	10d
storage	4.7.18	True	False	False	10d

Figure 3-9 cluster objects in available status.

You now can log in to the Red Hat OpenShift GUI by using the temporary **kubeadmin** user ID and password. The next steps are create a permanent user ID and password by using the identity manager option in the GUI console.

The Red Hat OpenShift cluster is now deployed successfully and waiting for container workloads to be deployed on the nodes.



Use Case: Cloud Pak for Data platform that is running containerized Db2 service

In this chapter, we discuss a use case that involves an IBM Cloud Pak for Data platform that is running a containerized Db2 service.

This chapter includes the following topics:

- ▶ 4.1, “Overview” on page 78
- ▶ 4.2, “Architecture” on page 79
- ▶ 4.3, “Solution implementation” on page 80
- ▶ 4.4, “Best practices” on page 87

4.1 Overview

Databases are centric to all types of enterprises that are storing and processing an excessive amount of data, practically of any type, structured or nonstructured. The databases require a fast I/O subsystem to process the millions of transactions that are submitted from all types of applications. Often, the transaction response time is subject to a Service Level Agreement (SLA), which tend to be demanding.

IBM Z is suited for high-intensity data serving workloads. By having a high-throughput channel I/O subsystem, it ensures zero I/O wait. Several other IBM Z software and hardware enhancements facilitate fast data serving, including four levels of high-speed cache, the mechanism for data-prefetching, and high-speed FICON channels.

With the latest technological enhancements, IBM Z is an essential part of the Hybrid Cloud model. It provides a hosting platform for containerized databases and applications.

New cloud-born applications need to access and process the Systems of Record data from a colocated container environment, without the need to connect to the original data sources. The microservices approach provides ways to individually scale the databases and data services. The databases that are running in a containerized environment provide this agility and cloud experience to their users (containerized applications).

IBM Cloud Pak for Data is the platform that manages, processes, analyzes, and transforms data that comes from various sources, including on-premises and multi clouds.

On the IBM Z platform, Cloud Pak for Data provides the following services (as of this writing):

- ▶ DataGate
- ▶ Db2 AESE
- ▶ Db2 Warehouse
- ▶ Data Management Console
- ▶ Apache Analytics Spark Engine
- ▶ IBM Watson® Studio
- ▶ IBM Watson Machine Learning.

If you are running a Db2 z/OS environment, you can virtualize all of the data, including non-SQL data (VSAM, IMS, and so on) by using IBM Data Virtualization Manager (DVM) and then, selectively replicate read-only data by using the DataGate component into IBM Cloud Pak for Data Db2 or Db2 Warehouse containerized instances.

In this scenario, your cloud native applications (which also are containerized), can access the data without the need to access Db2 z/OS directly. Colocation removes the network latency and prevents the data from leaving the secure IBM Z perimeter.

The Cloud Pak for Data range of services provides the tools for further data analysis, collection, machine learning, and business analytics processing.

Cloud Pak for Data is a “Swiss army knife” for the modern data scientist. It allows you to transform, modernize, and integrate the data and cloud-ready applications with cloud agility and scale.

4.2 Architecture

Figure 4-1 shows the major components when Cloud Pak for Data is used for running a containerized Db2 service.

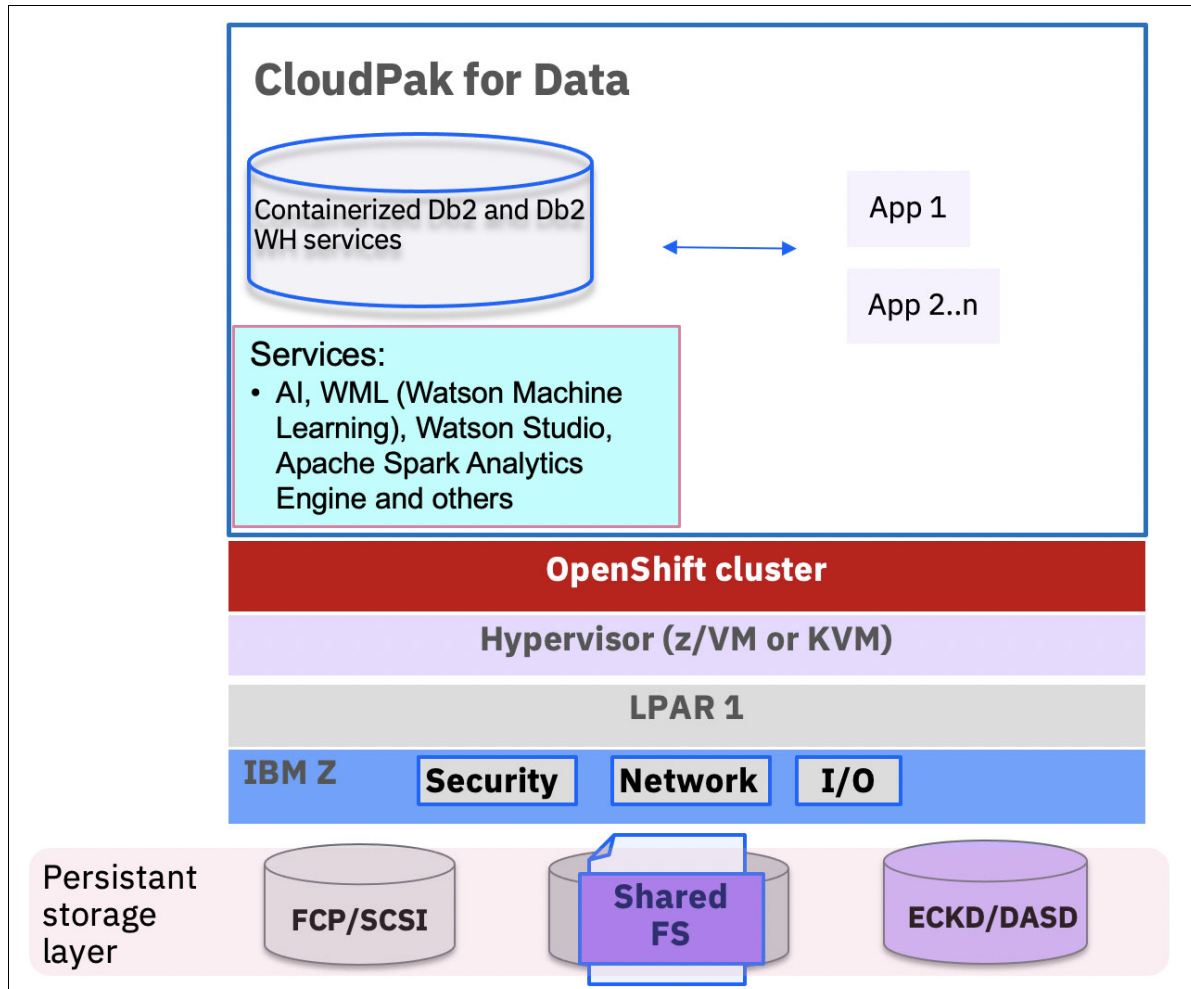


Figure 4-1 Running containerized Db2 services in Cloud Pak for Data

The architecture that is shown in Figure 4-1 features the following major components (from the top to the bottom of the figure):

► Applications layer

In this scenario, a sample IBM WebSphere® application is accessing Cloud Pak for Data Db2 Warehouse (Db2 WH) services. The application runs under OpenShift Container Platform to use the colocation and scalability.

► Cloud Pak for Data layer

The platform provides the core services and the other components (in our use case, Db2 AESE and Db2 DataWarehouse). Cloud Pak for Data runs on top of the Red Hat OpenShift Container Platform. All components run isolated in the containers, and can be scaled individually.

► Virtualization layer

This layer is provided by a unique combination of IBM Z hardware (LPAR technology) and software (the z/VM hypervisor). Alternatively, you can use KVM.

- On the storage level, the hardware disks are abstracted away by the software and can be ECKD/DASD or FCP/SCSI. The persistent storage layer is needed for Cloud Pak for Data, and in our scenario is provided by IBM Spectrum Scale Container Native Storage Access (CNSA) and Container Storage Interface (CSI). These components allow the container workloads to use IBM Spectrum Scale cluster storage by way of Persistent Volume Claims (PVCs). Alternatively, you can use Red Hat OpenShift Data Foundation (ODF).

NFS setup is *not* recommended for the production.

- IBM Spectrum Scale storage cluster is a cluster file system that provides a concurrent access to a single file system or a set of file systems from multiple nodes. The Spectrum Scale cluster is highly reliable and ensures the data availability by way of the replication between the nodes of the cluster.

Spectrum Scale cluster can run on IBM Z, IBM Power, or x86 platforms and can be external to the Red Hat OpenShift cluster. In our scenario, a basic single node Spectrum Scale cluster was deployed that is running on Linux on Z.

IBM Spectrum Scale is a part of IBM Storage Suite, which is a package of solutions for Hybrid Multi Cloud environment.

4.3 Solution implementation

In our lab environment, we set up a z/VM 7.2 LPAR with an Red Hat OpenShift 4.8 cluster that hosts Cloud Pak for Data (v.4.0.3) and our sample application.

Another instance of Red Hat Enterprise Linux (running under z/VM) was used for the Spectrum Scale file system. Two Spectrum Scale components (CNSA and CSI) allow containerized workloads to request, create, and manage the persistent volumes.

We used the following CNSA command to verify the Spectrum Scale storage cluster state:

```
[root@rdbkbas6 ~]# oc exec ibm-spectrum-scale-core-bkhwk -n ibm-spectrum-scale-ns -- mm1sc1uster
```

The results of this command are shown in Example 4-1.

Example 4-1 GPFS cluster information

```
GPFS cluster name:      ibm-spectrum-scale.ibm-spectrum-scale-ns.cluster.local
GPFS cluster id:       4980302168053280719
GPFS UID domain:       ibm-spectrum-scale.ibm-spectrum-scale-ns.cluster.local
Remote shell command:  /usr/bin/ssh
Remote file copy command: /usr/bin/scp
Repository type:       CCR
Node Daemon node name IP address Admin node name Designation
-----
1  rdbko6w0      9.76.61.55 rdbko6w0      quorum-manager-perfmon
2  rdbko6w1      9.76.61.56 rdbko6w1      quorum-manager-perfmon
3  rdbko6w2      9.76.61.57 rdbko6w2      quorum-manager-perfmon
4  rdbko6w3      9.76.61.58 rdbko6w3      manager-perfmon
```

To check the file system, we used the command that is shown in Example 4-2. The results of the command also are shown.

Example 4-2 Checking the file system

```
[root@rdbkbas6 ~]# oc exec ibm-spectrum-scale-core-bkhwk -n ibm-spectrum-scale-ns
-- mmremotefs show
```

Local Name	Remote Name	Cluster name	Mount Point	Mount Options
Automount	Drive	Priority		
sssl	sssl	cluster1.cpolab.ibm.com	/mnt/sssl	rw
yes	-	0		

Because we successfully deployed the persistent storage level, we can deploy Cloud Pak for Data. We have used a separate namespace for all of its components. The command is shown in Example 4-3, along with the results.

Example 4-3 Deploy Cloud Pak for Data

```
[root@rdbkbas5 ~]# oc get pods
```

NAME	READY	STATUS
RESTARTS	AGE	
asset-files-api-df6b5c4db-brjq6	1/1	Running
0	9d	
ax-cdsx-jupyter-notebooks-converter-deploy-7ff886b6c8-ggmtj	1/1	Running
2	27d	
ax-cdsx-notebooks-job-manager-deploy-86f4775f47-q9p97	1/1	Running
0	27d	
ax-environments-api-deploy-c8c6b4f7b-rnmbb	1/1	Running
0	9d	
ax-environments-ui-deploy-5bb9b74849-w7h8c	1/1	Running
0	9d	
ax-wdp-notebooks-api-deploy-7d9657d6bd-vw7jh	1/1	Running
1	27d	
ax-ws-notebooks-ui-deploy-5cb78b9f49-b9ggt	1/1	Running
0	27d	
c-db2oltp-1638542422602654-db2u-0	1/1	Running
0	27d	
c-db2oltp-1638542422602654-etcd-0	1/1	Running
1	27d	
c-db2oltp-1638542422602654-instdb-xqtn9	0/1	Completed
0	27d	
c-db2oltp-1638542422602654-restore-morph-ml5td	0/1	Completed
0	27d	
c-db2wh-1638543810897370-db2u-0	0/1	Pending
0	27d	
c-db2wh-1638543810897370-etcd-0	1/1	Running
0	27d	
c-db2wh-1638543810897370-instdb-lpf2t	0/1	Completed
0	27d	
catalog-api-f4df56b57-4bx5n	1/1	Running
0	9d	
catalog-api-f4df56b57-nw2b1	1/1	Running
0	9d	
ccs-post-install-job-9rk7w	0/1	Completed
0	9d	

create-dap-directories-lcxjs	0/1	Completed
0 9d		
create-secrets-job-xxdjh	0/1	Completed
0 27d		
create-watson-studio-secrets-g4qrf	0/1	Completed
0 9d		
dap-base-extension-translations-zddqk	0/1	Completed
0 9d		
dap-dashboards-api-6fd8b9fc66-114w7	1/1	Running
0 9d		
dataview-api-service-c987947cf-wqpwb	1/1	Running
0 9d		
dc-main-7b9d5dcb99-1hqz2	1/1	Running
0 9d		
diagnostics-cronjob-27348110-qnjpr	0/1	Completed
0 21s		
dsx-influxdb-0	1/1	Running
0 27d		
dsx-influxdb-set-auth-ltwrt	0/1	Completed
0 27d		
dsx-requisite-pre-install-job-g4blt	0/1	Completed
0 9d		
dsx-userhome-pre-install-job-bh5dp	0/1	Completed
0 9d		
elasticsearch-master-0	3/3	Running
19 9d		
elasticsearch-master-1	3/3	Running
18 9d		
elasticsearch-master-2	3/3	Running
21 9d		
env-spec-sync-job-27348110-8hj6l	0/1	Completed
0 21s		
environments-init-job-q7bpc	0/1	Completed
0 9d		
event-logger-api-6b78b5b47-6ddbg	1/1	Running
0 9d		
ibm-0100-model-viewer-prod-79cb7ddf86-wv5wg	1/1	Running
0 27d		
ibm-nginx-648894b8b5-9szsw	1/1	Running
1 27d		
ibm-nginx-648894b8b5-h2jcn	1/1	Running
2 27d		
init-runtime-py38main-n5wm9	0/1	Completed
0 9d		
jobs-api-67dfd57d95-12qkq	1/1	Running
0 9d		
jobs-ui-65f447c94c-5qlxl	1/1	Running
0 9d		
jobs-ui-extension-translations-gkvqb	0/1	Completed
0 9d		
ngp-projects-api-57d984955c-dz9vc	1/1	Running
0 9d		
notebooks-init-job-dkxdn	0/1	Completed
0 9d		

portal-catalog-7559cb5778-xk8ss 0 9d	1/1	Running
portal-common-api-8b767bf45-9p484 0 9d	1/1	Running
portal-dashboards-8555bf65dc-s9nbv 0 9d	1/1	Running
portal-job-manager-fb546f96d-hvbk5 0 9d	1/1	Running
portal-main-6595659847-qjf9j 0 9d	1/1	Running
portal-ml-dl-5b884dd448-nrbxs 0 27d	1/1	Running
portal-notifications-8d4479b55-ddq5g 0 9d	1/1	Running
portal-projects-ffbd54b9b-zn9g6 0 9d	1/1	Running
post-install-upgrade-spaces-wml-global-asset-type-fwfvm 0 9d	0/1	Completed
preinstall-wml-secret-c4tvv 0 9d	0/1	Completed
preinstall-wml-secret-etcd-n6bgk 0 9d	0/1	Completed
projects-ui-refresh-users-swvm9 0 9d	0/1	Completed
rabbitmq-ha-0 0 9d	1/1	Running
rabbitmq-ha-1 0 9d	1/1	Running
rabbitmq-ha-2 0 9d	1/1	Running
redis-ha-haproxy-585d7f7db-zp9f5 0 9d	1/1	Running
redis-ha-server-0 0 9d	2/2	Running
redis-ha-server-1 0 9d	2/2	Running
redis-ha-server-2 0 9d	2/2	Running
setup-nginx-job-75xrz 0 27d	0/1	Completed
spaces-76ff58b97f-hpphh 0 9d	1/1	Running
spaces-ui-extension-translations-94521 0 9d	0/1	Completed
spaces-ui-refresh-users-w5wlf 0 9d	0/1	Completed
spark-hb-control-plane-7c6c49dd6d-r5w27 0 27d	2/2	Running
spark-hb-create-trust-store-7d9856999d-s6h4j 4 27d	1/1	Running
spark-hb-deployer-agent-7cbfdd85d-hh7bh 0 27d	1/1	Running
spark-hb-helm-repo-fd564c9dd-jf6gp 0 27d	1/1	Running

spark-hb-job-cleanup-cron-27348090-1npqh 0 20m	0/1	Completed
spark-hb-kernel-cleanup-cron-27348090-w4k5r 0 20m	0/1	Completed
spark-hb-nginx-6dc6f5cc47-kqr9m 0 27d	1/1	Running
spark-hb-preload-jkg-image-27334080-4s9w4 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-6t17q 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-79wwr 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-8h7ld 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-9dzwt 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-9ktpn 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-b67bt 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-bgzks 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-cqhff 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-h16pt 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-nq85s 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-qb65w 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-rqzwb 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-rs8r9 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-s5x8x 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-vfvkx 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-vgwf1 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27334080-x2t19 0 9d	0/2	Completed
spark-hb-preload-jkg-image-27348000-4kq6q 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-8dck4 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-99f4d 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-9k2d8 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-9zv7d 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-dngkm 0 110m	0/2	Completed

spark-hb-preload-jkg-image-27348000-f48fq 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-fd9xd 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-g28c7 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-g8g8g 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-jr1gp 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-pn894 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-ptcd9 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-qjvss 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-rh2vs 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-t4kkk 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-v9glg 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-wp5xh 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-xsls 0 110m	0/2	Completed
spark-hb-preload-jkg-image-27348000-xv2cf 0 110m	0/2	Completed
spark-hb-register-hb-dataplane-7d7ff9b7b6-gljbv6 0 27d	1/1	Running
spark-hb-terminating-pod-cleanup-cron-27348090-p4qvx 0 20m	0/1	Completed
spawner-api-5859f84cc9-6z8xz 0 9d	1/1	Running
usermgmt-5866c88858-75slb 0 27d	1/1	Running
usermgmt-5866c88858-kn2sb 0 27d	1/1	Running
watchdog-alert-monitoring-cronjob-27309120-zsm2j 0 27d	0/1	Terminating
watchdog-alert-monitoring-cronjob-27348110-qn2qn 0 21s	0/1	Completed
wdp-connect-connection-64cf749f4-795vr 0 9d	1/1	Running
wdp-connect-connector-85cddb4c6-rwgqq 0 9d	1/1	Running
wdp-connect-flight-84db5b6b6d-4124r 0 9d	1/1	Running
wdp-couchdb-0 0 9d	2/2	Running
wdp-couchdb-1 0 9d	2/2	Running
wdp-couchdb-2 0 9d	2/2	Running

wdp-dataview-cfdd8bb67-66p6r	1/1	Running
0 9d		
wkc-base-roles-init-95czl	0/1	Completed
0 9d		
wkc-extensions-translations-init-jr4n8	0/1	Completed
0 9d		
wkc-search-dc4b97cf4-kf6r2	1/1	Running
3 9d		
wml-deployment-agent-0	1/1	Running
0 27d		
wml-deployment-envoy-5fbc7d5879-9lzz8	1/1	Running
0 27d		
wml-deployment-manager-0	1/1	Running
0 27d		
wml-deployments-etcd-0	1/1	Running
5 27d		
wml-deployments-etcd-1	1/1	Running
3 27d		
wml-deployments-etcd-2	1/1	Running
3 27d		
wml-main-7dbd4b94cb-pz2dc	1/1	Running
0 9d		
wml-post-upgrade-and-rollback-f987h	0/1	Completed
0 9d		
wml-repositoryv4-6446968d8-tgh6j	1/1	Running
0 27d		
zen-audit-74f95bc9f4-m1xkt	1/1	Running
0 27d		
zen-core-66dfbbcc7c-k67m1	1/1	Running
0 27d		
zen-core-66dfbbcc7c-wfw1b	1/1	Running
0 27d		
zen-core-api-78dbf59d58-mpfxn	1/1	Running
8 27d		
zen-core-api-78dbf59d58-p8lt8	1/1	Running
10 27d		
zen-data-sorcerer-7997f96d44-vqbb5	1/1	Running
0 27d		
zen-database-core-fc579d6bf-v9j2j	1/1	Running
0 9d		
zen-databases-576d588ddd-jc44n	1/1	Running
0 9d		
zen-databases-576d588ddd-wwqqg	1/1	Running
0 9d		
zen-metastoredb-0	1/1	Running
2 27d		
zen-metastoredb-1	1/1	Running
3 27d		
zen-metastoredb-2	1/1	Running
3 27d		
zen-metastoredb-certs-hljgl	0/1	Completed
0 27d		
zen-metastoredb-init-tfkdz	0/1	Completed
0 27d		

zen-post-requisite-job-fhm4h	0/1	Completed
0 27d		
zen-pre-requisite-job-82rwv	0/1	Completed
0 27d		
zen-watchdog-76fbd894f6-tvpcp	1/1	Running
0 27d		
zen-watchdog-cronjob-27348110-hdsrz	0/1	Completed
0 21s		
zen-watchdog-post-requisite-job-fhqkt	0/1	Completed
0 27d		
zen-watcher-68f694cccc-pjlkn	1/1	Running
0 27d		

We used the Cloud Pak for Data integrated console for a common Db2 Warehouse operation, such as data insertion.

