

# Using the IBM Block Storage CSI Driver in a Red Hat OpenShift Environment

Detlef Helmbrecht

Simon Casey

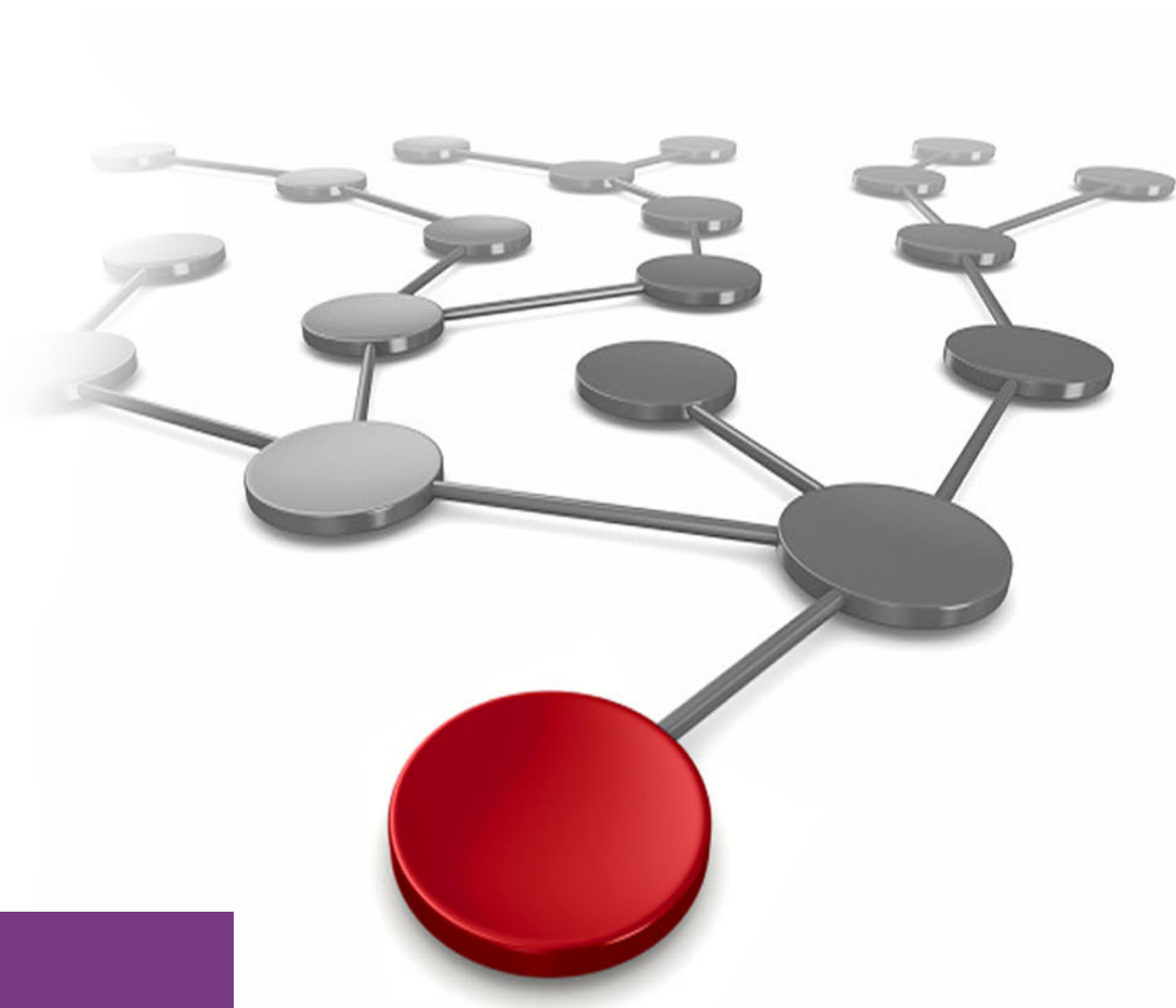
Mathias Defiebre

Bert Dufrasne

Michael Schaefer

Harald Seipp

Ralf Wohlfarth



Storage





IBM Redbooks

**Using the IBM Block Storage CSI Driver in a Red Hat OpenShift Environment**

May 2021

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

**First Edition (May 2021)**

This edition applies to Red Hat OpenShift Container Storage 4.3, 4.4. and 4.5 and IBM block storage CSI driver version 1.3.

This document was created or updated on May 12, 2021.

**© Copyright International Business Machines Corporation 2021. 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

<b>Notices</b> .....	v
Trademarks .....	vi
<b>Preface</b> .....	vii
Authors .....	viii
Now you can become a published author, too! .....	x
Comments welcome .....	x
Stay connected to IBM Redbooks .....	x
<b>Chapter 1. Introduction and concepts</b> .....	1
1.1 Overview of containers, orchestration, and OpenShift .....	2
1.2 Container persistent data challenges .....	4
1.2.1 Container Storage Interface .....	5
1.2.2 OpenShift Container Storage .....	7
1.2.3 OCS deployment approaches .....	8
<b>Chapter 2. OpenShift Container Storage and IBM Cloud Paks</b> .....	11
2.1 IBM Cloud Paks .....	12
2.2 IBM Storage Suite for Cloud Paks .....	13
2.3 IBM and Red Hat Storage architecture for OpenShift and IBM Cloud Paks .....	15
2.4 IBM DS8000 family and Red Hat Storage architecture for OpenShift and IBM Cloud Paks .....	16
2.5 IBM Cloud Pak storage requirements .....	18
<b>Chapter 3. Container Storage Interface architectural overview</b> .....	19
3.1 Kubernetes .....	20
3.1.1 Kubernetes platform .....	20
3.1.2 Control plane .....	21
3.1.3 Nodes .....	22
3.1.4 pods .....	23
3.1.5 Workload controllers .....	24
3.1.6 Persistent storage .....	30
3.1.7 Application configuration .....	33
3.1.8 Extension points in Kubernetes: The operator pattern .....	34
3.2 Kubernetes Container Storage Interface .....	36
3.2.1 Volume lifecycle .....	37
3.2.2 CSI driver deployment .....	38
3.3 IBM block storage CSI driver .....	41
3.3.1 IBM CSI Operator .....	41
3.3.2 IBM CSI controller .....	42
3.3.3 IBM CSI node .....	42
3.3.4 CSI driver storage back-end communication .....	43
<b>Chapter 4. OpenShift and Container Storage Interface deployment</b> .....	45
4.1 IBM block storage Container Storage Interface .....	46
4.1.1 CSI configuration overview .....	46
4.2 Installing Red Hat OpenShift Container Platform in an IBM Power Systems PowerVM environment .....	52
4.2.1 Red Hat OCP 4.3 on Power installation overview .....	53

4.3 CSI deployment on IBM Power by using the command-line interface . . . . .	72
4.3.1 Configuring the storage system . . . . .	73
4.3.2 Configuring the multipath driver on the worker nodes. . . . .	73
4.3.3 Installing the driver by using CLI . . . . .	75
4.3.4 Configuring the CSI driver by using the CLI . . . . .	78
4.4 CSI and OpenShift on IBM Z. . . . .	87
4.4.1 Enabling FCP adapters. . . . .	87
4.5 CSI deployment on Z by using the command-line interface . . . . .	91
4.5.1 Deploying the operator . . . . .	91
4.5.2 Deploying the driver . . . . .	92
4.5.3 Configuring the storage backend . . . . .	93
4.6 Installing the CSI driver by using the OpenShift web console on x86. . . . .	95
4.6.1 Fulfilling installation prerequisites. . . . .	96
4.7 Updating the CSI driver. . . . .	110
4.7.1 Updating the CSI driver by using a subscription . . . . .	110
4.7.2 Updating the CSI driver manually . . . . .	111
<b>Chapter 5. Maintenance and troubleshooting</b> . . . . .	<b>113</b>
5.1 General hints. . . . .	114
5.2 Studying the good case. . . . .	114
5.3 CSI operator issues. . . . .	117
5.4 CSI driver CustomResource issues . . . . .	118
5.5 CSI driver snapshot feature . . . . .	119
5.6 CSI driver operation issues. . . . .	120
5.6.1 Configuration issues . . . . .	120
5.6.2 PVC not binding . . . . .	123
5.6.3 Volume not mounting . . . . .	123
5.7 Searching the CSI related log entries . . . . .	124
5.8 Changing the CSI driver subscription by using the CLI. . . . .	125
<b>Appendix A. Terminology</b> . . . . .	<b>127</b>
Term and acronyms . . . . .	128
<b>Appendix B. Container Storage Interface support matrix</b> . . . . .	<b>131</b>
IBM block storage CSI driver compatibility and requirements . . . . .	132
Red Hat OpenShift access modes for persistent storage . . . . .	133
Red Hat OpenShift CSI volume snapshot support. . . . .	133
<b>Related publications</b> . . . . .	<b>135</b>
IBM Redbooks . . . . .	135
Other publications . . . . .	135
Online resources . . . . .	135
IBM Knowledge Center Cloud Pak documentation . . . . .	136
Help from IBM . . . . .	137

# 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](http://www.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.