4.4 Best practices

Several factors might influence the infrastructure choices and deployments. For production environments, consider the following points:

- Performance

Performance metrics are workload dependent and often the KPIs are specific to the use case. In general, databases require a high-throughput I/O subsystem, which can serve and maintain high transaction throughput. The IBM Z I/O channel subsystem was specifically maintains high transaction throughput.

In our scenario, we used the colocation benefit. By colocating all the components (Cloud Pak for Data Db2 WH instance and the sample application) inside a single server and a single LPAR, we can reduce the delays that are associated with network latency. High-speed disks also contribute to the overall performance. In our scenario, we used high-speed DASD/ ECKD devices.

- High availability and disaster recovery

High availability (HA) can be achieved on various layers, from a service availability to a data center. The SLAs' Recovery Point Objective (RPO) and Recovery Time Objective (RTO) values dictate the availability level.

IBM Z is a highly reliable server with a strong Reliability, Availability, and Serviceability (RAS) philosophy and implementation on various levels of hardware and software. Red Hat OpenShift cluster also provides a certain level of HA by using the redundancy of the control and compute nodes with a seamless workloads migration between the compute nodes.

To exclude the single point of failure, many enterprises implement more than one Red Hat OpenShift cluster that are running as active-active on separate IBM Z servers. The underlying storage (Spectrum Scale cluster) was chosen because of its availability and stability (it has been a trusted and verified solution for some time on the market).

► Security

Security incident avoidance is a number one strategy for most enterprises (regardless of their size or portfolio of services) with a surge of cyberattacks and cybercrimes.

IBM Z server provides a unique approach; that is, a set of software, tools, and operations that are called *Pervasive Encryption*. This encryption allows companies to secure, encrypt, and manage their data, wherever it is stored. All of the workloads that are running on IBM Z (including Red Hat OpenShift containerized platform) inherit the following highly efficient security capabilities:

- Encryption of etcd data by way of CPACF with Red Hat OpenShift 4.8. For more information, see this [web page](#).
- The integration with IBM Z Crypto Express card and Red Hat OpenShift, delivered in 2021 specifically for IBM Z platform, allows the storage of highly sensitive data (such as private keys) on a most secure hardware security module (HSM). For more information, see this [web page](#).

► Operability

Streamlining of day 1 and day 2 operations contribute to overall effectiveness of DevOps processes. Immediately available Red Hat OpenShift and Cloud Pak for Data provides a set of tools that must be implemented in the practice. Those tools can be complemented by various DevOps tools that are available for IBM Z platform. These tools allow users to bring the applications faster to the hybrid cloud and accelerate the digital transformation and modernization of the assets.



Use Case: A Red Hat OpenShift environment that is colocated with z/OS transactional services and DB2 data

This chapter outlines the benefits of running a containerized application on Red Hat OpenShift Container Platform on IBM Z that is colocated with the System Of Records, which is hosted in another logical partition (LPAR) that is running an IBM Db2 database on the IBM z/OS operating system.

This use case describes the architecture and deployment of an Online Transaction Processing (OLTP) workload that simulates a lightweight banking application that performs eight different services.

This chapter includes the following topics:

- ▶ 5.1, “Colocation benefits on IBM Z” on page 90
- ▶ 5.2, “Workload and architecture” on page 92

5.1 Colocation benefits on IBM Z

One of the value propositions of Red Hat OpenShift Container Platform on IBM Z is the ability to colocate the application layer with the data layer in the same Central Processing Complex (CPC) in a different LPAR. Colocating the application layer with the data layer uses many of the inherent advantages of the IBM Z hardware and operating system to be used to your business advantage.

5.1.1 Performance and efficiency

Colocation within the same CEC enables operating systems to take advantage of cross-memory data transfer, which reduces overall request latency and CPU utilization and improves overall throughput by eliminating network traffic handling.

Shared Memory Communications over Remote Direct Memory Access

With Shared Memory Communications over Remote Direct Memory Access (SMC-R), IBM Z network capability takes a new leap, strengthening performance for sharing data and reducing data transmission network overhead. The new IBM Z RDMA over Converged Ethernet (RoCE) feature, 10 GbE RoCE Express, enables industry-standard RDMA capability on the Z platform.

As an RDMA-capable network interface card (RNIC), it can transfer data by using direct memory-to-memory communication, which reduces the CPU overhead of the networking software stack.

SMC-R provides application transparent use of this new RoCE feature to reduce the network overhead and latency of data transfers, which effectively offers the benefits of optimized network performance across processors. This breakthrough also lowers the CPU cost that is associated when moving large amounts of data.

Shared Memory Communications over Direct Memory Access

Internal Shared Memory (ISM) is a virtual PCI network adapter that enables direct access to shared virtual memory providing a highly optimized network interconnect for IBM Z intra-CPC (Central Processing Complex) communications.

HiperSockets

HiperSockets is a technology that provides high-speed TCP/IP connectivity within a central processor complex. It eliminates the need for any physical cabling or external networking connection between servers that are running in different LPARs. The communication occurs through the system memory of the processor; therefore, servers are connected to form an “internal LAN.”

The HiperSockets implementation is based on the OSA-Express Queued Direct I/O (QDIO) protocol. As a result, HiperSockets also is called internal QDIO (IQDIO). The microcode emulates the link control layer of an OSA-Express QDIO interface.

5.1.2 Operational benefits

Colocation within the same CEC across multiple Logical Partitions (LPARs) enables several benefits, such as the assertion of security identity, maintaining the same thread of execution, and dynamically tuning the workload with per-defined goals.

Security and dynamic workload placement

In today's digital era, data is the primary asset for most organizations. Access to data and changes to the data must be restricted to authorized persons, devices, or processes. Lack of stringent security measures means to risk an organization's core assets, their reputation, and customer confidence. They also face a direct financial burden from the breach, especially the risk of getting fined by government regulations when personal client data is lost.

The hypervisor PR/SM that enables the creation of logical partitions on IBM Z has the highest Common Criteria Evaluation Assurance Level of 5+ (EAL5+). This certification reaffirms that the applications that are running on one Logical Partition (LPAR) are isolated from applications that are running in another LPAR without compromising any data.

IBM Secure Service containers

IBM Secure Service containers (SSC) on IBM Z and IBM LinuxONE is container technology that helps to quickly deploy software appliances with a focus on application consumption and application security.

SSC includes the following key features:

- ▶ Provides tamper protection during installation and run time
- ▶ Ensures confidentiality of data and code that is in-flight and at-rest
- ▶ Management is provided by way of Remote APIs (RESTful) and web interfaces *only*
- ▶ Enables containers to be delivered by using distribution channels

IBM Secure Execution for Linux

IBM Secure Execution for Linux is a hardware-based security technology that is built into the IBM Z and LinuxONE latest generation systems. It provides scalable isolation for individual workloads to help protect them from external attacks and insider threats.

Secure Execution gives you the ability to use hardware-based security technology, such as Trusted Execution Environment (TEE) that enables hosted workloads to process unencrypted memory securely without exposing it to the host or any other workloads in the same environment.

Virtual Machine Resource Manager (VMRM) provides functions to dynamically tune the hypervisor z/VM system. Groups of virtual machines can be defined to be part of a workload. The workloads then can be managed by VMRM to goals that are also defined.

The system administrator can use VMRM to create a form of group scheduling for a set of virtual machines. Multiple workloads (groups of virtual machines) can be used, each managed to different goals. VMRM automatically adjusts performance parameters when contention exists between virtual machines for a resource. VMRM is not effective in environments where resources are unconstrained.

5.2 Workload and architecture

The workload that is used for this scenario in our IBM Redbooks lab environment is an Online Transaction Processing (OLTP) workload. This workload simulates a light-weight banking application that performs the following services:

- ▶ Login
This service performs customer login processes that validate customer identification numbers and updating status field to reflect the log-on processes.
- ▶ Accounts Summary
This service queries and provides results that contain account-specific information for each logged on customer.
- ▶ Deposit
This service is started when a credit request must be performed on a specific customer account that logged on to application after passing the necessary validations.
- ▶ Withdrawal
This service is started when a debit must be performed against a specific customer account.
- ▶ Customer Profile
The service is started if the customer wants to update any personal information that is stored in the database.
- ▶ Customer Transaction
This service provides the history of transaction services that were performed on various accounts for a specific customer.
- ▶ Log-off
This service invalidates the session after committing updates and updating the STATUS field in the database.
- ▶ Web-tier
This service provides the front-end logic for displaying various pages as the user browses through the application and starts the corresponding service that is based on the functions that are started.

The application follows a micro services-based architecture. The eight lightweight micro services that are listed here include seven back-end services and one front-end web tier service.

These application services are written in Java and are deployed as containers on an IBM WebSphere Liberty server. These containers are deployed as distinct Kubernetes pods on a Red Hat OpenShift Container Platform, as shown in Figure 5-1.

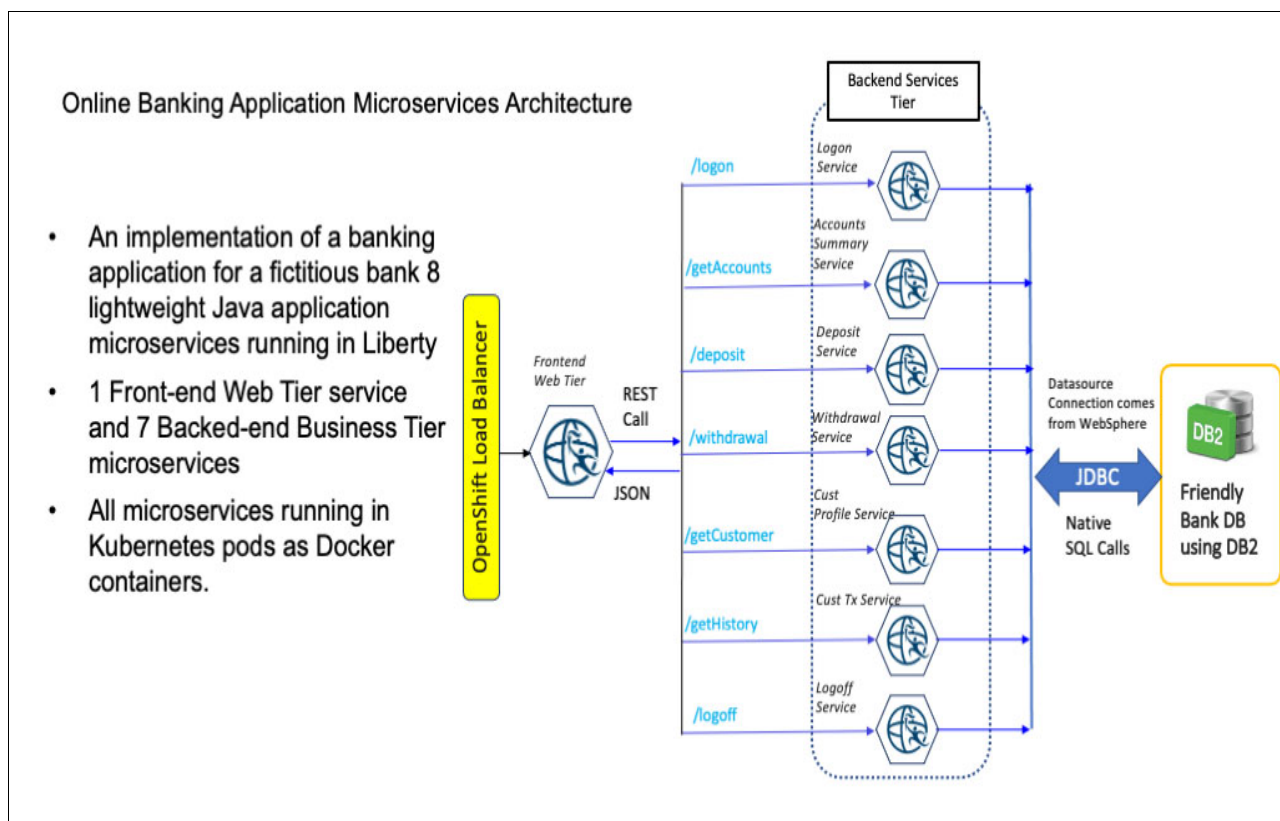


Figure 5-1 Banking application architecture

The Red Hat OpenShift Container Platform is based on version 4.7 and includes three Control Plane (master) nodes and four Compute (worker) nodes. The services are deployed across these four Compute nodes.

Resource allocation for Red Hat OpenShift Container Platform nodes that are used in this IBM Redbooks publication are listed in Table 5-1.

Table 5-1

z/VM Guest Name	Virtual CPUs	Memory (GB)	Disk Capacity (GB)
rdbko6m0	4	16	100
rdbk06m1	4	16	100
rdbko6m2	4	16	100
rdbko6w0	8	16	100
rdbk06w1	8	16	100
rdbk06w2	8	16	100
rdbkorw3	8	16	100

The System Of Records for the online banking application is an IBM Db2 database that is colocated in an LPAR that is running on the IBM z/OS operating system. This configuration is within the same IBM Z where an application is deployed on another LPAR with Red Hat OpenShift running on top of an IBM z/VM operating system (see Figure 5-2).

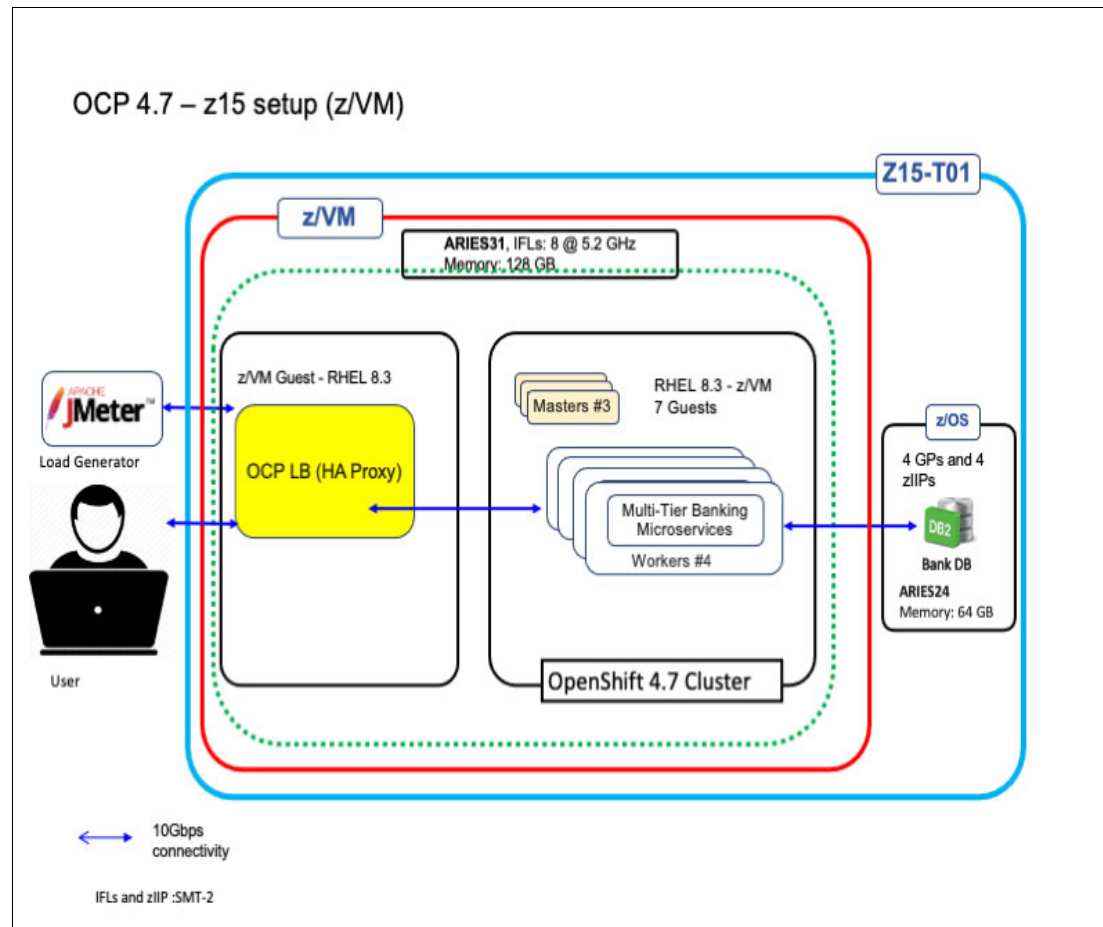


Figure 5-2 OCP-zVM-Deployment



Installing Red Hat OpenShift Container Storage on IBM Z

This appendix introduces Red Hat OpenShift Container Storage on IBM Z and includes the following topics:

- ▶ A.1, “Overview” on page 96
- ▶ A.2, “Architecture” on page 97
- ▶ A.3, “Planning and requirements” on page 99
- ▶ A.4, “Installation” on page 101
- ▶ A.5, “Red Hat OpenShift Container Platform registry integration with Red Hat OpenShift Container Storage” on page 126

A.1 Overview

Red Hat OpenShift Container Storage is software-defined storage that is optimized for container environments. It runs as an operator on Red Hat OpenShift Container Platform to provide highly integrated and simplified persistent storage management for containers.¹

Applications that are seeking resilient and scalable containerized deployments must be designed for a persistent storage option to work seamlessly in true stateless mode. This design helps to recover transient data that maintains the state and context of the associated container process during redeployment and scaling.

Figure A-1 shows an overview of the relationship between Red Hat OpenShift Container Platform and Red Hat OpenShift Container Storage. Whether Red Hat OpenShift Container Platform is on-premises or a cloud-based deployment, Red Hat OpenShift Container Storage provides an integrated solution that supports block-based devices, shared file system levels, and multicloud object storage.

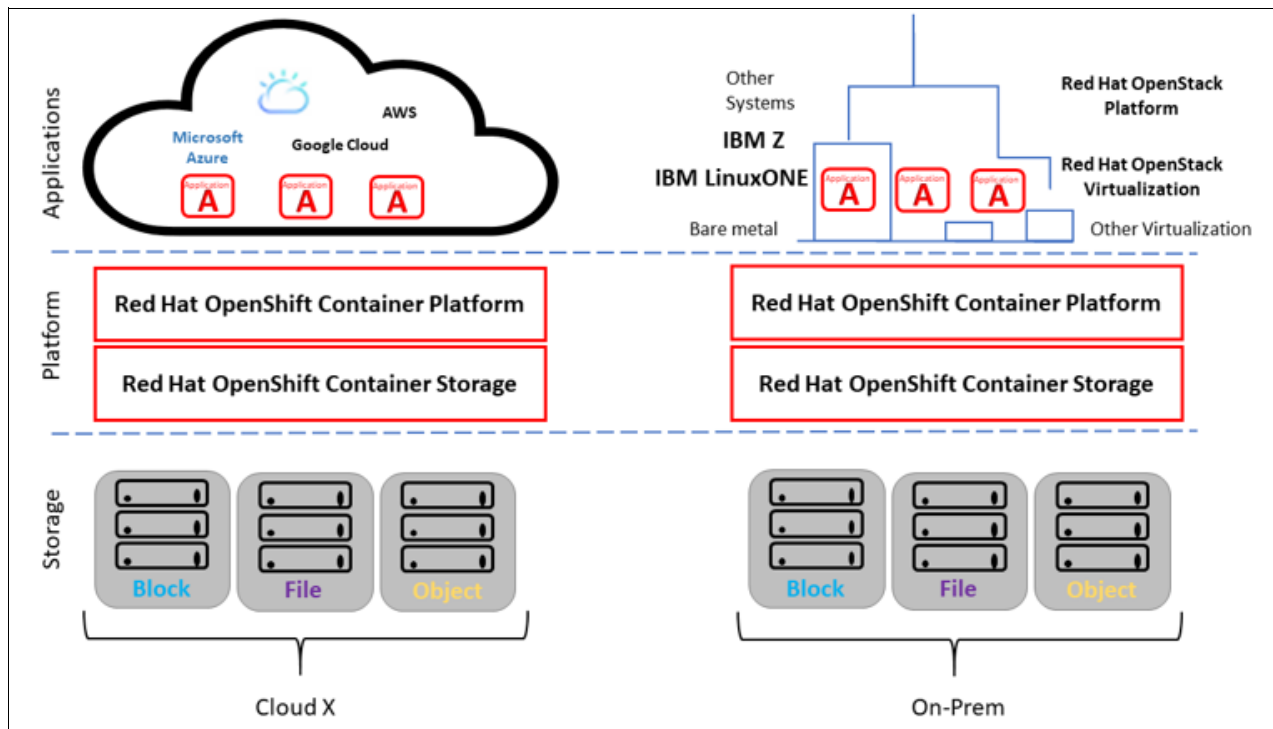


Figure A-1 Red Hat OpenShift Container Storage

¹ https://access.redhat.com/documentation/en-us/red_hat_openshift_container_storage/4.7/html/4.7_release_notes/introduction

A.2 Architecture

Red Hat OpenShift Container Storage (OCS) services are primarily made available to applications by way of storage classes that represent the following components:

- ▶ Block storage devices, primarily for database workloads. Prime examples include Red Hat OpenShift Container Platform logging and monitoring, and PostgreSQL.
- ▶ Shared and distributed file systems, which cater to software development, messaging, and data aggregation workloads. Examples include Jenkins build sources and artifacts, Wordpress uploaded content, Red Hat OpenShift Container Platform registry, and messaging by using JBoss AMQ.
- ▶ Multicloud object storage, which features a lightweight S3 API endpoint that can abstract the storage and retrieval of data from multiple cloud object stores.
- ▶ On-premises object storage, which features a robust S3 API endpoint that scales to tens of petabytes and billions of objects, primarily targeting data intensive applications, such as Spark, Presto, Red Hat AMQ Streams (Kafka), and even machine learning frameworks, such as TensorFlow and Pytorch.

In the subsequent sections, we discuss an implementation of Red Hat OpenShift Container Storage services and how to integrate a shared file system with the Red Hat OpenShift Container registry.

Red Hat OpenShift Container Storage consists of the following components:

- ▶ Ceph, a software-defined storage solution that is based on open source. It provides block storage, a shared and distributed file system, and on-premises object storage
- ▶ Ceph Container Storage Interface (CSI) to manage provisioning and lifecycle of persistent volumes and claims
- ▶ NooBaa, which provides a Multicloud Object Gateway
- ▶ Red Hat OpenShift Container Storage, Rook-Ceph, and NooBaa operators to initialize and manage Red Hat OpenShift Container Storage services

An overview of the Red Hat OpenShift Container Storage component level architecture and highlights where Rook fits into the architecture is shown in Figure A-2 on page 98.

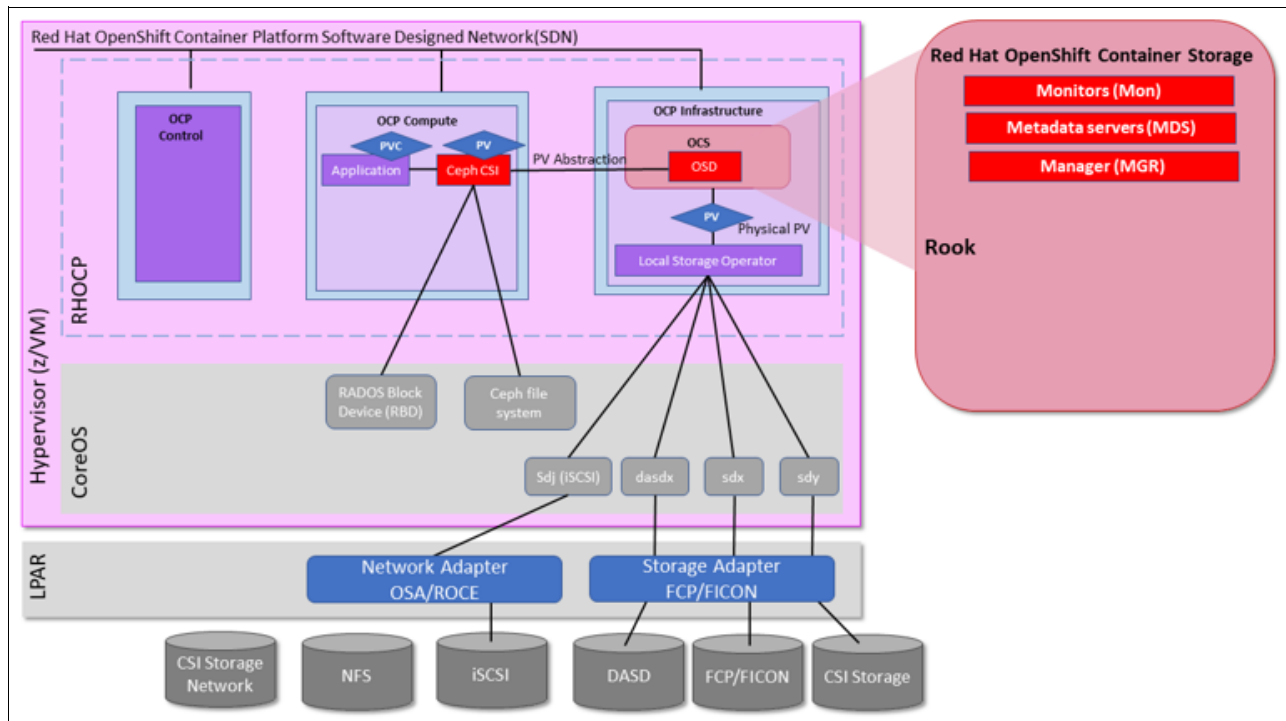


Figure A-2 Red Hat OpenShift Container Storage architecture

Rook is an open source file, block, and object storage for your cloud native environment and is based on battle tested Ceph storage. Rook offers storage for your Red Hat OpenShift applications through persistent volumes that can be dynamically provisioned with Kubernetes StorageClass.

As shown in Figure A-2, a Control Plane (master nodes), Compute nodes (worker nodes), and an Infrastructure node are all part of the Red Hat OpenShift Container platform. The network communication between all these components occurs through the Red Hat OpenShift Container Platform Software Defined Network.

Red Hat OpenShift Storage can be deployed on a Compute node along with application pods or on an Infrastructure node.

Applications make a Persistent Volume Claim (PVC) that are passed through to a Ceph CSI, which manages the lifecycle of PVC and Persistent Volumes (PVs).

Ceph CSI, along with the Red Hat OpenShift Container Storage Object Storage daemon (OSD), encompasses the required components, such as metadata servers, monitors, and various gateways for multiple storage access.

Red Hat OpenShift Container Storage access to various storage devices, such as iSCSI, DASD(FICON), and SCSI(FCP) is managed by the Local Storage Operator of Red Hat OpenShift Container Platform.

A.3 Planning and requirements

This section describes the planning and various prerequisites that must be met to install Red Hat OpenShift Container Storage on an IBM Z platform.

A.3.1 Planning

Red Hat OpenShift Container Storage can be deployed by using one of the following approaches:

► Internal

Deployment of Red Hat OpenShift Container Storage entirely within a Red Hat OpenShift Container Platform has all the benefits of operator-based deployment and management. An internally attached device approach in the graphical user interface can be used to deploy Red Hat OpenShift Container Storage in internal mode by using the local storage operator and local storage devices.

An internal deployment approach has the following two options:

– Simple

This option is suitable when storage requirements are not clear. In the Simple option, Red Hat OpenShift Container Storage is with the application and is *not* on separate nodes.

– Optimized

This option is ideal when storage requirements are clear and Red Hat OpenShift Container Storage can be deployed on dedicated Infrastructure Nodes.

► External

Red Hat OpenShift Container Storage exposes the Red Hat Ceph Storage services that are running outside of the Red Hat OpenShift Container Platform cluster as storage classes.

A.3.2 Resource requirements

Red Hat OpenShift Container Storage services consist of an initial set of base services and can be extended with more device sets. All of these Red Hat OpenShift Container Storage services pods are scheduled by Kubernetes on Red Hat OpenShift Container Platform nodes, according to the resource requirements that are listed in Table A-1.

Table A-1 Resource requirements

Deployment model	Base services	Other device set
Internal	<ul style="list-style-type: none">► 30 CPU (logical)► 72 GB memory► 3 storage devices	<ul style="list-style-type: none">► 6 CPU (logical)► 15 GB memory► 3 storage devices
External	<ul style="list-style-type: none">► 4 CPU (logical)► 16 GB memory	Not Applicable

In a three-node internal deployment approach, each node is assigned 10 CPU (logical) and 24 GB memory.

A.3.3 Node requirements

The cluster must consist of at least three Red Hat OpenShift Container Platform worker nodes with locally attached-storage devices on each of them. Consider the following points:

- ▶ Each of the three selected nodes must have at least one raw block device available to be used by Red Hat OpenShift Container Storage.
- ▶ The devices that you use must be empty; the disks must not include physical volumes, volume groups, or logical volumes that are remaining on the disk.

Note: As of this writing, support for the IBM Z platform is available for FCP attached storage devices only.

A.3.4 Storage requirements

Disk sizes of 4 TB or less can be used for local storage deployments (see Table A-2).

Table A-2 Storage requirements

Storage Device size (TB)	Storage Devices per node	Total capacity (TB)	Usable storage capacity
0.5	1	1.5	0.5
2	1	6	2
4	1	12	4

All disks must be of the same size and type. Always ensure that available storage capacity stays ahead of consumption. Recovery is difficult if available storage capacity is exhausted, and requires more intervention than adding capacity or deleting or migrating content.

A.4 Installation

This section describes the steps to install Red Hat OpenShift Container Storage on IBM Z.

A.4.1 Lab environment

Table A-3 lists our lab environment on which Red Hat OpenShift Container Storage was installed. The hypervisor on which the Red Hat OpenShift Container Platform cluster was running for this specific scenario was a Red Hat Kernel-based Virtual Machine (KVM).

Table A-3 Lab environment

Node Type	Node	Virtual CPU	Memory (GB)	OCP Disk (GB) (DASD/ECKD)	OCS Raw capacity (TB) (FCP/SCSI)
Master	rdbko7m0	4	16	300	Not Applicable
Master	rdbko7m1	4	16	300	Not Applicable
Master	rdbko7m2	4	16	300	Not Applicable
Worker	rdbko7w1	10	64	300	Not Applicable
Worker	rdbko7w2	10	64	300	Not Applicable
Worker	rdbko7w3	10	64	300	Not Applicable

Figure A-3 shows our lab deployment topology.

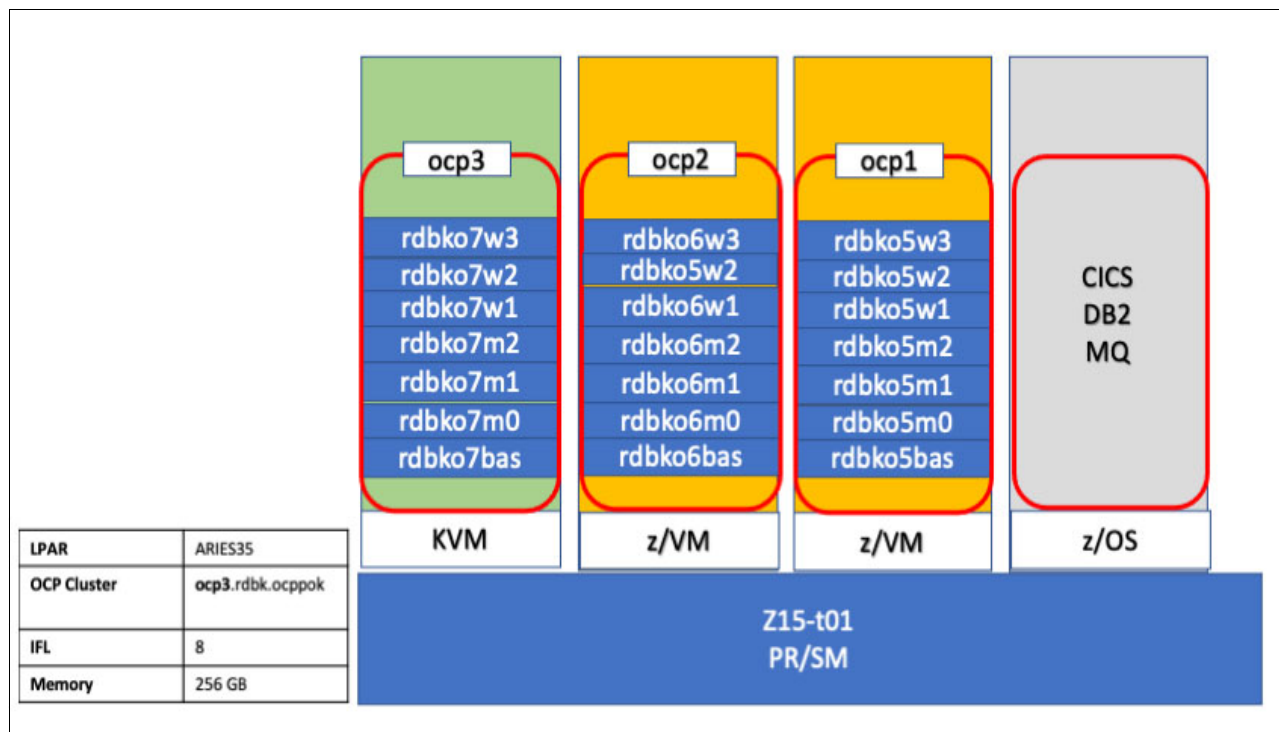


Figure A-3 Lab deployment topology

A.4.2 Assigning disks to KVM Guest

The Red Hat OpenShift Container Storage internal deployment approach needs a minimum of three nodes. In our lab environment, we assigned the three worker nodes for this purpose where Red Hat OpenShift Container storage pods are with the application pods.

We completed the following steps to use FCP attached storage devices as raw block devices:

1. We used **cio_ignore** allowlist for the required FCP device adapters in our KVM host to access the assigned LUNs, as shown in Example A-1

Example A-1 Checking whether devices are allowlisted

```
[root@rdbkvmr ~]# cio_ignore -l
Ignored devices:
=====
0.0.0000-0.0.0f9f
0.0.0fa3-0.0.1e1f
0.0.1e23-0.0.1e2b
0.0.1e2f-0.0.1e3f
0.0.1e43-0.0.1e4b
0.0.1e4f-0.0.90ca
0.0.90cc-0.0.90d7
0.0.90dc-0.0.91ca
0.0.91cc-0.0.91d7
0.0.91db-0.0.b740
0.0.b742-0.0.c740
0.0.c742-0.0.ffff
0.1.0000-0.1.ffff
0.2.0000-0.2.ffff
0.3.0000-0.3.ffff
[root@rdbkvmr ~]
```

2. In our lab environment, the FCP adapters that we used were b741 and c741. Because these devices were white-listed (see Example A-1), the devices must be activated to be used in host environment, as shown in Example A-2.

Example A-2 Enabling the devices

```
[root@rdbkvmr ~]# chccwdev -e b741
Setting device 0.0.b741 online
Done
[root@rdbkvmr ~]# chccwdev -e c741
Setting device 0.0.c741 online
Done
```

3. We verified that the devices are visible after the device adapters are online by using the command that is shown in Example A-3.

Example A-3 Checking whether devices are visible

```
root@rdbkvmr ~]# ls SCSI
[0:0:0:1073823744] disk IBM 2107900 6.30 /dev/sda
[0:0:0:1073823745] disk IBM 2107900 6.30 /dev/sdc
[0:0:0:1073889280] disk IBM 2107900 6.30 /dev/sdb
[0:0:1:1073823744] disk IBM 2107900 6.30 /dev/sdd
[0:0:1:1073823745] disk IBM 2107900 6.30 /dev/sdf
[0:0:1:1073889280] disk IBM 2107900 6.30 /dev/sde
[1:0:0:1073823744] disk IBM 2107900 6.30 /dev/sdg
[1:0:0:1073823745] disk IBM 2107900 6.30 /dev/sdi
[1:0:0:1073889280] disk IBM 2107900 6.30 /dev/sdh
[1:0:1:1073823744] disk IBM 2107900 6.30 /dev/sdj
[1:0:1:1073823745] disk IBM 2107900 6.30 /dev/sdl
[1:0:1:1073889280] disk IBM 2107900 6.30 /dev/sdk
```

4. The required devices must be partitioned as raw disks to be made available to Red Hat OpenShift Container Storage. We partitioned the devices by using the **fdisk** command, as shown in Example A-4.

Example A-4 Partitioning the disks

```
[root@rdbkvmr ~]# fdisk /dev/sda
```

Welcome to fdisk (util-linux 2.32.1).

Changes will remain in memory only until you decide to write them.
Be careful before using the write command.

Command (m for help): m

Help:

DOS (MBR)

- a toggle a bootable flag
- b edit nested BSD disklabel
- c toggle the dos compatibility flag

Generic

- d delete a partition
- F list free unpartitioned space
- l list known partition types
- n add a new partition
- p print the partition table
- t change a partition type
- v verify the partition table
- i print information about a partition

Misc

- m print this menu
- u change display/entry units
- x extra functionality (experts only)

Script

```
I load disk layout from sfdisk script file
O dump disk layout to sfdisk script file
```

Save & Exit

```
w write table to disk and exit
q quit without saving changes
```

Create a new label

```
g create a new empty GPT partition table
G create a new empty SGI (IRIX) partition table
o create a new empty DOS partition table
s create a new empty Sun partition table
```

Command (m for help):

To add a new partition use command “n”. To write the changes permanently to disk issue “w” at command prompt.