AIX®	IBM Elastic Storage®	PureSystems®
DB2®	IBM FlashSystem®	Redbooks®
DS8000®	IBM Spectrum®	Redbooks (logo)  ®
FlashCopy®	IBM Z®	Storwize®
IBM®	Informix®	XIV®
IBM Cloud®	POWER8®	z/VM®
IBM Cloud Pak®	PowerVM®	

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

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

VMware, VMware vSphere, and the VMware logo are registered trademarks or trademarks of VMware, Inc. or its subsidiaries in the United States and/or other jurisdictions.

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



# Preface

RedHat OpenShift container platform is one of the leading enterprise-grade container orchestration platforms. It is designed for rapid deployment of web applications, databases, and microservices.

Categorized as a container orchestration Platform as a Service (PaaS), it is based on open industry standards, such as the Container Runtime Interface - Open (CRI-O) and Kubernetes. OpenShift allow developers to focus on the code, while the platform manages the complex IT operations and processes.

Although open-source, community-driven container orchestration platforms are available, such as OKD and Kubernetes, this IBM® Redpaper® publication focuses on Red Hat OpenShift. It describes the basic concepts of OpenShift persistent storage architecture and its integration into IBM Cloud® Paks. The deployment of the IBM block storage CSI driver also is discussed.

This publication also describes the concepts, technology and current working practices for installing the Container Storage Interface (CSI) plug-in for Kubernetes to use IBM Enterprise Storage platforms for persistent storage coupled with Red Hat OpenShift Container Platform (OCP).

This publication also provides an overview of containers, Kubernetes, and Openshift for context (it is expected that the reader has a working knowledge of these underlying technologies). It also includes architectural examples of the orchestration platform will be given.

This paper serves as a guide about how to deploy the CSI driver for block storage by using the DS8000® and Spectrum Virtualize platforms as persistent storage in a Red Hat OpenShift platform.

The publication is intended for storage administrators, IT architects, OpenShift technical specialists and anyone who wants to integrate IBM Enterprise storage on OpenShift V4.3/4.4/4.5 on IBM Power, IBM Z®, and x86 systems.

This paper complements documentation that is available at [IBM Documentation](#).

**Note:** OpenShift and the CSI driver are under continuous development, with frequent code updates. Contents of this IBM Redpaper publication might not reflect the latest updates, but the information provided still provides a helpful basis and reference for upcoming OpenShift and CSI driver versions.

# Authors

This paper was produced by a team of specialists from around the world:



**Detlef Helmbrecht** is an Advanced Technical Skills (ATS) IT Specialist working for the IBM Systems. He is in the EMEA Storage Competence Center (ESCC) in Kelsterbach, Germany. Detlef has over 35 years of experience in IT, performing various roles, including software engineer, sales, and solution architect. His areas of expertise include high-performance computing (HPC), disaster recovery, archiving, application tuning, and IBM FlashSystem®. He is the author of several IBM Redbooks® publications about IBM storage. He holds a degree in Mathematics and is the author of several Storage Redbooks publications.



**Simon Casey** is a Technical Lead and founding member of the Cloud Advisory Services practice in EMEA and is based out of IBM Hursley Labs. With 20 years IT experience, he has a broad, practical application of a multitude of technologies encompassing multiple compute architectures, storage, virtualization and orchestration platforms. He also is a member of IBM's Technical Consultants Group for technical leadership and provides input into proof of concepts and technologies and new initiatives within the company. Simon is an author of several Redbooks publications that span IBM Power Systems, IBM PureSystems®, and Cloud Object Storage technologies.



**Mathias Defiebre** is a leading IBM expert for Analytics, Object Storage, and Data Protection with over 20 years of storage experience. From IBM's EMEA Storage Competence Centre (ESCC), he provides support to customers through the Advanced Technical Skills (pre-sales support) and Lab Services channels (Implementations, Migrations, Health checks, Proof of Concepts, and Workshops). He graduated from the University of Cooperative Education Mannheim with a German Diploma in Information Technology Management and a Bachelor of Science. Mathias also is a Master Certified IT Specialist and an IBM Certified Specialist for TotalStorage Networking and Virtualization Architectures. He is the author of several Storage-related IBM Redbooks publications.



**Bert Dufrasne** is an IBM Certified Consulting IT Specialist and Project Leader for IBM System Storage disk products for the Redbooks organization, in San Jose, California. He has worked at IBM in various IT areas. He has written many IBM publications, and has developed and taught technical workshops. Before Bert joined the IBM Redbooks organization, he worked for IBM Global Services as an Application Architect. He holds a master's degree in electrical engineering.



**Michael Schaefer** is a software developer for OpenShift on Z in Germany. He has 30 years of experience around UNIX and Linux operating systems, their management, software development, and compute center management. Over the past few years, he has concentrated on cloud computing as site reliability and is a DevOps engineer for the IBM Cloud public cloud offerings, namely the CloudFoundry services. Today, he works as developer for RedHat OpenShift on the s390 architecture. He holds a master's degree for Computer Science (Diplom-Informatiker) from the University of Karlsruhe.



**Harald Seipp** is a Senior Technical Staff Member with IBM Systems in Germany. He is the founder and Technical Leader of the Center of Excellence for Cloud Storage as part of the IBM EMEA Storage Competence Center. He provides guidance to worldwide IBM teams across organizations, and works with customers and IBM Business Partners across EMEA to create and implement complex storage cloud architectures. His more than 25 years of technology experience includes previous job roles as Software Developer, Software Development Leader, and Lead Architect for successful software products, and co-inventor of an IBM storage product. He holds various patents on storage and networking technology.



**Ralf Wohlfarth** is an IT Management Consultant in the IBM European Storage Competence Center in Kelsterbach and has been working as Advanced Technical Skills (ATS) in technical sales support with focus on the IBM FlashSystem A9000 and IBM XIV® Storage System since 2008. He is a known speaker about different IBM Technical Universities and joined IBM in 1998. Before his role in technical sales support, he also worked in the last level product support for IBM System Storage and Software, including assignments in the US, Israel, Dubai, and other countries.

Thanks to **Ingo Dimmer**, EMEA Storage Competence Center (ESCC) in Kelsterbach, Germany, for his contributions to this project. Ingo contributed his knowledge and experience for OpenShift on IBM Power.

## 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](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

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

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

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an email to:

[redbooks@us.ibm.com](mailto: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

- ▶ Look for 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>



# Introduction and concepts

This chapter defines the scope of this publication. It reviews the concepts of containers, container orchestration, and the challenges around managing persistent data storage for container orchestration platforms and environments.

This chapter includes the following topics:

- ▶ 1.1, “Overview of containers, orchestration, and OpenShift” on page 2
- ▶ 1.2, “Container persistent data challenges” on page 4

# 1.1 Overview of containers, orchestration, and OpenShift

Containers are self-contained, packaged, executable units of software that hold all of their required libraries and dependencies. These components are packaged in a way that they can be run in many environments or platform architectures, such as IBM Power, IBM Z, and x86 based systems.

Containers use functions and features in the Linux kernel that were initially available decades ago in FreeBSD and in AIX® Workload Partitions (WPARs) in 2007. The modern container era started in 2013 with the introduction of Docker for x86 based systems.

In contrast to virtual machines (VMs) or logical partitions (LPARs), containers virtualize the operating system (normally Linux) rather than virtualizing the underlying hardware. Each container includes only the application, libraries, and run times on which it is dependent. Removing the need to include the guest operating system is what makes containers so lightweight, fast, and portable.

Figure 1-1 shows the infrastructure component stack comparisons between VMs and containers.