5. We printed the partition table (as shown in Example A-5) to verify that the partition was successfully created.

Example A-5 Printing the partition table

```
Command (m for help): p
Disk /dev/sda: 500 GiB, 536870912000 bytes, 1048576000 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x7b651768
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sda1		2048	1048575999	1048573952	500G	83	Linux

Command (m for help):

6. We repeated steps 4 and 5 for the required number of devices that must be attached to Red Hat OpenShift Container Storage nodes.
7. We attached the devices to corresponding KVM guests and restarted the nodes, as shown in Example A-6.

Example A-6 Editing the KVM guests to add corresponding storage devices

```
[root@rdbkvmr ~]# virsh edit rdbko7w1
```

Add the following section to guest xml definition file

```
</disk>
<disk type='block' device='disk'>
  <driver name='qemu' type='raw' cache='none' io='native'/>
  <source dev='/dev/sda'/>
  <target dev='vdb' bus='virtio'/>
  <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0004'/>
</disk>
```

8. We saved the definition file and restarted the guests by using the commands that are shown in Example A-7.

Example A-7 Shutting down and restarting guests

```
[root@rdbkvmr ~]# virsh shutdown rdbko7w1
[root@rdbkvmr ~]# virsh start rdbko7w1
```

9. We repeated steps 7 and 8 for all corresponding Red Hat OpenShift Container Storage nodes.
10. We logged on to the Red Hat OpenShift Container Storage nodes and verified that the devices are visible as block devices, as shown in Example A-8.

Example A-8 Verifying whether devices are present inside storage nodes

```
[root@rdbkbas7 .ssh]# ssh -i ~/.ssh/id_ocp3 core@9.76.61.43
[core@rdbko7w1 ~]$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
loop0       7:0    0   500G  0 loop
rbd0        251:0    0    50G  0 disk
/var/lib/kubelet/pods/ad26fad1-8cd0-4d0b-94e2-34bfc34473d2/volumes/kubernetes.i
o~csi/pvc-ab336d18-57ff-4f79-98df-d1b6ab261479/
vda         252:0    0   500G  0 disk
~-vda1 252:1    0   500G  0 part
vdb         252:16   0   100G  0 disk
|-vdb3 252:19   0    384M  0 part /boot
~-vdb4 252:20   0   99.6G  0 part /sysroot
[core@rdbko7w1 ~]$
```

A.4.3 Red Hat OpenShift Container Storage installation by using Operator Hub

Complete the following steps to install Red Hat OpenShift Container Storage on IBM Z by using the Red Hat OpenShift Container Platform storage operator:

1. Log on to the Red Hat OpenShift web console by using the kubeadmin user ID or htpasswd credentials. The Red Hat OpenShift web console is shown in Figure A-4.

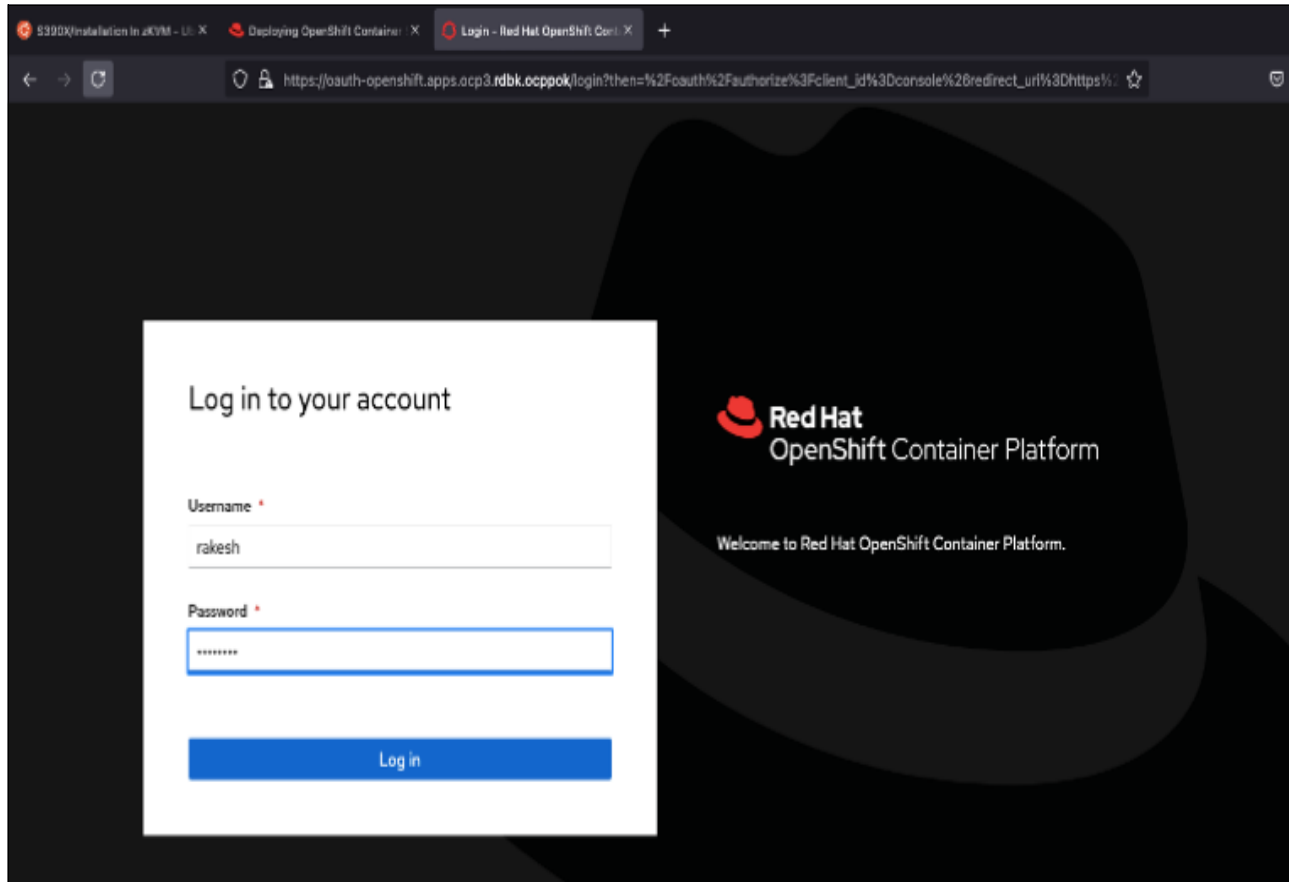


Figure A-4 Red Hat OpenShift web console

2. After successfully logging in, select the **Operators** option (see Figure A-5).

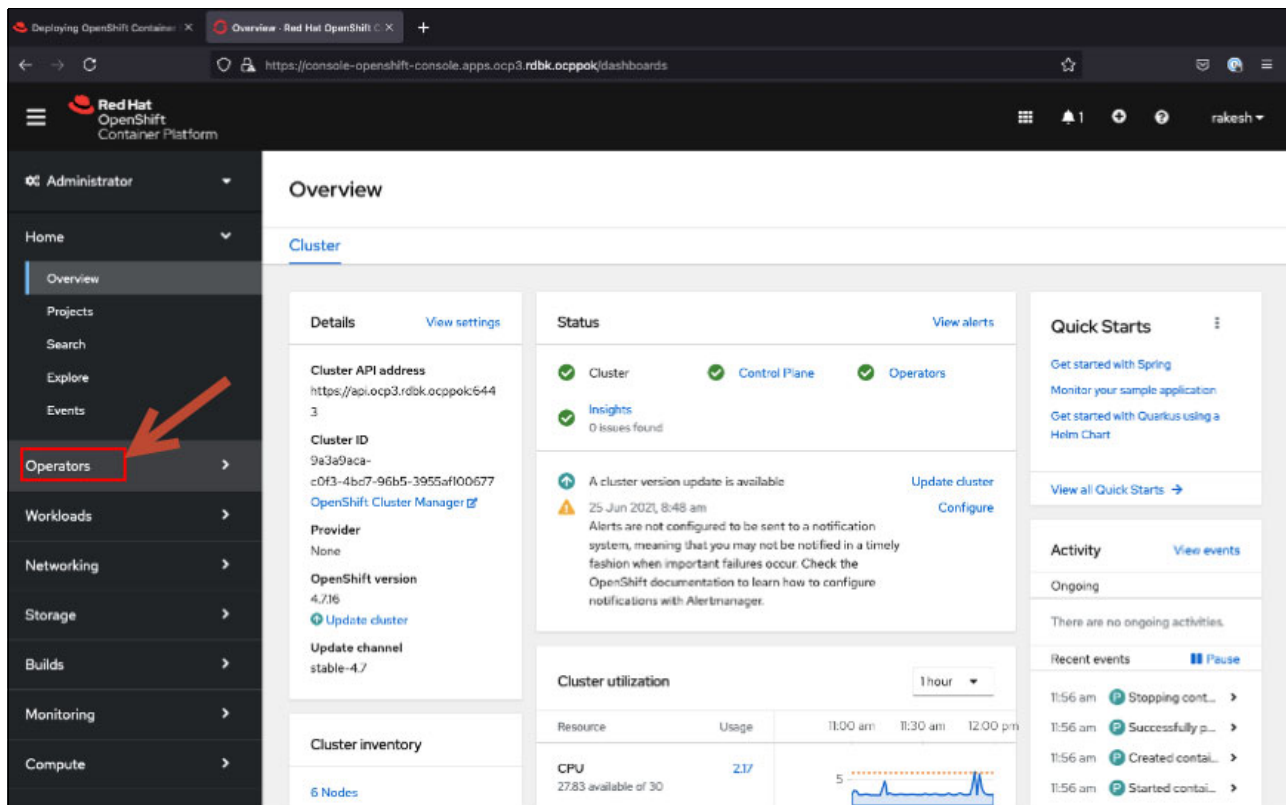


Figure A-5 Selecting the Operators option

3. Select **OperatorHub** (see Figure A-6).

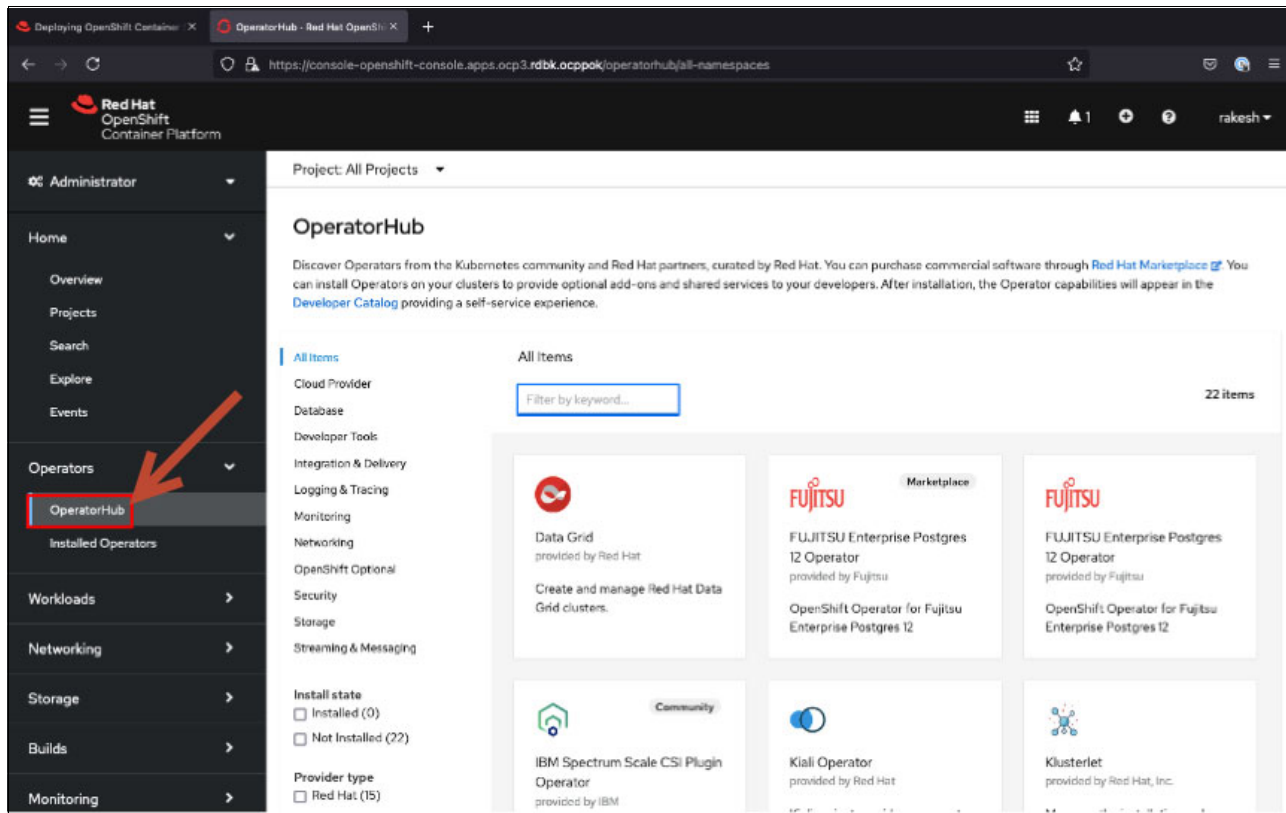


Figure A-6 OperatorHub

4. Apply a filter to narrow your search to specific operator names. As shown in Figure A-7, we used the conta filter to narrow our search for the Red Hat OpenShift Container Storage operator.

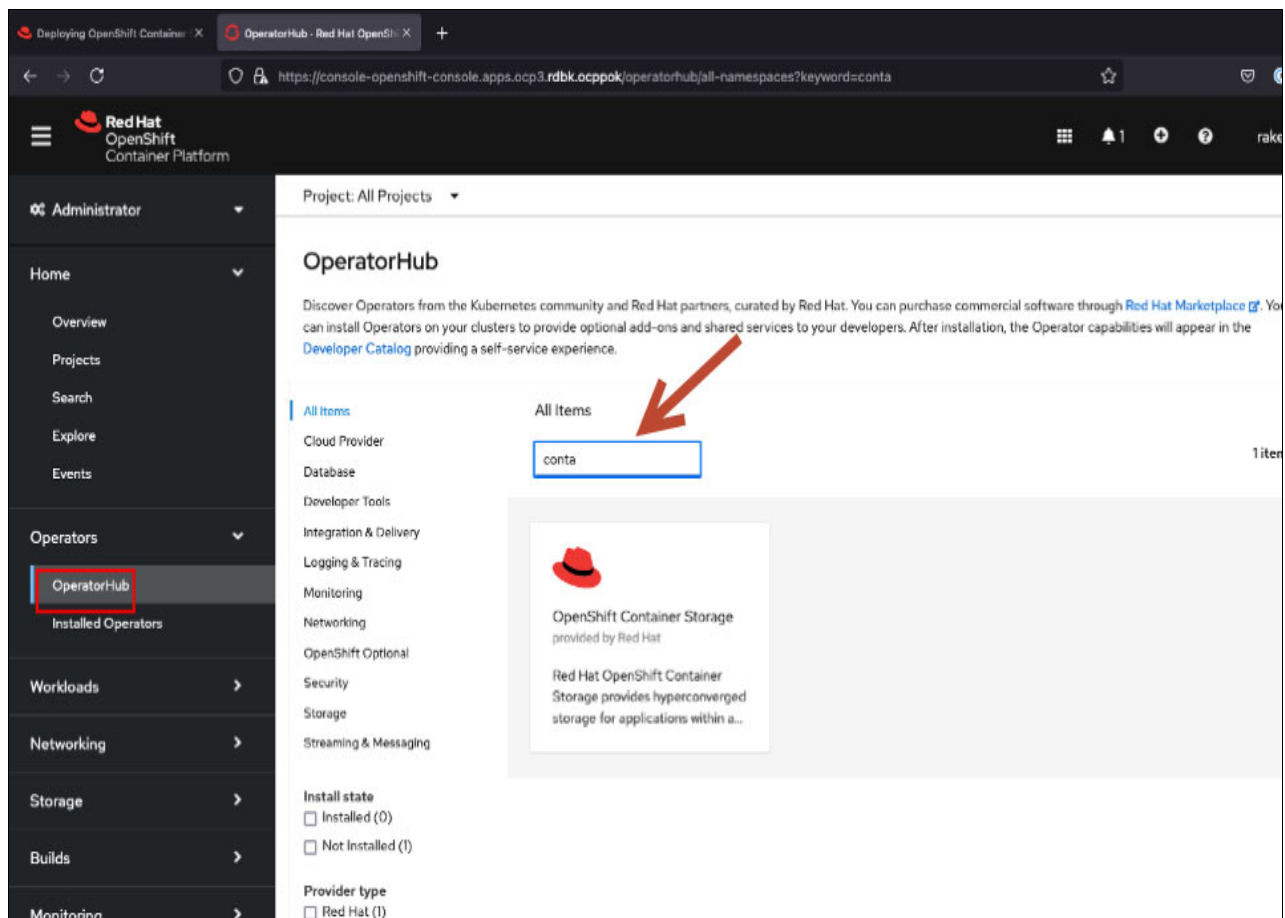


Figure A-7 Applying a filter

5. Select the Red Hat OpenShift Container Storage operator that is listed and proceed with operator installation by clicking **Install**, as shown in Figure A-8.

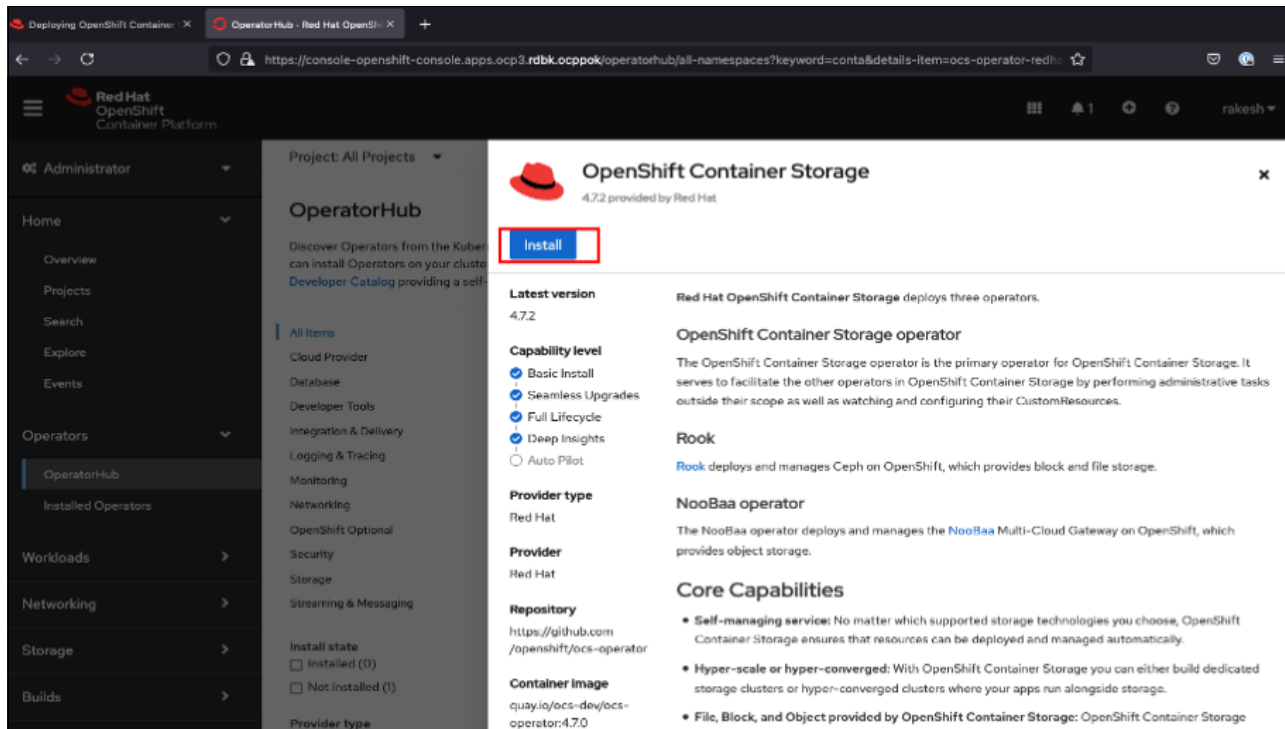


Figure A-8 OCS operator installation

6. Retain all the default values and click **Install**, as shown in Figure A-9.

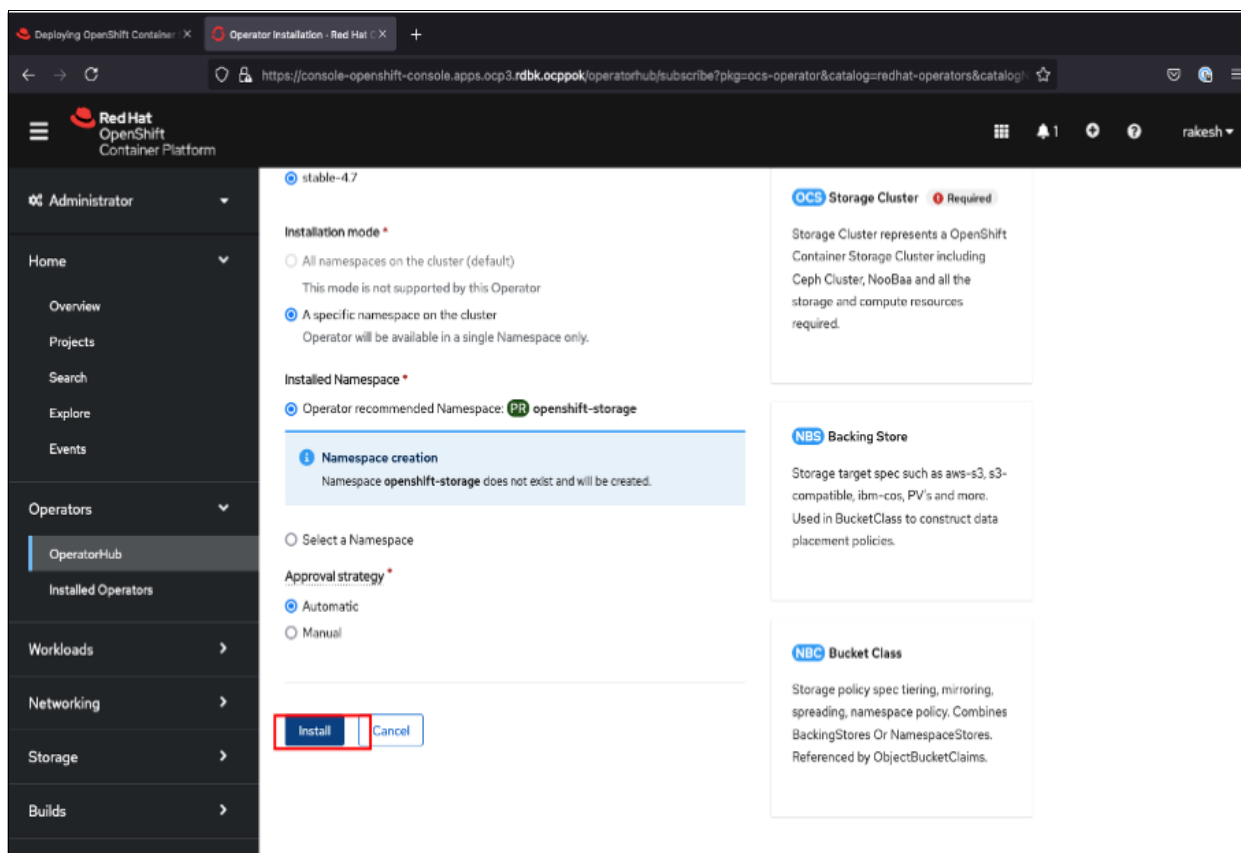


Figure A-9 OCS Operator installation options

After the operator is successfully installed, a green check mark is displayed, as shown in Figure A-10.

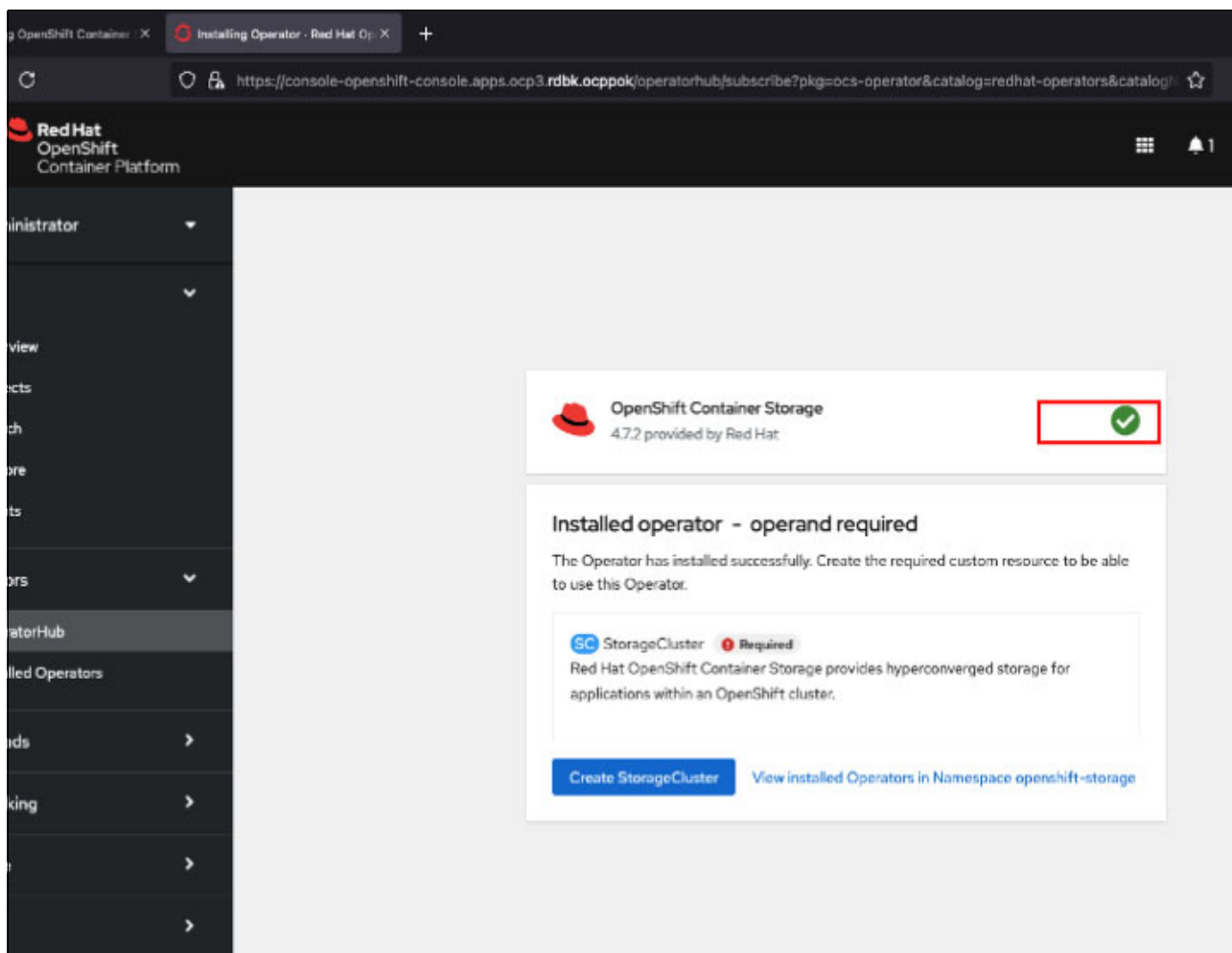


Figure A-10 OCS operator successful installation confirmation

7. After the Red Hat OpenShift Container Storage operator is installed, a storage cluster must be created. Click the **Installed Operators** option and select **Red Hat OpenShift Container Storage Operator**, which was installed in the previous step. Ensure that the selected project is **openshift-storage**).
8. Select the **Create Instance** link of the Storage Cluster option.
9. Select **Internal-Attached devices** for select-mode.

10. Select the required storage capacity (see Figure A-11) to be provisioned to the nodes as raw capacity and the nodes that are to be used for Red Hat OpenShift Container Storage.

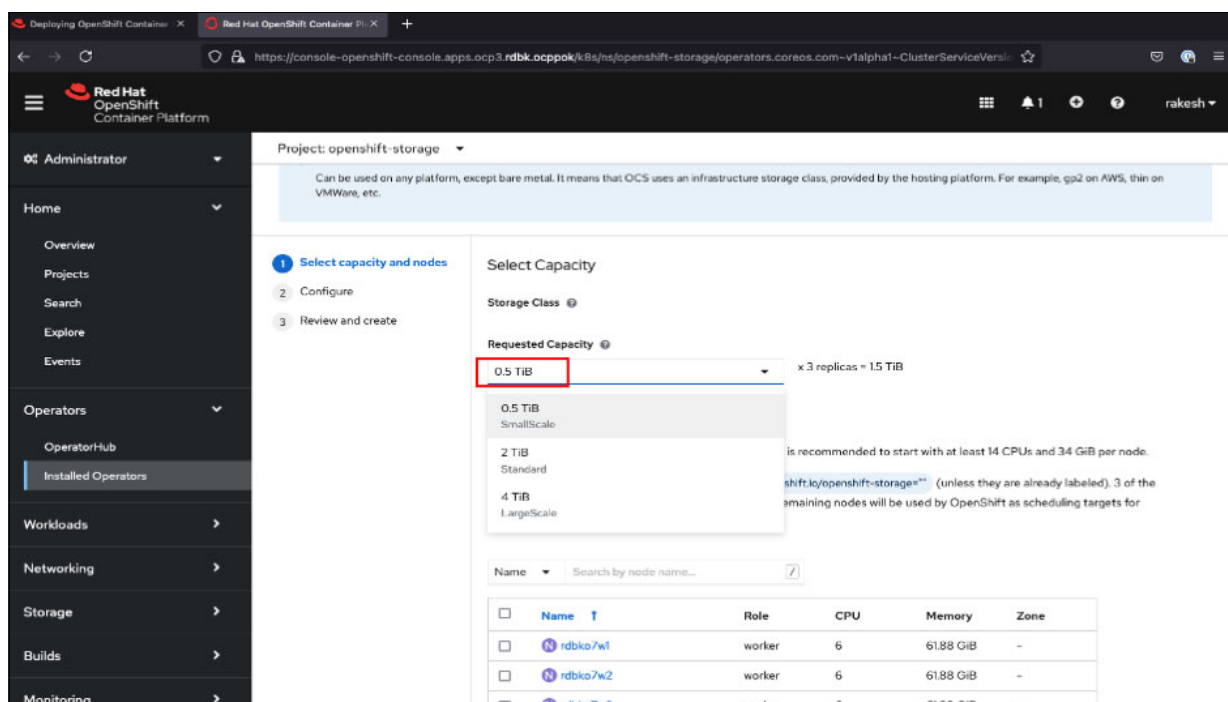


Figure A-11 Selecting storage capacity and node

11.If the minimum required vCPU allocations for Red Hat OpenShift Container Storage are not met, an error is displayed (see Figure A-12).

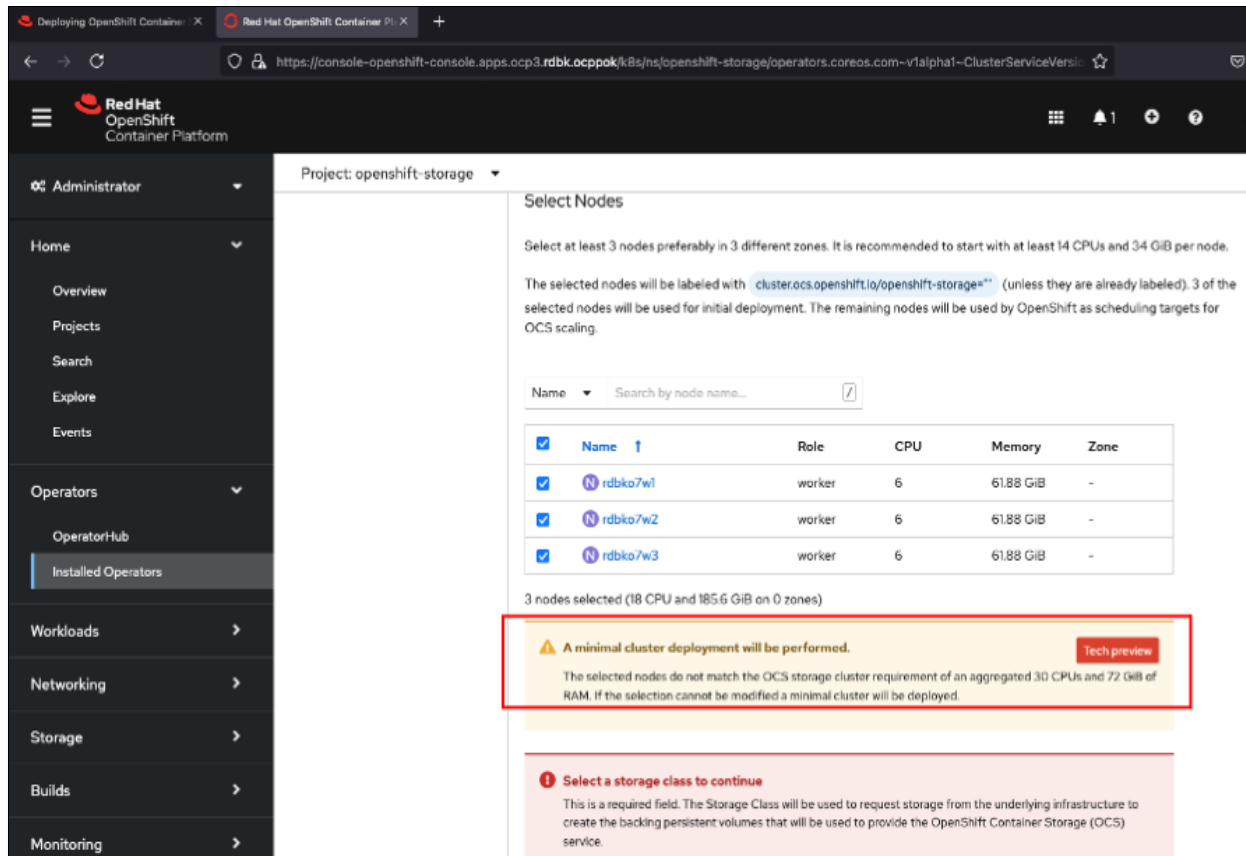


Figure A-12 CPU allocation

In our lab environment, six CPUs were allocated for the worker nodes, which were selected for Red Hat OpenShift Container Storage as opposed to minimum requirement of 30 aggregate CPUs.

The worker nodes were reconfigured to have 10 CPUs; therefore, the aggregate of 30 CPUs was achieved.

12. Install Local Storage operator *before* proceeding with the Create Storage Cluster wizard task. If the operator is not installed, a window is displayed that features an option to install it for you, as shown in Figure A-13.