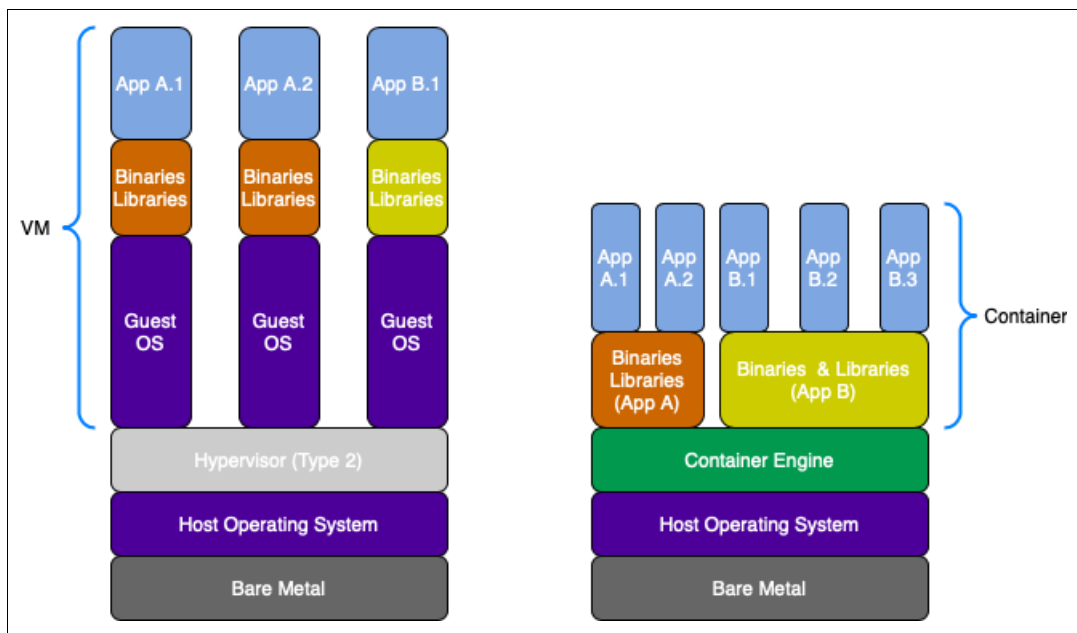


Figure 1-1 Virtual Machine Stack compared to Container Stack

Containers offer the advantage that they do not need to include an operating system image, which requires less physical resources than VM. As such, containers are a better fit for cloud deployment and applications that require horizontal scaling to meet demands and economies of scale.

Because containers package all their dependencies, their software generally can be written once and run without needing to be reconfigured across multiple platforms. These characteristics make containers an ideal fit for modern development methodologies, such as DevOps. They are key for emerging computing architectures that are server-independent and combine evolving paradigms, such as serverless computing and microservices, which allows for new application functions to be continuously added and deployed in small increments.

As adoption of containers increased, a requirement for suitable management of container environments grew, and container orchestration platforms emerged. As with VMs, containers and bare metal servers. They also benefit from a comprehensive orchestration tool to manage and automate areas, such as application deployment, scaling, and health of the environment. The most commonly used of these orchestration platforms is Kubernetes (K8s).

Kubernetes was released by Google in 2015. It was formed off an internal engineering platform that Google uses. Along with the initial release, Google partnered with IBM, Red Hat, Intel, Docker, and Cisco, among others, to form the Cloud Native Computing Foundation (CNCF) and seeded Kubernetes there. Full operational control of Kubernetes was handed over to the CNCF in 2018 and it is the second-most-worked on open source project after Linux.

Within the environment, the container orchestrator manages the provisioning of containers from a container registry: redundancy at a hardware and container level (otherwise known as ReplicaSets), health monitoring, resource allocation, scaling, load balancing, and moving containers between physical hosts.

Red Hat OpenShift Container Platform (OCP) is built upon Kubernetes and can be a fully fledged enterprise application platform. It augments Kubernetes with numerous cluster, developer, application, and platform services, such as automation, cloud-native apps, independent software vendor (ISV) services, Continuous Integration/Continuous Delivery (CI/CD) pipelines, charge back tools, and full stack logging.

Because OCP is based on Kubernetes, its fundamental base architecture is the same. The platform includes a control plane that consists of at least three master nodes, and a workload environment that contains two or more worker nodes. All nodes within the control plane are master nodes; therefore, the term *master* and *control plane* are used interchangeably to describe them. Figure 1-2 shows master and worker nodes.

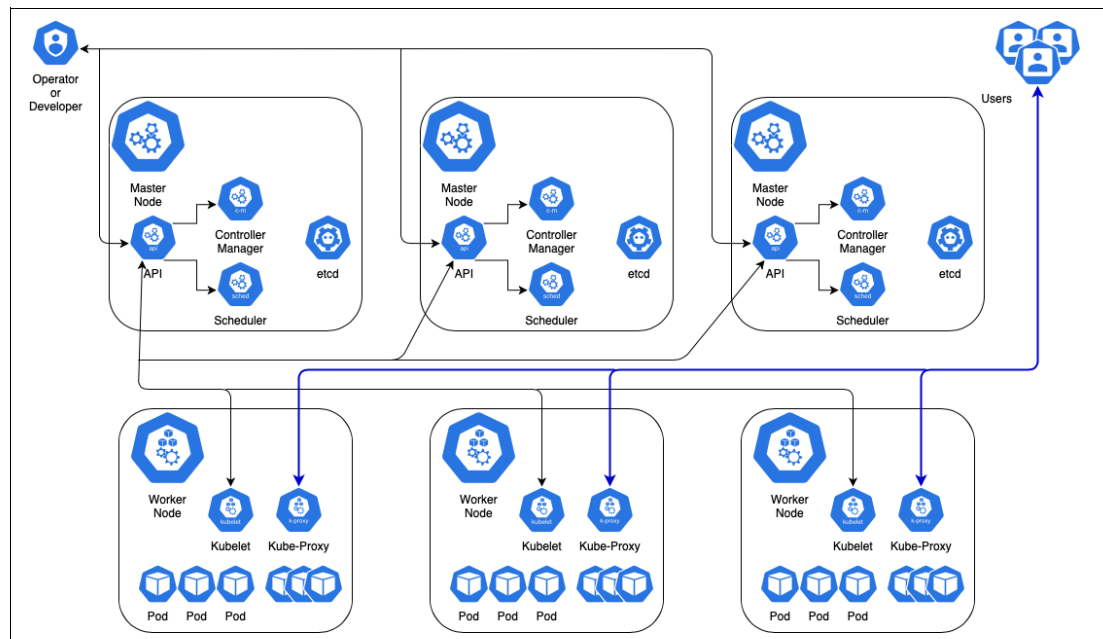


Figure 1-2 Kubernetes Master and Worker Node overview

The control plane makes global decisions about the platform, such as scheduling, detecting, and responding to cluster events. The master nodes in the cluster run the following Kubernetes services:

- ▶ Application programming interface (API) Server  
This service serves the front end for the control plane, and validates and configures the data in pods, services, and replication controllers.
- ▶ etcd  
This service is the *key value* store that is used as the backing store for all cluster data and is essential to the function of Kubernetes. Because Kubernetes is stateful in nature, the performance and backing up of etcd data is vital.
- ▶ Controller Manager Service  
Controllers run as a single process on each node, but perform several roles and can be summarized by using the following categories:
  - Node Controller: Notices and responds when nodes go down.
  - Replication Controller: Responsible for maintaining the correct number of pods and replicas for every object.
  - Endpoint Controller: Populates endpoint objects, such as services and pods.
  - Service Account Controller: Creates accounts and API tokens for namespaces.

For more information about OCP architecture as it pertains to a deployment with IBM storage and Container Storage Interface (CSI), see Chapter 3, “Container Storage Interface architectural overview” on page 19.

To learn more about containers and container orchestration, [this web page](#).

## 1.2 Container persistent data challenges

When containers are created because of their ephemeral, self-contained, portable nature, they are not tied to persistent storage unless defined in the container’s `StorageClass` property.

By default, when a container is created from its master image, an ephemeral read/write layer is created that handles all written data and is ephemeral. When the container stops, whether intentionally terminated, unintentionally terminated, or because the underlying pod failed, that read/write data layer disappears along with the container. That is, any writes that are performed to the container are limited to that container’s lifetime. Even if a container is restarted by the orchestrator, the storage that is written to the ephemeral layer in the old container is lost.