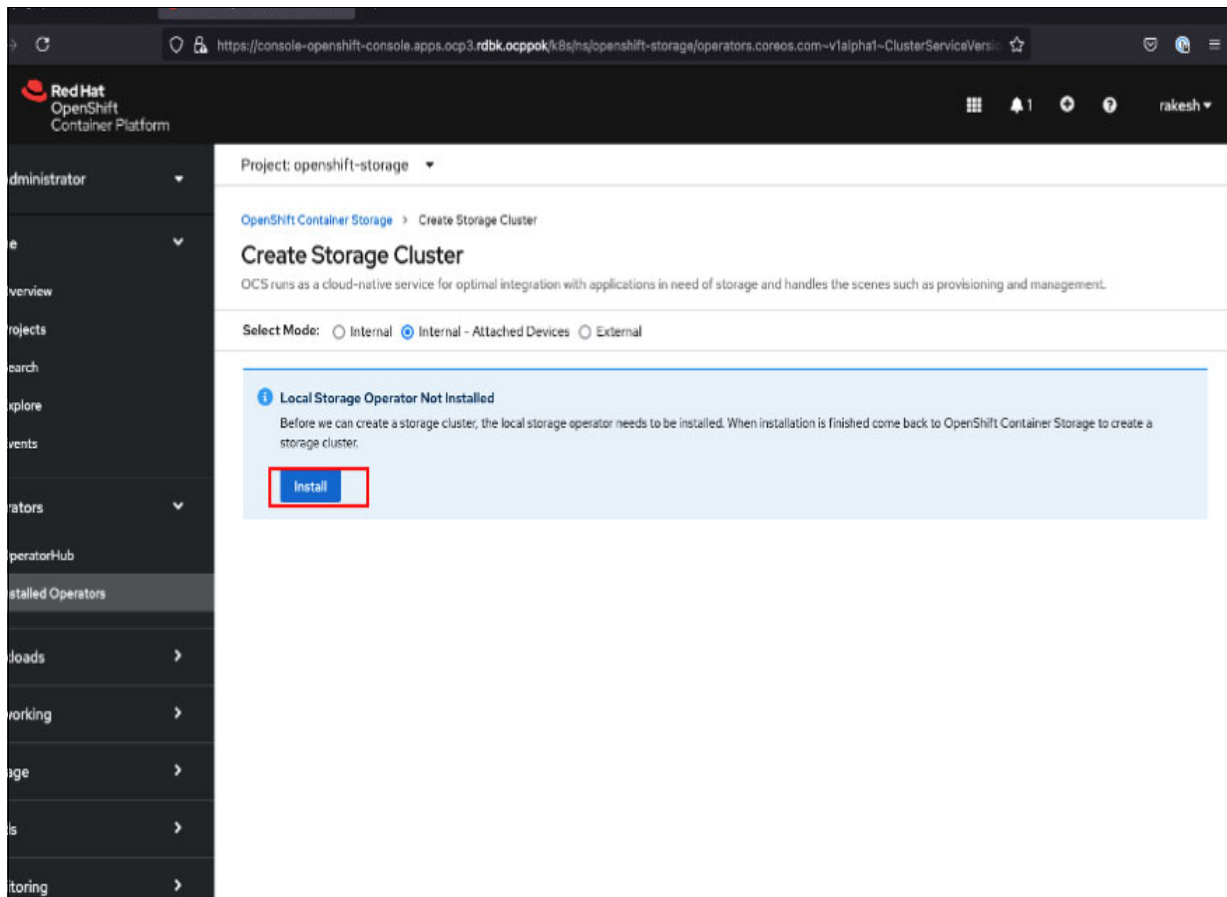


Figure A-13 LocalStorage Operator

13. Select the **Install Option** for Local Storage Operator in the Create Storage Cluster wizard.

14. Use all default values in the Local Storage Operator installation window and continue with the installation (see Figure A-14 and Figure A-15 on page 117).

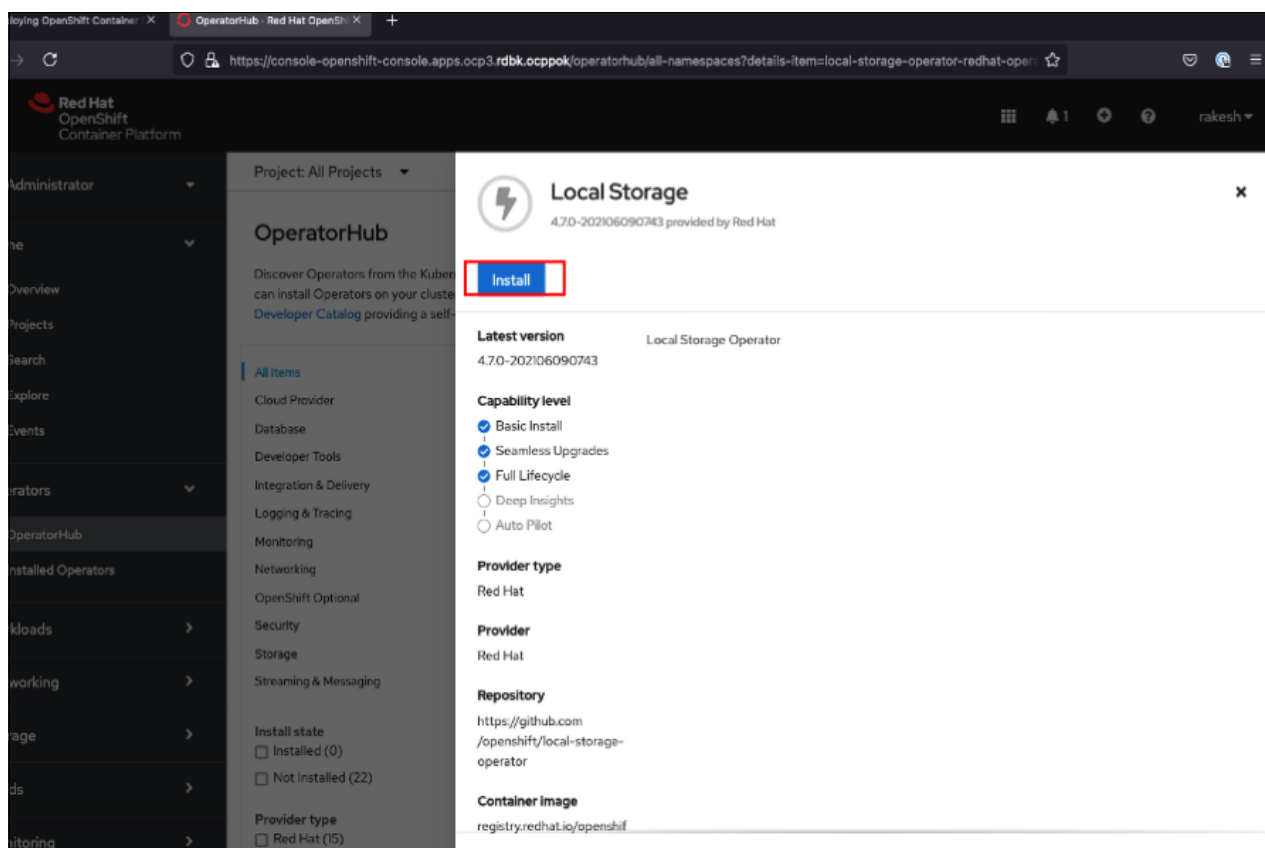


Figure A-14 LocalStorage Operator installation options

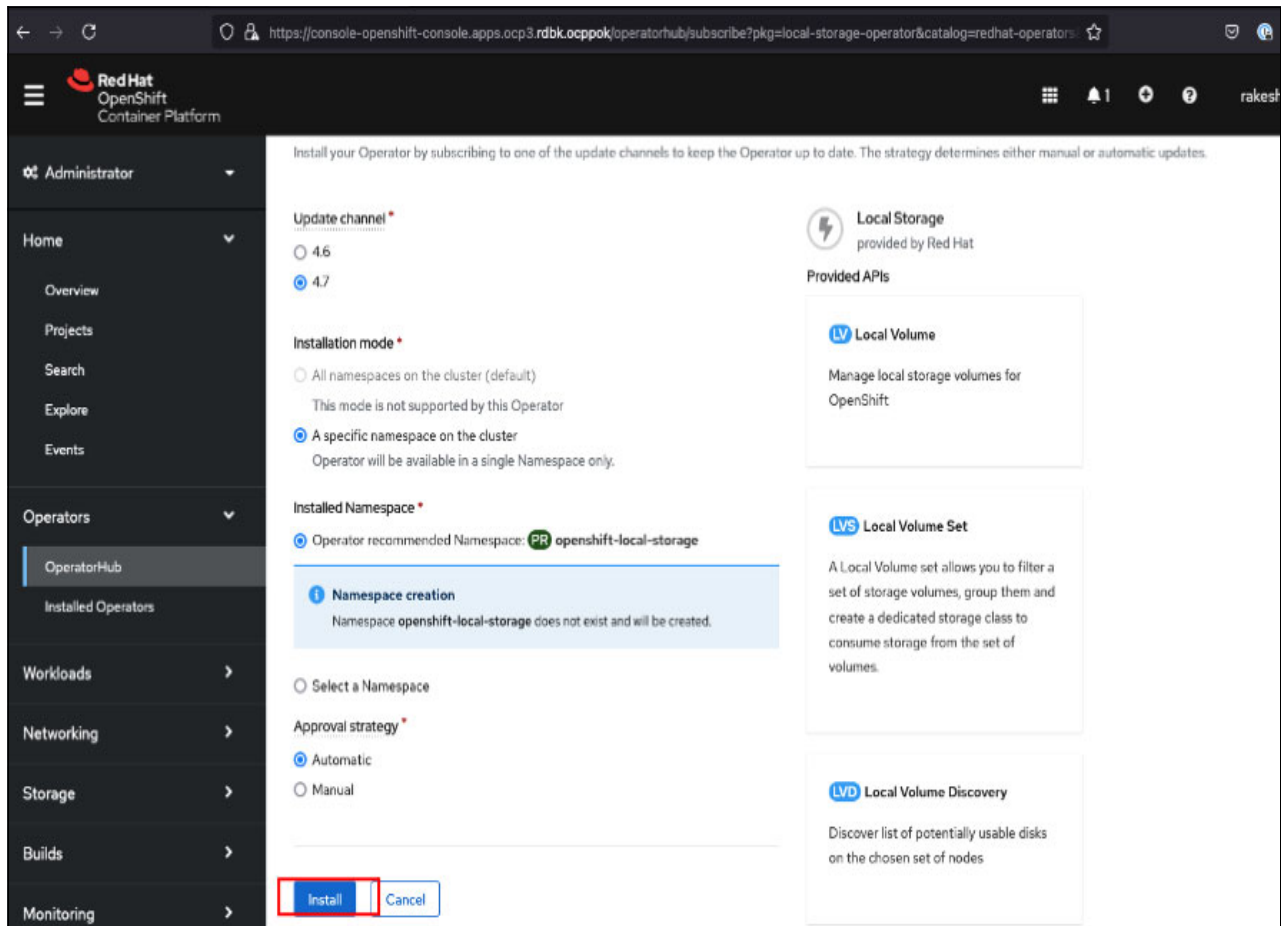


Figure A-15 Continuing with installation

15. After the installation is completed successfully, a green checkmark is displayed, as shown in Figure A-16.

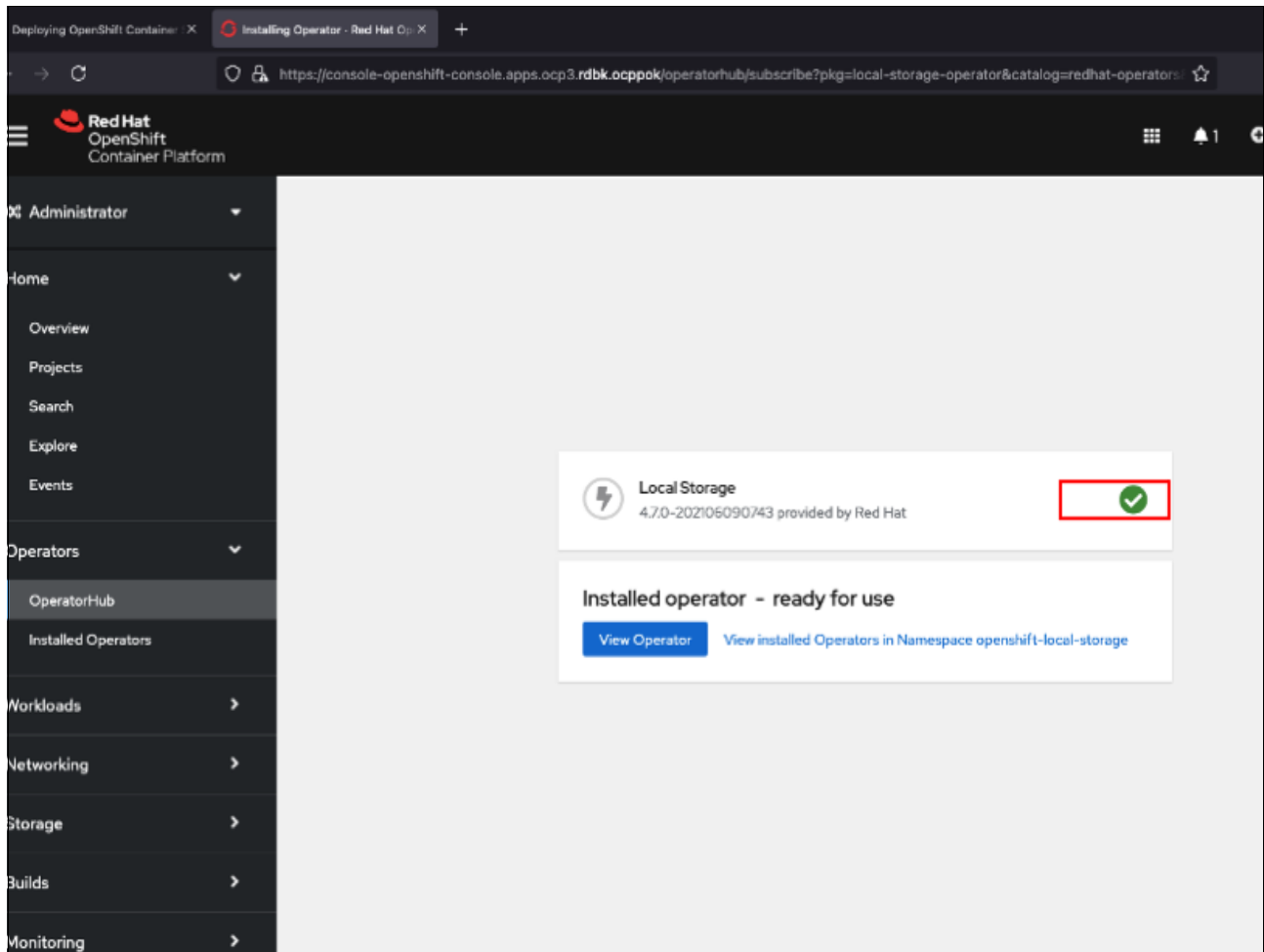


Figure A-16 LocalStorage operator installation completed successfully

16. To continue with the Create Storage Cluster wizard, click **Installed Operators** → **OpenShift Container Storage** and then, select **Create StorageCluster**, as shown in Figure A-17 and Figure A-18.

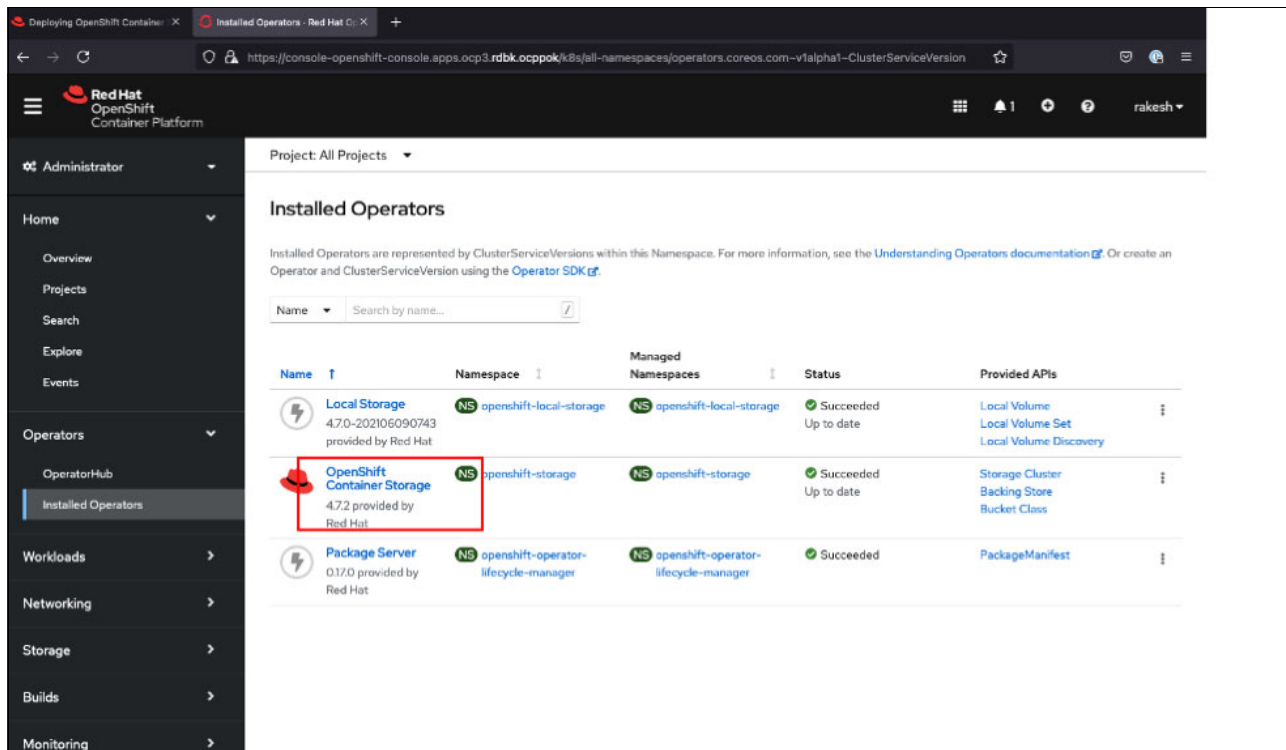


Figure A-17 Installed Operator-OCS

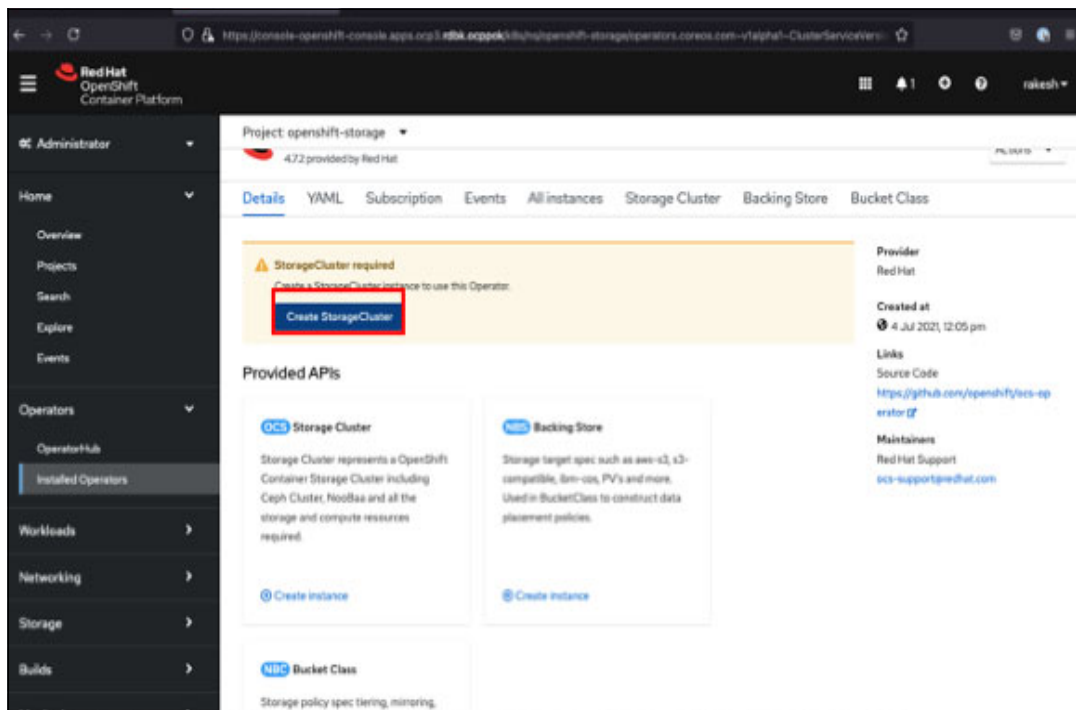


Figure A-18 Creating StorageCluster

17. In the wizard, select the **openshift-storage** project and the **Internal-attached devices** Select Mode (see Figure A-19). Click **Next**.

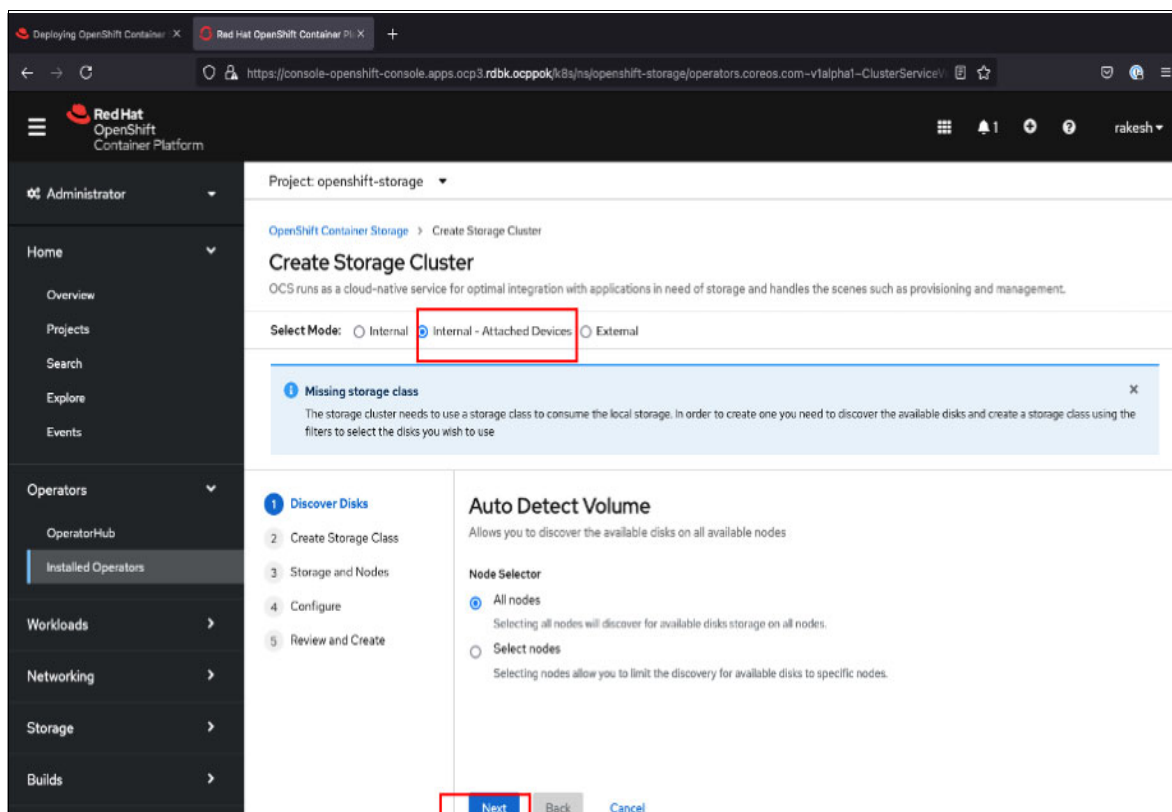


Figure A-19 StorageCluster Auto Detect Volume

For the Discover Disks option in our lab environment, we had three worker nodes. All three nodes were selected to act as Red Hat OpenShift Container Storage nodes. Therefore, the **All nodes** option was selected for automatically detecting the disks that were available to the nodes.

18. For Create Storage Class option, enter a name for the local volume set to create dedicated storage class and then, click **Next** (see Figure A-20).

Deploying OpenShift Container Platform | Red Hat OpenShift Container Platform

Project: openshift-storage

- Discover Disks
- Create Storage Class
- Storage and Nodes
- Configure
- Review and Create

Local Volume Set

A Local Volume Set allows you to filter a set of storage volumes group them and create a dedicated storage class to consume storage for them.

Volume Set Name *

ocp3vs

Storage Class Name

ocp3vs

3 Nodes 3 Disks

146 TiB Out of 146 TiB

Filter Disks

Node Selector

☒ All nodes (3 nodes)
Selecting all nodes will use the available disks that match the selected filters on all nodes selected on previous step.

☐ Select nodes
Selecting all nodes will use the available disks that match the selected filters only on selected nodes.

Disk Type

All

Advanced

Next Back Cancel

Figure A-20 Local volume set name

19. In the Confirmation window, select **Yes** (see Figure A-21).

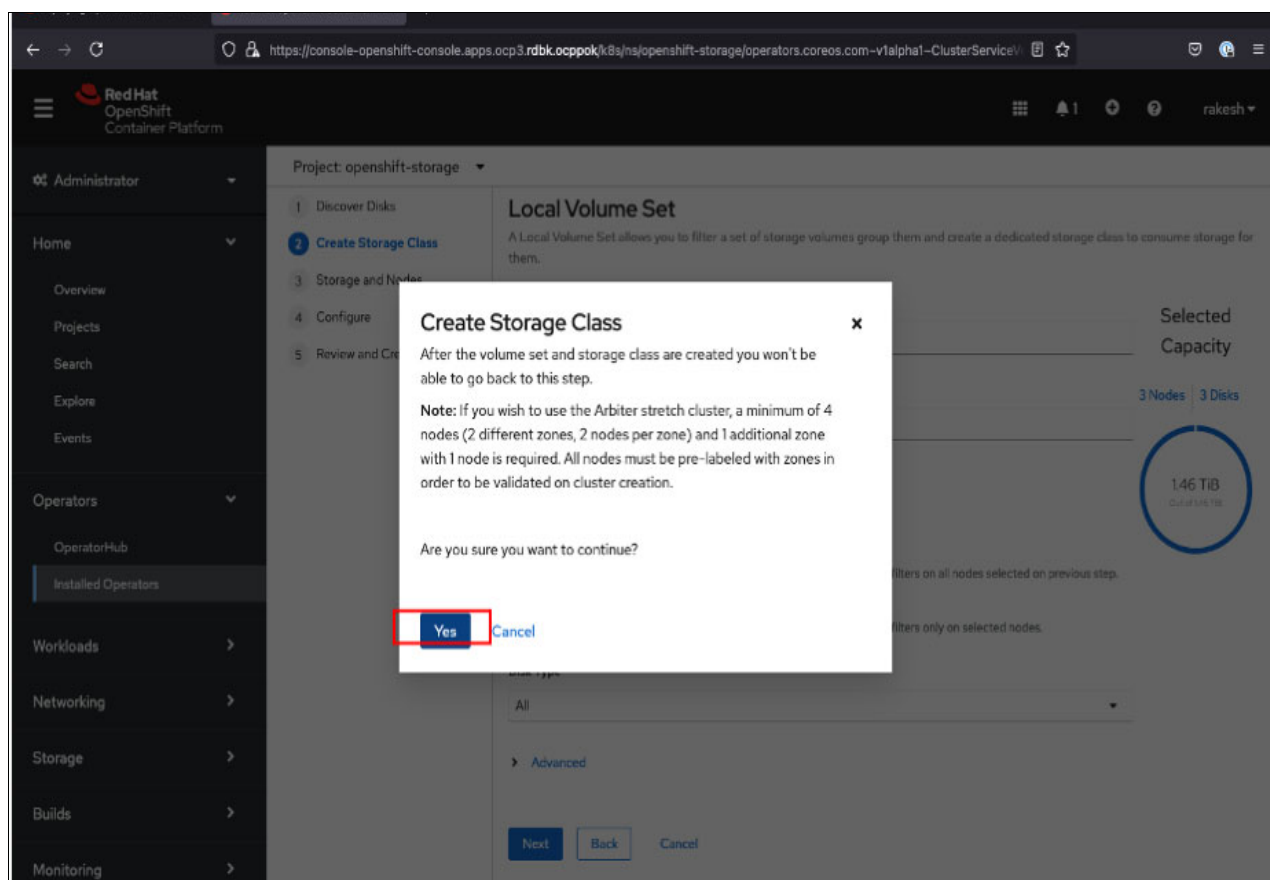


Figure A-21 StorageClass creation confirmation

In our lab environment, we did not enable Encryption for raw disks. Then, we proceeded with StorageClass creation without encryption, as shown in Figure A-22.

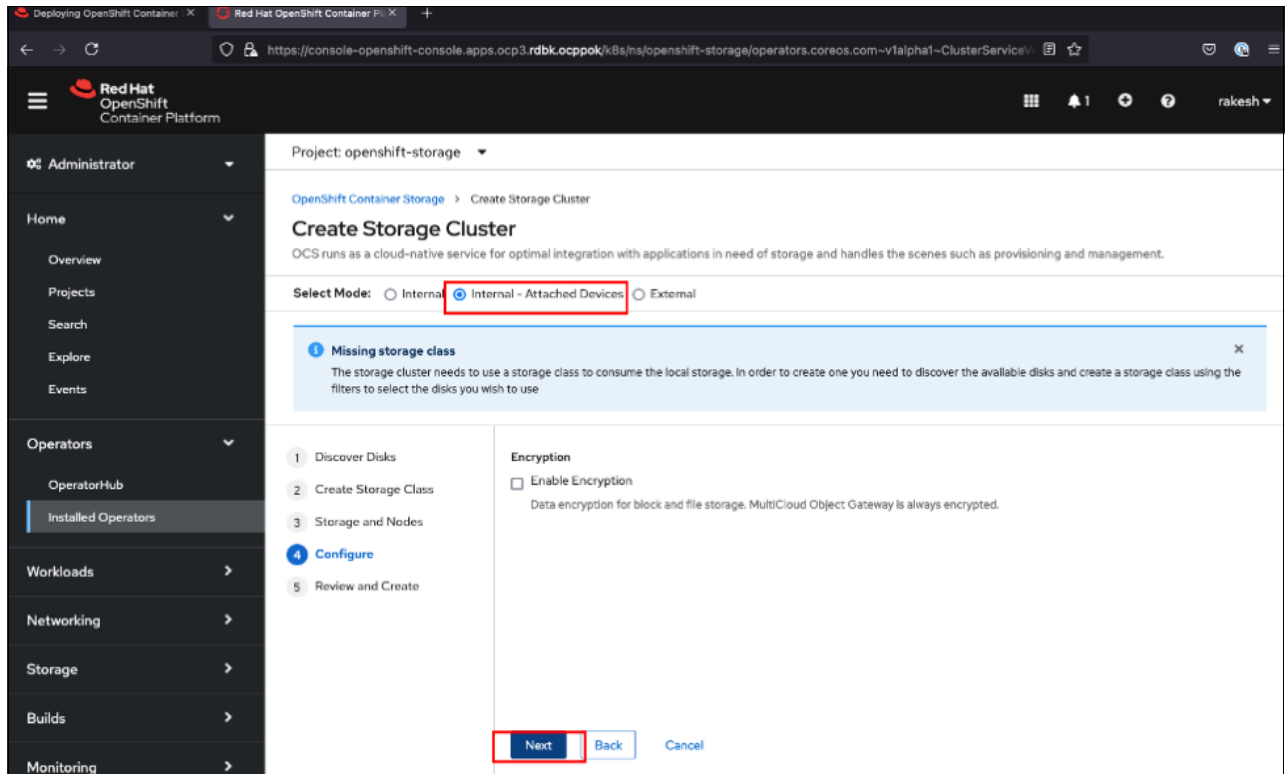


Figure A-22 StorageClass Encryption

20. Review and confirm the configurations by selecting the **Create** option (see Figure A-23).

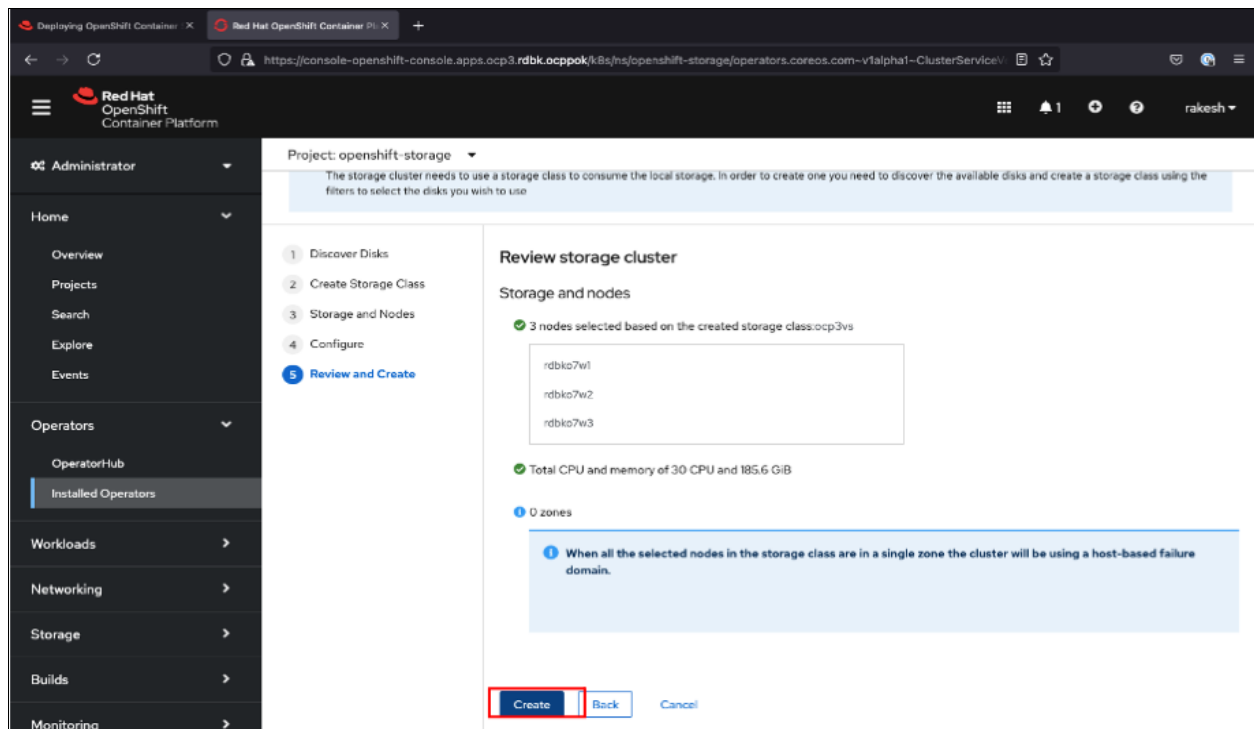


Figure A-23 Creating StorageClass

21. Select the Installed Operators from the Operators option that show the status as succeeded for Red Hat OpenShift Container Storage (see Figure A-24).

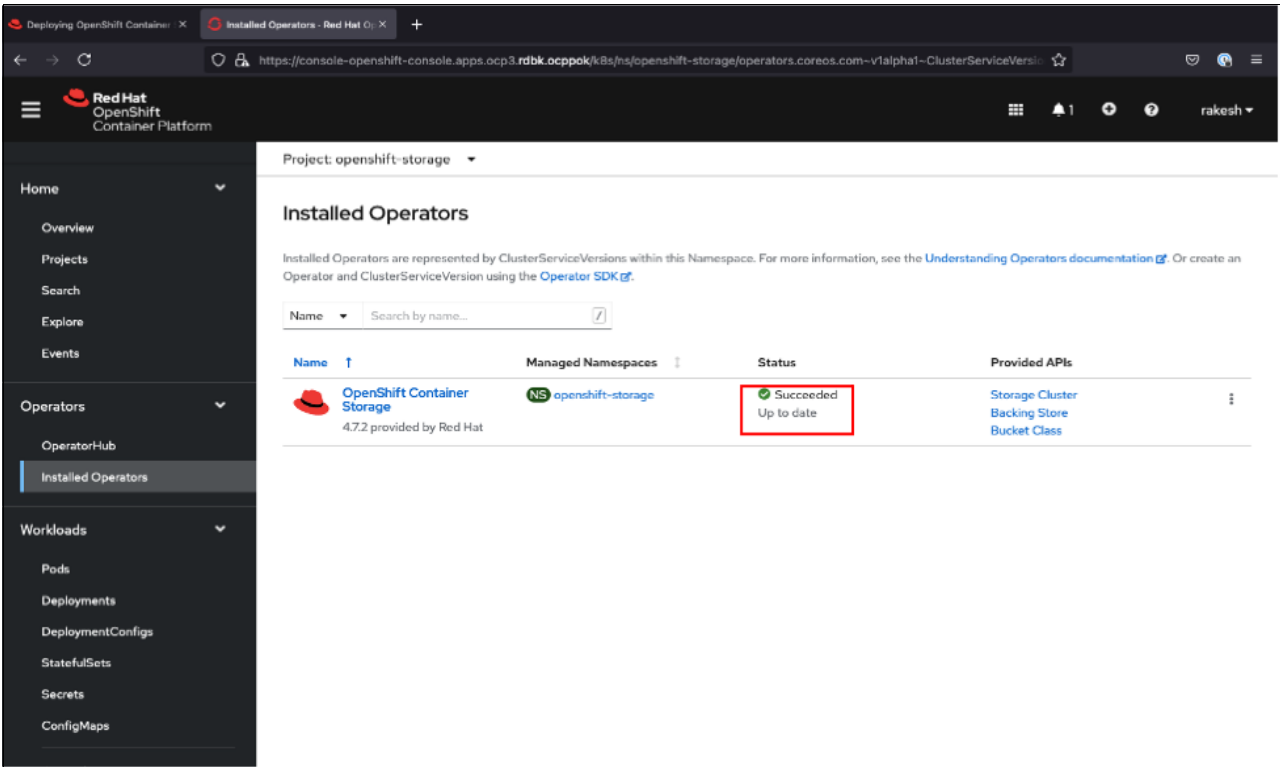


Figure A-24 Successful StorageClass creation

22. Verify the Red Hat OpenShift container Storage status by selecting the **Overview** option (see Figure A-25) and then, click the Persistent Storage option (see Figure A-26).

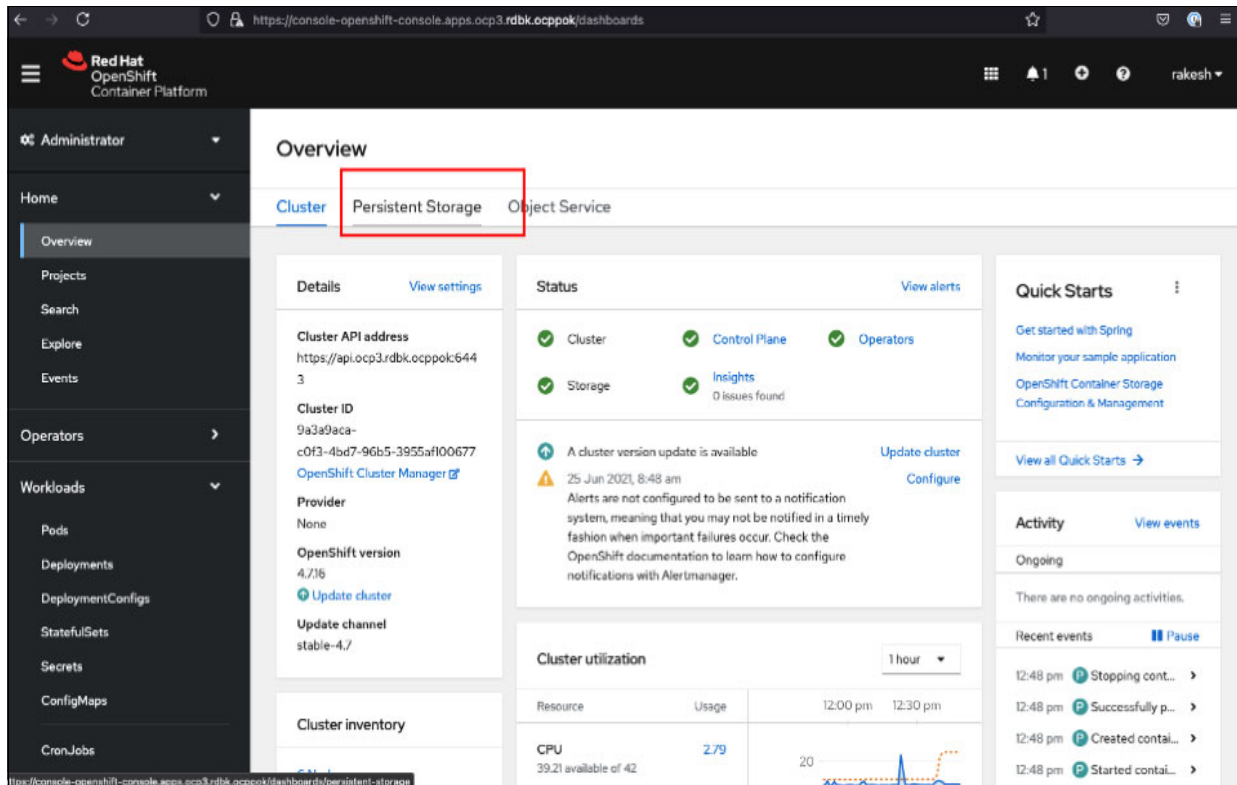


Figure A-25 Persistent storage overview

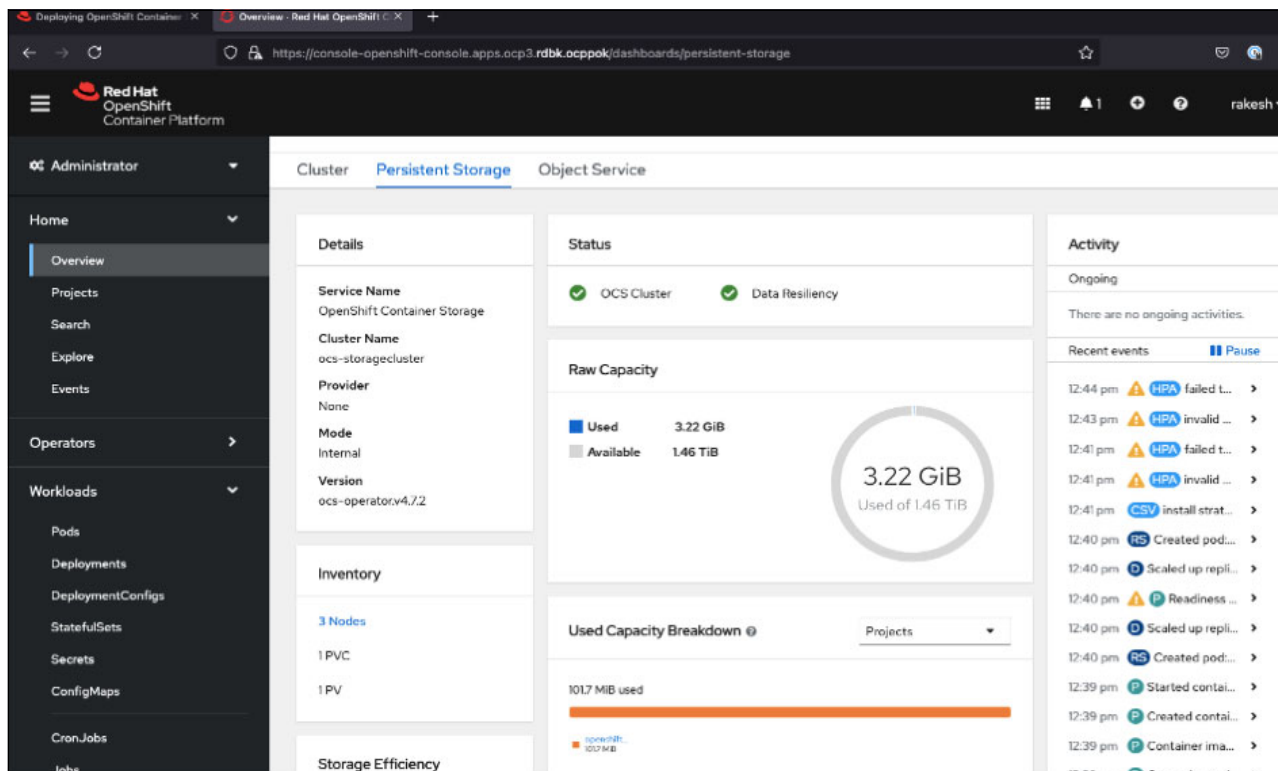


Figure A-26 OCS cluster status

A.5 Red Hat OpenShift Container Platform registry integration with Red Hat OpenShift Container Storage

Complete the following steps to patch Red Hat OpenShift Container Platform registry with Red Hat OpenShift Container Storage so that images are persistently stored across cluster:

1. Log on to Red Hat OpenShift Container Platform web console by using `kubeadmin` or `htpasswd` credentials.
2. Select **Storage** from the left side menu and then, click **StorageClasses** to list the available storage classes that were created during the installation of Red Hat OpenShift Container Storage (see Figure A-27).

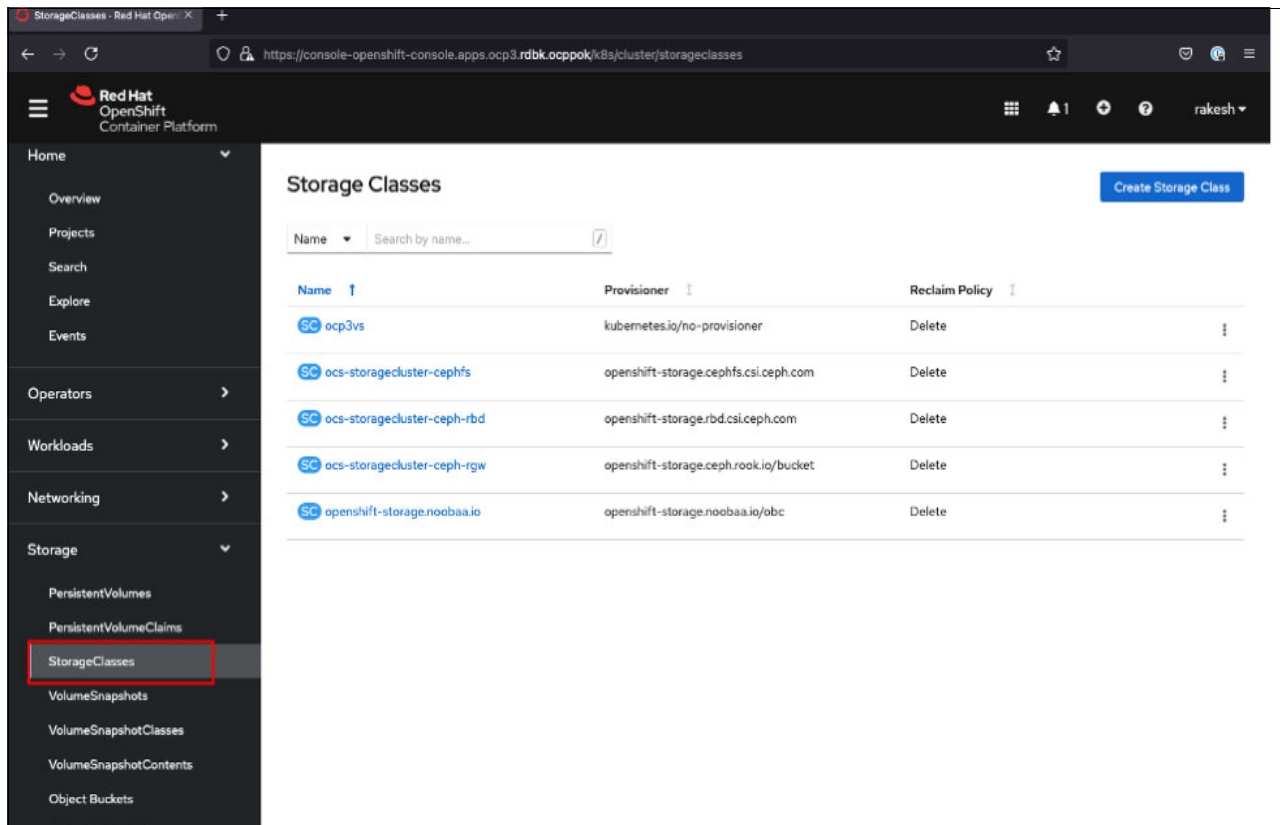


Figure A-27 StorageClasses listing

3. Red Hat OpenShift Container Platform registry supports shared file system storage for PVC because a block device-based PVC can have only a single replication controller.
Click the **Storage** option on the left side menu and browse to `PersistentVolumeClaim`. Select **Create PersistentVolumeClaim** (see Figure A-28 on page 127).

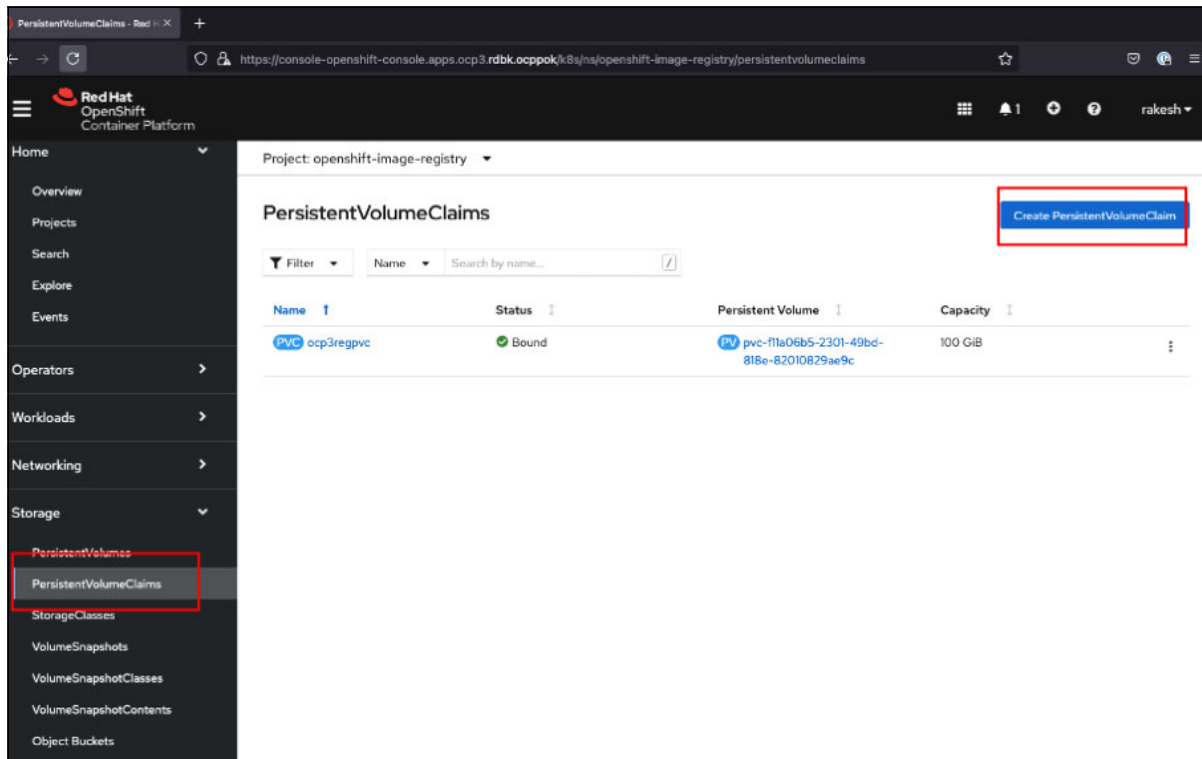


Figure A-28 Creating PVC

4. In the Select Storage class field, browse to ocs-storagecluster-cephfs (see Figure A-29).

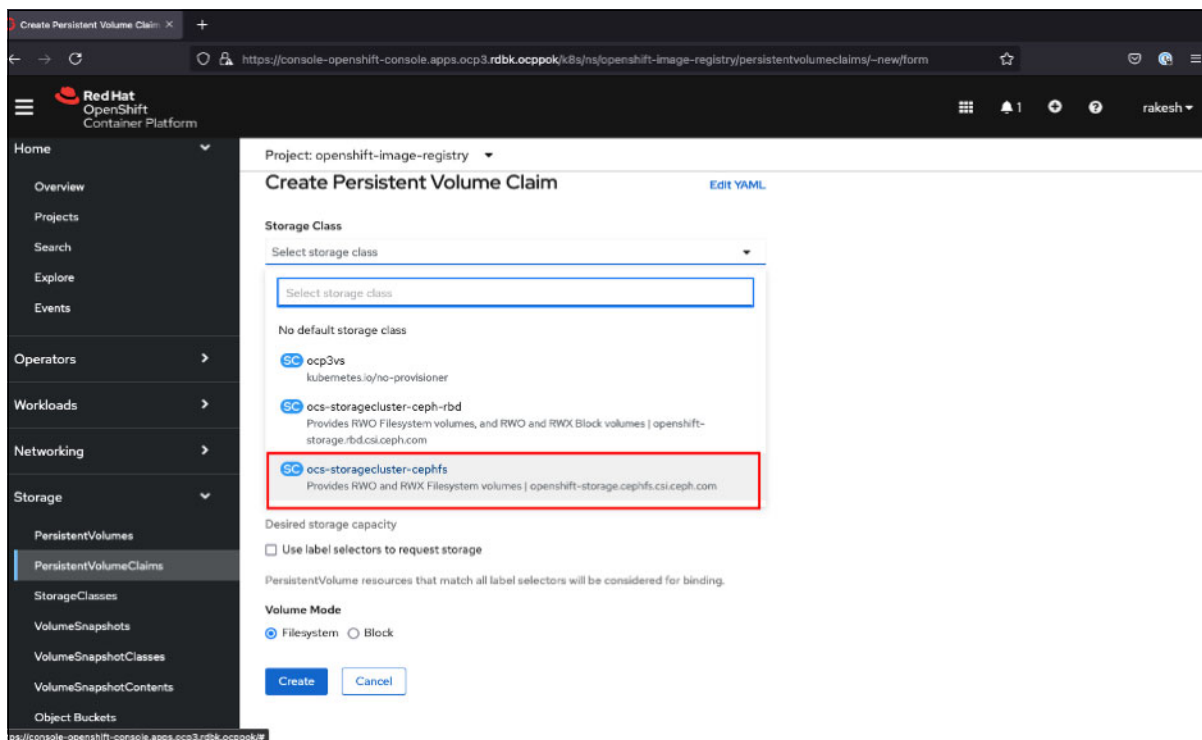


Figure A-29 Selecting the StorageClass

5. Enter the wanted storage capacity in the Size field and then, click **Create** (see Figure A-30).

The screenshot shows the 'Create Persistent Volume Claim' interface in the Red Hat OpenShift Container Platform. The left sidebar contains navigation links: Home, Overview, Projects, Search, Explore, Events, Operators, Workloads, Networking, and Storage. The 'Storage' section is expanded, showing 'PersistentVolumes', 'PersistentVolumeClaims' (selected), 'StorageClasses', 'VolumeSnapshots', 'VolumeSnapshotClasses', 'VolumeSnapshotContents', and 'Object Buckets'. The main content area is titled 'Create Persistent Volume Claim' and includes the following fields and options:

- Project:** openshift-image-registry
- Storage Class:** A dropdown menu showing 'ocs-storagecluster-cephfs'.
- Persistent Volume Claim Name:** A text input field containing 'testpvc'.
- Access Mode:** Radio buttons for 'Single User (RWO)' (selected), 'Shared Access (RWX)', and 'Read Only (ROX)'. A note below states: 'Access mode is set by storage class and cannot be changed'.
- Size:** A text input field containing '50' and a unit dropdown menu set to 'GiB'. This field is highlighted with a red box.
- Use label selectors to request storage:** An unchecked checkbox.
- Volume Mode:** A dropdown menu set to 'Filesystem'.
- Create and Cancel buttons:** Located at the bottom, with the 'Create' button highlighted by a red box.

Figure A-30 Selecting the PCVC size

- On successful creation of the PVC, the PVC status changes to Bound, as shown in Figure A-31.

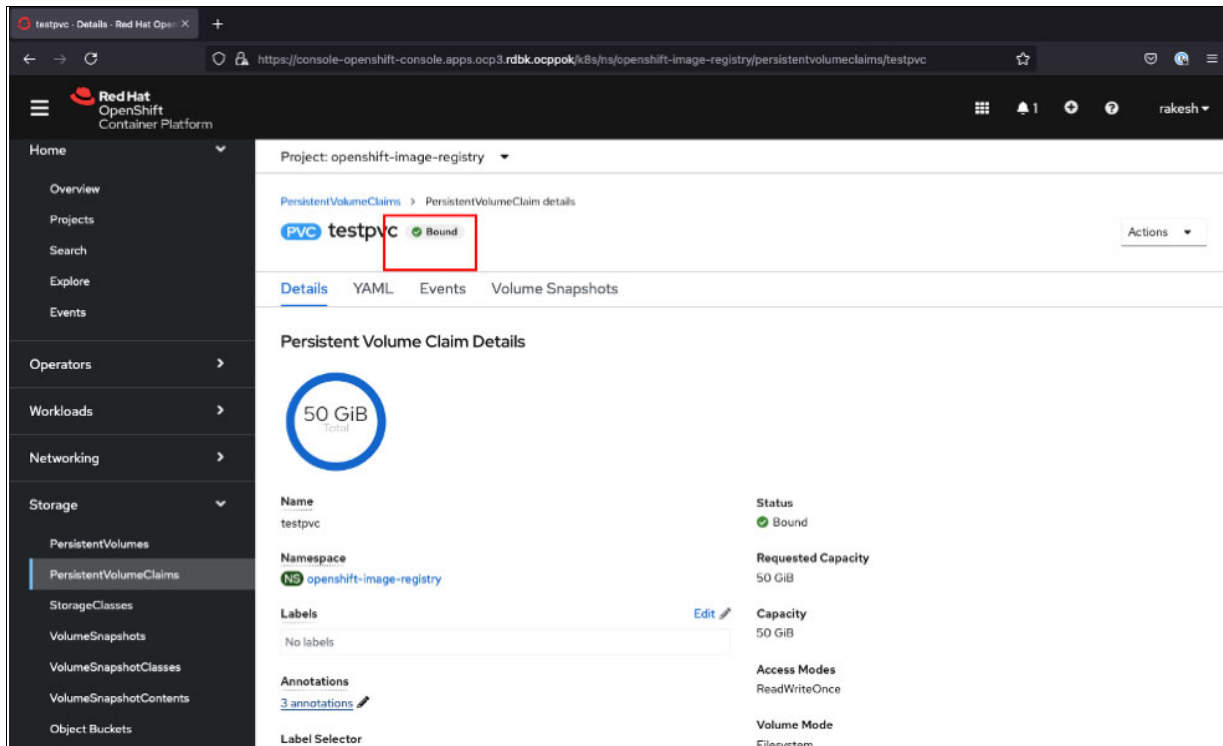


Figure A-31 PVC bound

- Using the Red Hat OpenShift cluster client, edit the openshift-image-registry configuration to include the newly created PVC in the storage resource, as shown in Example A-9. The configuration that was used in our lab was named ocp3regpvc (see Figure A-28).

Example A-9 Editing the configuration