Not all storage for containers can be ephemeral and the nature of some applications (for example, databases) need some persistent storage for work that is done by non-trivial containers, or the ability to share some data or files between a pod or `ReplicaSet` of containers. A pod is the smallest workload entity in Kubernetes and it can consist of one or more containers. A `ReplicaSet` ensures that a specific number of the same pod replicas of the pod are running at any time.

Two constructs can be used: volumes and persistent volumes.



The fundamental difference between a volume and a persistent volume is that a volume exists for the lifetime of the pod. If the pod persists, the volume also persists. When the pod ceases to exist, its volumes also cease to exist.

Multiple volumes and classes of volume can be used by a pod simultaneously, but volumes always are managed as non-persistent data that is bound to the lifecycle of the pod.

Persistent volumes are pieces of storage that were provisioned by the system administrator, or are dynamically provisioned by using a Storage Class and can be defined to persist longer than the lifecycle of any pod.

Kubernetes treats persistent volumes as a cluster resource, just as it treats a node as a cluster resource; that is, a resource that is available for something to call and use and lifecycle independent of any individual pod that uses the persistent volume.

Generally, when designing a deployment of OCP, the only requirement that exists for ephemeral storage for workloads that are normally locally assigned in the Worker nodes.

**Note:** It is a good practice for any persistent storage to be assigned outside of the node so that if a physical hardware encounters a failure, persistent volumes are still available when a pod is restarted elsewhere.

Use cases for allocating persistent block storage are database applications, such as IBM DB2®, MySQL, or Continuous Integration, and Continuous Delivery (CI/CD) products, such as Jenkins.

The main challenge with data volumes in a containerized environment is the mode of the storage volume:

- ▶ ReadWriteOnce (RWO)  
The volume can be mounted as read/write by a single pod.
- ▶ ReadOnlyMany (ROX)  
The volume can be mounted as read-only by many pods.
- ▶ ReadWriteMany (RWX)  
The volume can be mounted as read/write by many pods.

Persistent volumes in Kubernetes are allocated by a persistent volume claim (PVC), which chooses the volume mode. Although a volume can be mounted in multiple ways, it can be mounted in one mode at any time only.

When volumes are created, mounted, and then claimed by way of CSI, they can be set to ReadWriteOnce mode only.

## 1.2.1 Container Storage Interface

Previously, in Kubernetes, all storage device drivers and volume plug-ins were kept “in-tree”; therefore, their code was inherent to the core Kubernetes code. When a vendor wanted to add support for a new storage system or fix a bug in a plug-in for Kubernetes, they had to follow the Kubernetes release cycle. The Kubernetes project also wanted to reduce third-party code, which caused potential reliability or security issues in the core Kubernetes code.

The fundamental benefit of the CSI driver is that it allows Kubernetes to dynamically provision storage to bind to persistent volumes for use by stateful containers. Otherwise, storage was allocated before the environment, volumes were created, and then, claims were made by persistent volumes to bind those volumes. The autonomy that CSI brings allows greater response, scalability, and management of the platform as a whole, including better use of the underlying infrastructure.

In addition to dynamic provisioning, the CSI driver brings such capabilities as snapshot of volumes to then be attached to a new ReplicaSet, dynamic de-provisioning, and the ability to define thinly or thickly provisioned volumes.

The CSI was designed with the objectives of being an open specification for exposing block and file storage systems to container orchestration systems, Kubernetes being one of them. The CSI is maintained [on GitHub](#).

All CSI drivers can perform the following tasks by way of the defined CSI API:

- ▶ Dynamically provision or deprovision a volume
- ▶ Enable local storage device mapping; for example, lvm, device mapper
- ▶ Attach or detach a volume from a node
- ▶ Mount or unmount a volume from a node
- ▶ Consumption of block and mountable volumes (the latter for CSI file drivers)
- ▶ Create or delete a snapshot
- ▶ Provision a new volume from a snapshot

IBM features the following written CSI driver families:

- ▶ IBM block storage CSI driver, which is used by Kubernetes for persistent volumes, dynamic provisioning of block storage, and volume snapshots.
  - This driver supports the following storage systems:
    - IBM DS8000 family
    - IBM FlashSystem A9000/R family
    - IBM Spectrum® Virtualize based block-storage
- ▶ IBM Spectrum Scale CSI driver, for file-based storage.

**Note:** This IBM Redpaper publication describes the block storage driver. For convenience, the IBM CSI block storage driver is referred to as the *CSI driver* here.

For more information about the IBM Spectrum Scale CSI driver, see this [IBM Documentation web page](#).

CSI consists of two objects within Kubernetes: the CSI Operator and the CSI Driver. An Operator is a Kubernetes software extension that uses custom resources that are outside core Kubernetes. It also interfaces with the API server and acts as a custom controller.

Two methods are available to install the CSI driver in OpenShift (depending on platform): the OpenShift CLI and the OpenShift Web Console.

After the CSI driver is installed and running, relevant storage classes, constructs, and secrets must be created to use CSI, as described in 4.1, “IBM block storage Container Storage Interface” on page 46.

## 1.2.2 OpenShift Container Storage

Red Hat OpenShift Container Storage (OCS) is a software-defined storage orchestration platform for container environments (see Figure 1-3). OCS can be integrated into OCP from V4.5 onwards, as discussed in 2.3, “IBM and Red Hat Storage architecture for OpenShift and IBM Cloud Paks” on page 15.

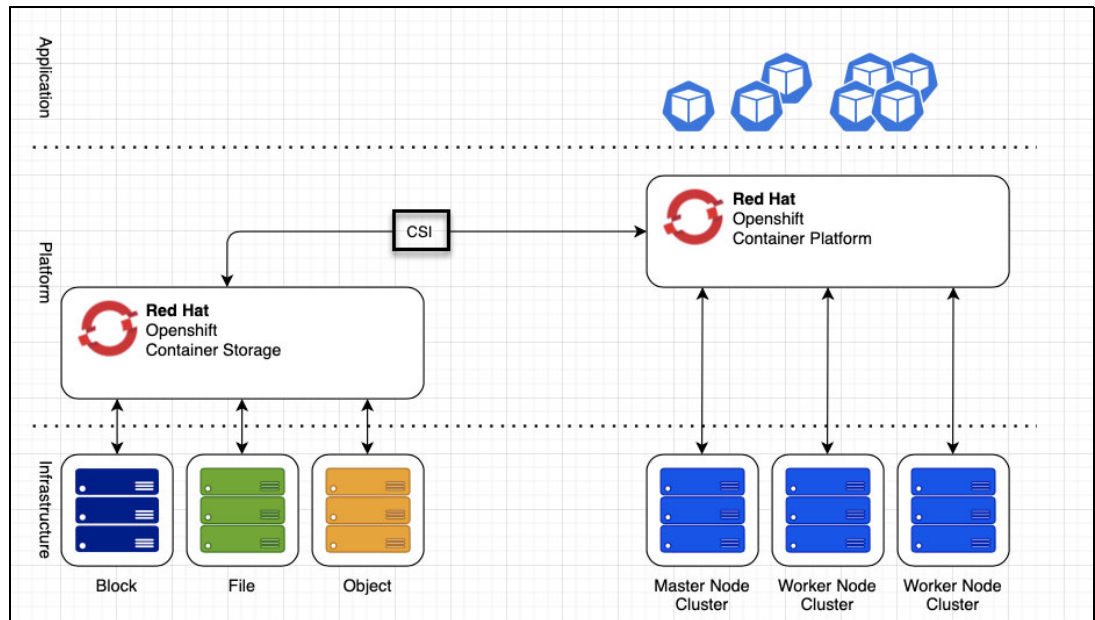


Figure 1-3 OpenShift Container Storage Platform serving OCP

OCS uses a technology stack that consists of Red Hat Ceph Storage, Rook.io as a storage operator, and NooBaa as a storage gateway behind which storage systems are knitted into a fabric design. OCS now uses CSI so that it can serve storage to platforms from pre-allocated storage and dynamically provision from storage subsystems that can use a CSI Driver, such as the IBM DS8000 family.

OCS is now packaged as an operator, and is available through the OCP Service catalog to allow an easy deployment and simplify management. OCS can be deployed by using two methods: an internal storage cluster and an external storage cluster. The platform provides the following types of storage services, which are exposed through storage classes:

- ▶ **Block Storage:** Primarily for database, logging, and monitoring workloads
- ▶ **Shared and distributed file:** For CI/CD tools, messaging, and data aggregation workloads
- ▶ **Object Storage:** Provides a lightweight S3 API Endpoint by way of NooBaa for abstraction of storage and retrieval from multiple object stores, which is ideal for cloud-native workloads or archival and backup data

The OCS platform uses the same stateful, declarative nature of Kubernetes. It codifies administrative tasks and custom resources, which makes automation of tasks and resources easier. Administrators can define the wanted state of the cluster and OCS operators can ensure that the cluster is in that state or approaching it while minimizing manual intervention.

For general-purpose persistent storage or dynamic provision requirements, OCS is suitable for consideration of workloads, such as data science and data analytics, artificial intelligence, machine learning, and Internet of Things workloads.

At the time of this writing, OpenShift Container Storage can be deployed on the following technologies:

- ▶ x86 Bare Metal using Red Hat Enterprise Linux
- ▶ Red Hat Virtualization Platform
- ▶ Red Hat OpenStack Platform
- ▶ VMware vSphere
- ▶ Amazon Web Services
- ▶ Google Cloud Platform (GCP): in technology preview
- ▶ Microsoft Azure: in technology preview

### 1.2.3 OCS deployment approaches

The core tenet of Red Hat OpenShift as a suite of platforms is flexibility. This tenet is evident by the fact that OCS can run as an internal service within an OCP deployment, or as an external deployment that can serve multiple OCP or Kubernetes based platforms.

#### Internal approach

OCS can be deployed entirely within an OCP deployment. This deployment brings the benefit of operator-based deployment and management within OCP. The following deployments are available, based on the requirements of your platform:

- ▶ Simple deployment

In a simple deployment, OCS runs its services alongside application workloads that are managed by OCP. A simple deployment is best suited when the following conditions are applicable to your OCP deployment:

- Storage requirements are unclear and the following storage factors are difficult to project:
  - Growth
  - Consumption
  - Performance
- Infrastructure has the room to run OCS services alongside OCP management or application workloads that are intermixing on the same nodes.
- Creating your platform by using a fixed “building-block” design method of the bare metal or virtualized infrastructure.
- Local direct-attach storage devices (DASD) are available within the node infrastructure.

For OCS services to run alongside OCP application workloads, local storage devices or portable storage devices must be made available. The internal, simple deployment uses only CSI Block provisioned storage if OCS manages a separate, externally based Ceph cluster.

- ▶ Optimized deployment

In an optimized deployment, OCS services run on dedicated infrastructure where the OCP management and the application workloads do not run on OCS nodes. The OCS nodes are still managed by the OCP deployment.

The optimized approach is best suited when the following conditions exist:

- Storage requirements are clear and are predictable to project growth, consumption, and performance.
- Dedicated infrastructure is available on which to run OCS services.
- Infrastructure is available in a well-suited “building-block” approach and can use a common node size in a cloud or on-premises data center.

- A farm is in place, or an as-a-service (aaS) model is available to provision OCS nodes rapidly with little to no manual intervention.

The internal, optimized deployment uses locally attached storage to the OCS nodes. It also uses only CSI Block provisioned storage if it manages a separate, externally based Ceph cluster.

## External approach

An external deployment of OCS makes available the Ceph storage service running outside your OCP deployment as storage classes to OCP. An external deployment must be considered when the following conditions exist:

- ▶ A significant storage need (600+ storage devices or persistent volumes)
- ▶ An environment with multiple OCP deployments that can use storage services from a common external cluster
- ▶ An organization that has a separate team or department responsible for managing storage services and separation of duties is still wanted

A single OCS deployment might be considered for use in a multi-tenancy model for higher usage and economies of scale of infrastructure. Also, when a clear exists need for separation of resource consumption from OCP workload platforms or nodes.

## OpenShift storage node types

In the same manner as an OCP deployment, OCS nodes run the container run times and services, which ensures that containers are running, and pod separation and maintenance of management network communications occurs. The following types of OCS nodes are available:

- ▶ **Master**

Master nodes run the Kubernetes API, watch and schedule pods, maintain node count and health, and control the interaction with underlying infrastructure providers.

- ▶ **Infrastructure (Infra)**

Infra nodes run the cluster level infrastructure services, such as logging and metrics, metering, registering containers, and routing for the cluster. Infrastructure nodes are optional. If no dedicated infra nodes exist, their services run across the available master nodes. Infra nodes are recommended when running OCS in cloud or virtualized environments.

- ▶ **Worker**

In an internal deployment mode, a minimum of three worker nodes are required to run OCS as a workload within an OCP deployment, ideally where the nodes are spread across racks or Availability Zones (AZs) to ensure availability of the platform. Worker nodes are required for an internal deployment only. These worker nodes also require internal storage for OCS to manage and serve OCP or predefined portable block storage volumes.





# OpenShift Container Storage and IBM Cloud Paks

This chapter discusses IBM Cloud Paks®, storage solutions for IBM Cloud Paks, and the importance of the IBM Container Storage Interface (CSI) driver.

This chapter includes the following topics:

- ▶ 2.1, “IBM Cloud Paks” on page 12
- ▶ 2.2, “IBM Storage Suite for Cloud Paks” on page 13
- ▶ 2.3, “IBM and Red Hat Storage architecture for OpenShift and IBM Cloud Paks” on page 15
- ▶ 2.4, “IBM DS8000 family and Red Hat Storage architecture for OpenShift and IBM Cloud Paks” on page 16
- ▶ 2.5, “IBM Cloud Pak storage requirements” on page 18

## 2.1 IBM Cloud Paks

IBM Cloud Paks are the heart of the full-stack, multi-cloud application architecture. These IBM Cloud Paks provides an open, faster, reliable way to build, move, and manage containerized applications on the cloud.

As shown in Figure 2-1, IBM Cloud Paks provide the industry's rich catalog of IBM and open source software containers and feature run times, along with orchestration capabilities for automated deployment with enterprise-grade configurations.

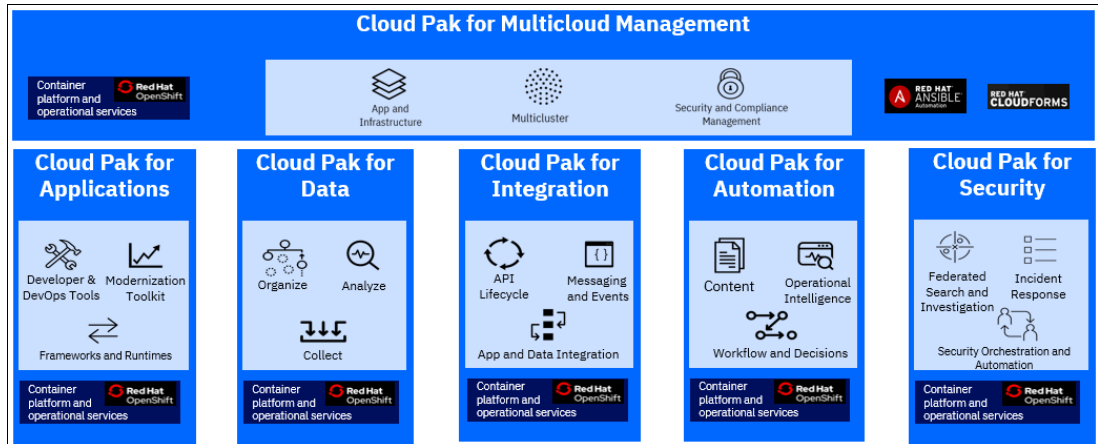


Figure 2-1 IBM Cloud Paks

The following IBM Cloud Pak types are available:

- ▶ IBM Cloud Pak® for Applications  
This IBM Cloud Pak can accelerate building cloud-native applications by using built-in developer tools and processes, including support for microservices functions and serverless computing.
- ▶ IBM Cloud Pak for Data  
By using this Pak, you can provision data and artificial intelligence (AI) services in minutes on-premises, instead of taking weeks.
- ▶ IBM Cloud Pak for Integration  
By using this Pak, enterprises can integrate across multiple clouds with a container-based platform that can be deployed across any on-premises or Kubernetes cloud environment. Applications, services, and data can be easily connected with the correct mix of integration styles, which spans API lifecycle management, application integration, enterprise messaging, Event Streams, and high-speed data transfer.
- ▶ IBM Cloud Pak for Automation  
This pre-integrated set of essential software enables you to easily design, build, and run intelligent automation applications at scale. With low-code tools for business users and real-time performance visibility for business managers, it is a flexible package with simple, consistent licensing.