```
[root@rdbkbas7 ~]# oc edit configs.imageregistry.operator.openshift.io/cluster

# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this
# file will be
# reopened with the relevant failures.
#
apiVersion: imageregistry.operator.openshift.io/v1
kind: Config
metadata:
  creationTimestamp: "2021-06-25T00:42:07Z"
  finalizers:
  - imageregistry.operator.openshift.io/finalizer
  generation: 27
  name: cluster
  resourceVersion: "3881558"
  selfLink: /apis/imageregistry.operator.openshift.io/v1/configs/cluster
  uid: 05fcbce9-543b-419b-b481-c55f43ae5438
spec:
  defaultRoute: true
```

```

httpSecret:
74b53647c9baf252698c581c451e7c56718e021f08a2769670cb48a4dfeddc5703d10d09fcc3460
38407260f387abf352fb70b43f94fdf280222d00a14f3411f
logLevel: Normal
managementState: Managed
observedConfig: null
operatorLogLevel: Normal
proxy: {}
replicas: 1
requests:
  read:
    maxWaitInQueue: 0s
  write:
    maxWaitInQueue: 0s
rolloutStrategy: Recreate
storage:
  managementState: Managed
  pvc:
    claim: ocp3regpvc
unsupportedConfigOverrides: null
status:
  conditions:
  - lastTransitionTime: "2021-06-25T00:42:07Z"
    reason: AsExpected
    status: "False"
    type: ImageConfigControllerDegraded
  - lastTransitionTime: "2021-06-25T00:42:07Z"
    reason: AsExpected
    status: "False"
    type: ImageRegistryCertificatesControllerDegraded

```

8. Verify the cluster operator status to ensure it is functional and not degraded. Use the command that is shown in Example A-10.

Example A-10 Verifying status

```

[root@rdbkbas7 ~]# oc get co

```

NAME	VERSION	AVAILABLE	PROGRESSING
DEGRADED SINCE			
authentication	4.7.16	True	False
False 10d			
baremetal	4.7.16	True	False
False 19d			
cloud-credential	4.7.16	True	False
False 19d			
cluster-autoscaler	4.7.16	True	False
False 19d			
config-operator	4.7.16	True	False
False 19d			
console	4.7.16	True	False
False 18d			
csi-snapshot-controller	4.7.16	True	False
False 19d			
dns	4.7.16	True	False
False 19d			

etcd	4.7.16	True	False
False 19d			
image-registry	4.7.16	True	False
False 9d			
ingress	4.7.16	True	False
False 10d			
insights	4.7.16	True	False
False 19d			
kube-apiserver	4.7.16	True	False
False 19d			
kube-controller-manager	4.7.16	True	False
False 19d			
kube-scheduler	4.7.16	True	False
False 19d			
kube-storage-version-migrator	4.7.16	True	False
False 10d			
machine-api	4.7.16	True	False
False 19d			
machine-approver	4.7.16	True	False
False 19d			
machine-config	4.7.16	True	False
False 19d			
marketplace	4.7.16	True	False
False 18d			
monitoring	4.7.16	True	False
False 11d			
network	4.7.16	True	False
False 19d			
node-tuning	4.7.16	True	False
False 19d			
openshift-apiserver	4.7.16	True	False
False 19d			
openshift-controller-manager	4.7.16	True	False
False 19d			
openshift-samples	4.7.16	True	False
False 19d			
operator-lifecycle-manager	4.7.16	True	False
False 19d			
operator-lifecycle-manager-catalog	4.7.16	True	False
False 19d			
operator-lifecycle-manager-packageserver	4.7.16	True	False
False 19d			
service-ca	4.7.16	True	False
False 19d			
storage	4.7.16	True	False
False 19d			

[root@rdbkbas7 ~]#

Related publications

The publications that are listed in this section are considered particularly suitable for a more detailed discussion of the topics that are covered in this book.

IBM Redbooks

The IBM Redbooks publications *Red Hat OpenShift on IBM Z Installation Guide*, REDP-5605, provides more information about the topic in this document. Note that this publication might be available in softcopy only.

You can search for, view, download or order this document and other Redbooks, Redpapers, Web Docs, draft, and additional materials, at the following website:

ibm.com/redbooks

Online resources

The following websites are also relevant as further information sources:

- Open Container Initiative:

<https://www.opencontainers.org/>

- Red Hat Enterprise Linux CoreOS:

<https://docs.openshift.com/container-platform/4.7/architecture/architecture-rhcos.html>

- Red Hat OpenShift Understanding persistent storage:

<https://docs.openshift.com/container-platform/4.7/storage/understanding-persistent-storage.html>

- MacVTap driver considerations:

<https://www.ibm.com/docs/sl/linux-on-systems?topic=cons-macvtap-driver-considerations>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



SG24-8515-00

ISBN 0738460745

Printed in U.S.A.

Get connected