- ▶ IBM Cloud Pak for Multicloud Management

To mitigate some of the complexity of a hybrid multicloud architecture, IBM Cloud Pak for Multicloud Management provides consistent visibility, automation, and governance across a range of multicloud management capabilities, such as cost and asset management, infrastructure management, application management, multi-cluster management, edge management, and integration with existing tools and processes.

- ▶ IBM Cloud Pak for Security

Your security data is frequently spread across different tools, clouds, and on-premises IT environments. This issue creates gaps that allow threats to be missed that often are solved by undertaking costly, complex integrations.

IBM Cloud Pak for Security provides an open security platform to help more quickly integrate your security tools to generate deeper insights into threats across hybrid, multicloud environments, by using an infrastructure-independent common operating environment that runs anywhere.

The platform helps you to find and respond to threats and risks, all while leaving your data where it is. Therefore, you can uncover hidden threats, make more informed, risk-based decisions, and respond to incidents faster.

The main goal of this IBM Cloud Pak is to help organizations detect, investigate, and respond to cybersecurity threats faster. Also, it helps to speed up your move to the cloud by facilitating the integration of their security tools to generate more in-depth insights into threats across hybrid, multicloud environments, by using an infrastructure-independent standard operating environment that runs anywhere.

All of these IBM Cloud Paks run on top of a Kubernetes-based orchestration platform that enables high availability, scalability, and ongoing maintenance for enterprise applications from a trusted source.

The software solutions run on a client's choice of infrastructure, including the following platforms:

- ▶ IBM Power Systems
- ▶ IBM Z, and x86
- ▶ IBM DS8000 family
- ▶ IBM Spectrum Virtualize family
- ▶ IBM FlashSystem
- ▶ IBM Spectrum Scale
- ▶ IBM Cloud Object Storage
- ▶ IBM Spectrum Protect Plus

IBM Cloud Paks consist of software packages, executables, and code templates, which are essentially IBM software-defined products. IBM Cloud Paks are sets of middleware tools, and different IBM Cloud Paks group software defined products for dedicated use cases.

## 2.2 IBM Storage Suite for Cloud Paks

IBM Storage Suite for IBM Cloud Paks is a complete, software-defined storage solution that helps build powerful, agile, and secure storage for hybrid multicloud environments.

It is not only about satisfying the basic storage requirements for deploying and running IBM Cloud Paks solutions. It is also about ensuring that container and application data is resilient, secure, and adaptable to grow as needs evolve.

Regardless of the state of application development and in which ever IBM Cloud Pak environment that is built (that is, Cloud Pak for Applications, Data, Integration, Automation, Multicloud Management, or Security), IBM Storage Suite for IBM Cloud Paks covers all possibilities, as shown in Figure 2-2.

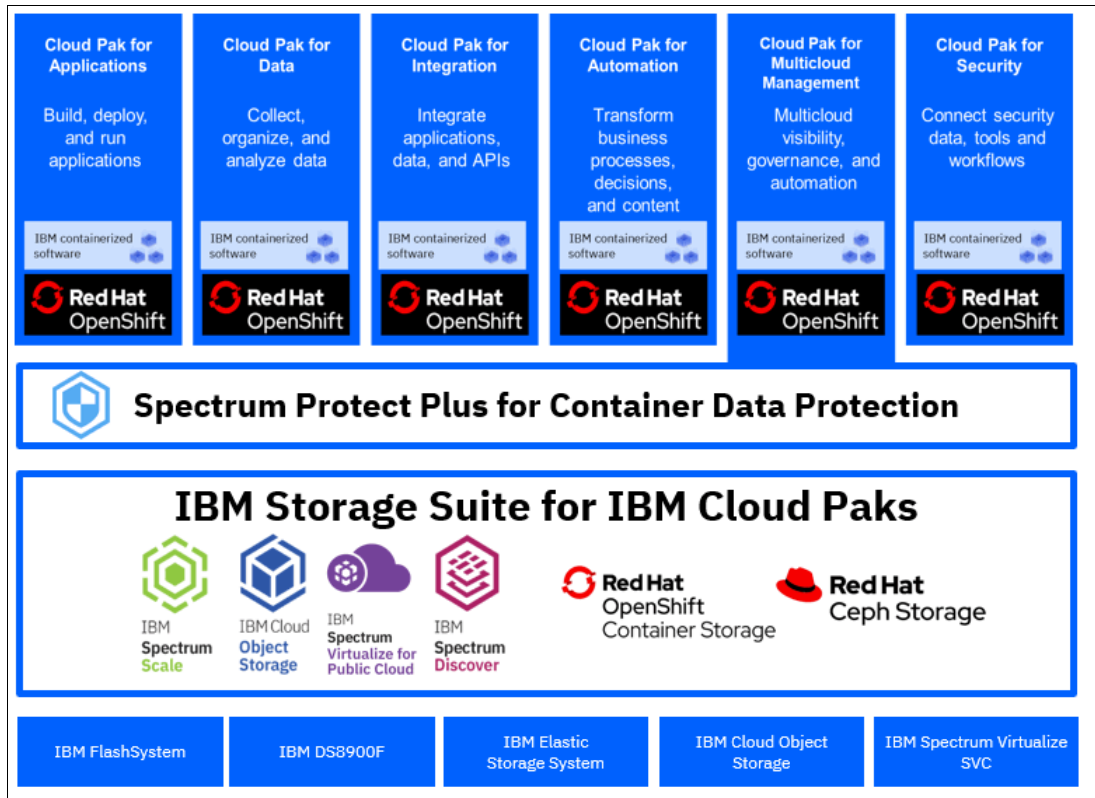


Figure 2-2 IBM Storage Suite for Cloud Paks in context of the overall IBM offering

IBM Storage Suite for IBM Cloud Paks information includes the following benefits:

- ▶ Provides enterprise backup and restore capabilities that support virtual machine (VM) and container environments
- ▶ Is modular, flexible, and easy to use with consistent data storage services
- ▶ Is licensed and priced to align with IBM Cloud Paks
- ▶ Is tested with Kubernetes and Red Hat OpenShift

As shown in Figure 2-2, IBM Storage Suite for Cloud Paks is complemented by the IBM Storage hardware offerings that comprise:

- ▶ IBM FlashSystem
- ▶ IBM DS8000 family
- ▶ IBM Elastic Storage® Server
- ▶ IBM Spectrum Virtualize Family and IBM SAN Volume Controller
- ▶ IBM Cloud Object Storage System

## 2.3 IBM and Red Hat Storage architecture for OpenShift and IBM Cloud Paks

The IBM Storage Suite for Cloud Paks supports the following storage options (the numbers in this list correlate to the numbers that are shown in Figure 2-3 on page 15):

1. IBM Spectrum Scale, as a high-performance parallel file system for AI and Data Lake use cases, provides ReadWriteOnce (RWO) and ReadWriteMany (RWX) storage through the IBM Spectrum Scale CSI driver
2. IBM Spectrum Virtualize for Public Cloud, which provides RWO storage through the IBM Block Storage CSI driver
3. IBM Spectrum Virtualize for Public Cloud, which provides RWO storage as backing storage for Red Hat OpenShift Container Storage (OCS). OCS also provides RWO and RWX block or file system storage through the OCS CSI driver and Object Storage.
4. IBM Cloud Object Storage and Red Hat Ceph storage, which provides Object Storage to OCS. That storage is made available through Object Bucket Claims to OpenShift applications that require Object Storage. The data access is facilitated by using the S3 API to the Object Storage http and https endpoint addresses and the associated credentials (Access Key and Secret Access Key).
5. IBM Cloud Object Storage and Red Hat Ceph can directly provide Object Storage to containerized applications.

Options 2 and 3 can be used by the software-only IBM Spectrum Virtualize for Public Cloud product and by the IBM Block Storage products, namely the IBM DS8000 family, IBM Spectrum Virtualize family, and IBM FlashSystem.

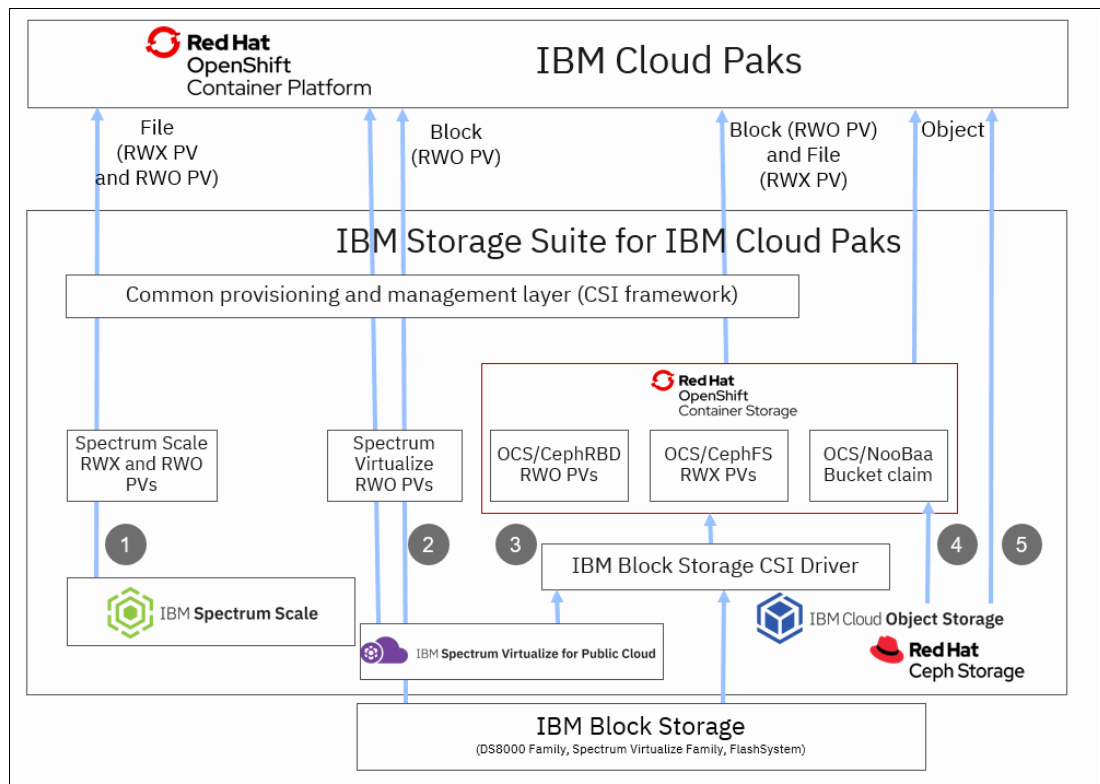


Figure 2-3 IBM and Red Hat storage options for OpenShift and IBM Cloud Paks

## 2.4 IBM DS8000 family and Red Hat Storage architecture for OpenShift and IBM Cloud Paks

The IBM Block Storage CSI driver supports block storage access to container applications. This support satisfies the storage requirements for applications, such as those applications that are included in the IBM Cloud Paks that require ReadWriteOnce (RWO) storage (see path 1 in Figure 2-4).

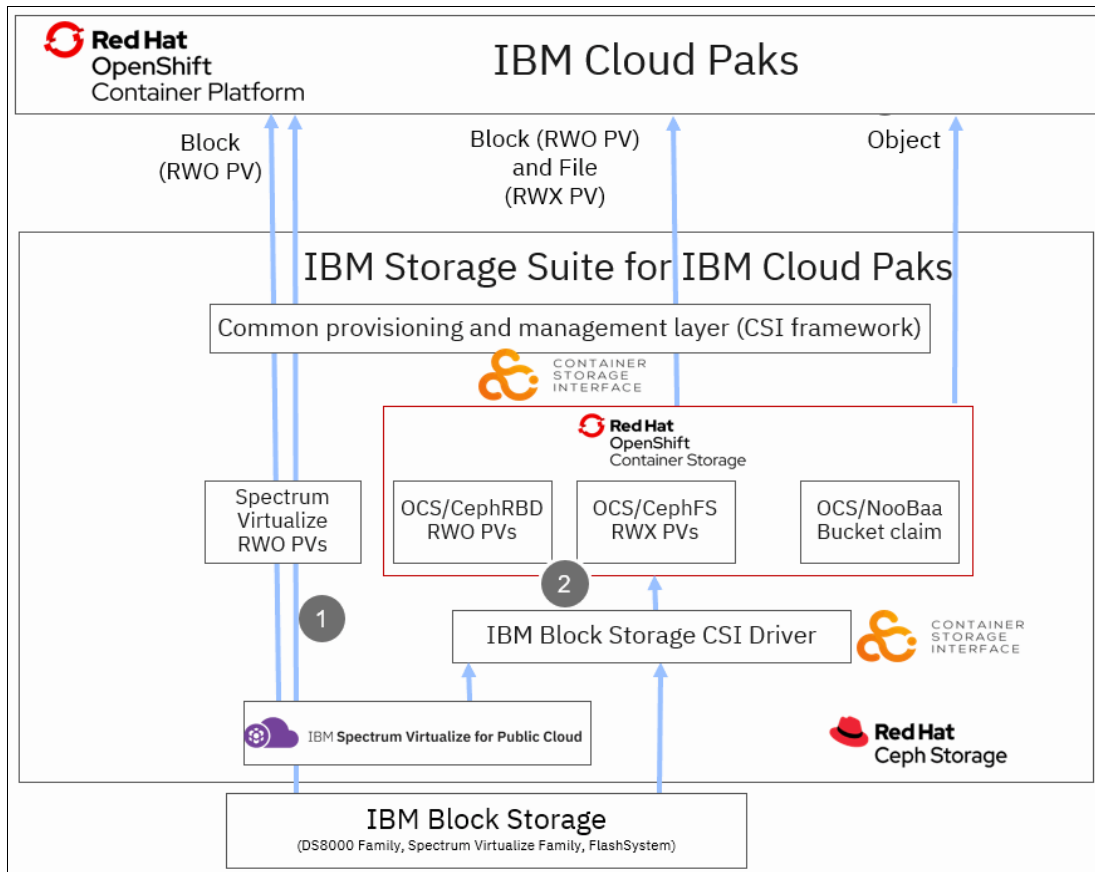


Figure 2-4 IBM Block Storage and Red Hat OpenShift Container Storage

Some applications that are included in the IBM Cloud Paks require ReadWriteMany (RWX) storage. For maximum reuse of block storage assets while keeping separation between Cloud infrastructure and Storage infrastructure where required, Red Hat OpenShift Container Storage (OCS) can be built with IBM Block Storage as backing storage (see path 2 in Figure 2-4) rather than server built-in disks.

That scenario requires OCS 4.5 or later or OpenShift on top of VMware, as shown in Figure 2-5.

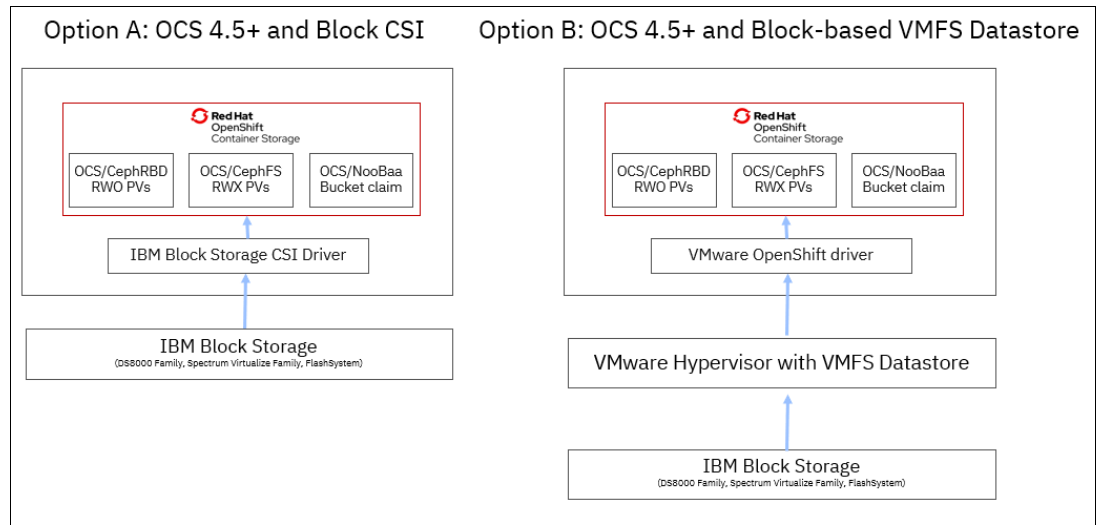


Figure 2-5 OCS on top of IBM Block Storage: CSI driver and VMware OpenShift driver

When Red Hat OpenShift Container Storage 4.5 and later versions are used, it is supported to select a CSI storage class as backing storage for OCS when the OCS storage cluster (Option A in Figure 2-5). That storage class can originate from the IBM Block Storage CSI driver so that PVCs that are backed by IBM Block storage serve as the data store for the storage components that are part of OCS.

For OCS versions 4.4 and earlier, the only supported option to use external block storage was through VMware VMFS data stores along with the VMware storage driver, only applying to scenarios where OpenShift is installed on top of VMware. The VMFS data stores are backed by IBM Block Storage.

At the time of this writing, OCS 4.5 supported only triple replication, which delivers only one-third usable capacity of the used IBM Block Storage backing storage capacity. Considering that the backing Block Storage is also protected with some data redundancy scheme (replication or distributed RAID/Erasure Coding), the overall storage efficiency of the solution is suboptimal.

An alternative, more storage-efficient approach is to use IBM Spectrum Scale to support RWX workload and use IBM Block storage volumes in shared access mode as backing storage for IBM Spectrum Scale (see Option 1 in Figure 2-3 on page 15).

## 2.5 IBM Cloud Pak storage requirements

As listed in Table 2-1, as part of the IBM Cloud Paks, the applications feature varying storage requirements. At the time of this writing, all IBM Cloud Paks are available for X86, but not all are available for OpenShift 4.x on IBM Power and IBM Z. After they become available, it is likely that the storage requirements are identical to those requirements for x86.

IBM Block Storage is recommended as backing storage for all IBM Cloud Paks, except IBM Cloud Pak for Data, which focuses on RWX storage. A potential combination of IBM OpenShift Container Storage or IBM Spectrum Scale as described in 2.4, “IBM DS8000 family and Red Hat Storage architecture for OpenShift and IBM Cloud Paks” can use IBM Block Storage as backing storage.

Table 2-1 IBM Cloud Pak Storage Access Mode Requirements

IBM Cloud Pak name	ReadWriteOnce required?	ReadWriteMany required?
IBM Cloud Pak for Applications	Yes (Primarily)	Yes (Few applications: CoreReady Workspaces and Codewind, Mobile Foundation)
IBM Cloud Pak for Data	Yes (Few applications: DB/2 partially, MongoDB, Informix®)	Yes (Primarily)
IBM Cloud Pak for Integration	Yes	Yes
IBM Cloud Pak for Automation	Yes	Yes
IBM Cloud Pak for Multicloud Management	Yes	Yes
IBM Cloud Pak for Security	Yes	No

For more information about access mode requirements that are listed in Table 2-1, see the “Related publications” in section “IBM Knowledge Center Cloud Pak documentation” on page 136.



# Container Storage Interface architectural overview

This chapter describes the architecture of the Kubernetes Container Storage Interface (CSI) and the IBM CSI driver for block storage systems.

Some relevant concepts in Kubernetes and OpenShift are introduced first because they help to understand the separation of duties and give the needed understanding for deploying the driver.

If you are familiar with the Kubernetes architecture and components, you might start with 3.2, “Kubernetes Container Storage Interface” on page 36, which describes the driver’s integration into the platform.

Kubernetes and OpenShift, and IBM’s block storage CSI driver, are open source projects, and every implementation detail also can be found on the internet. For more information about online resources, see “Related publications” on page 135.

This chapter includes the following topics:

- ▶ 3.1, “Kubernetes” on page 20
- ▶ 3.2, “Kubernetes Container Storage Interface” on page 36
- ▶ 3.3, “IBM block storage CSI driver” on page 41

## 3.1 Kubernetes

This section describes some of the most relevant concepts of Kubernetes to help the reader understand the IBM block storage for CSI driver and how it integrates “upwards” into an OpenShift environment, and “downwards” with IBM’s block storage products. These concepts and how the respective components work together give a basic understanding about how the driver is deployed and configured, and help with troubleshooting.

### 3.1.1 Kubernetes platform

Over the second half of the 2010s, Kubernetes evolved as the leading container orchestration system in the industry. All relevant cloud providers offer a Kubernetes or OpenShift service. The success of the platform seems to justify referring to it as the “operating system” of a cloud. As with a traditional operating system, it manages resources, which it presents in a consumable way to application developers who then build and run their applications on the platform.

The following basic design principles supported the success of the platform:

- ▶ Containerize everything

All types of programs or workloads, whether internal to the platform or user applications, are put into containers. These containers are self-contained pieces with all of their dependencies packed into one binary object; that is, the image. Kubernetes orchestrates containers; the platform organizes containers in a way to make them run “somewhere” in a cluster and provides all needed connectivity, be it among each other, to the network, storage, or other back-end systems.

- ▶ The Twelve-Factor App (for more information, see [this website](#))

An application design methodology that leads to applications that can easily be deployed, scaled, and most importantly be automatically managed. Kubernetes’ internal components follow that principle, and user workloads following that principle also are a perfect fit to run on the platform. This method is a powerful driver for the development of applications that can be containerized.

- ▶ Declarative approach

This paradigm is most contrary to the procedural approach to which “traditional” system managers and administrators are accustomed. The platform (and what the platform manages) is not managed by using some commands, which then “do something” until they come to an end, be it success or failure.

Instead, Kubernetes holds objects in its internal databases and watches these for changes. Users, administrators, internal and external processes interact with these database objects. After the platform detects a change, it does its best to have the real-world status meet the wanted status of the objects in the database. For example, a command for deploying an application (or the IBM block storage CSI driver) might return immediately and report success, although the platform is still busy combining all of the required pieces.



### 3.1.2 Control plane

Regarding Kubernetes as the operating system of a cloud, the kernel of this operating system is Kubernetes' control plane. The core components can be found here, which let the platform manage itself. Also, founder here is the entry point to the platform.

The significant elements of the control plane are shown in Figure 3-1.

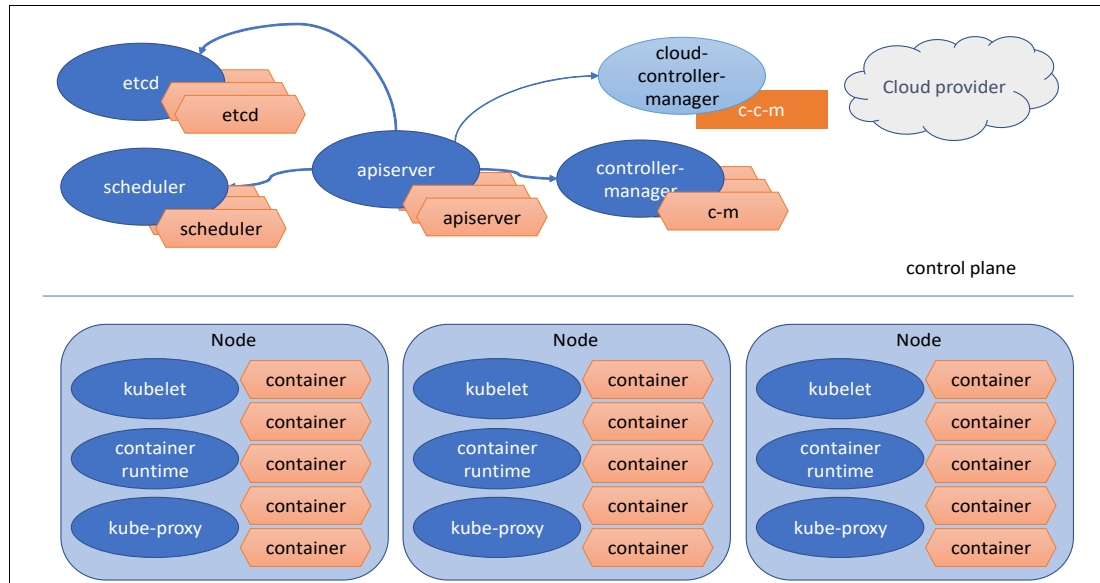


Figure 3-1 Kubernetes components overview

The control plane includes the following components:

► API server (apiserver)

This component forms the front end of the control plane and the central point for all communication within the control plane. It offers a RESTful web service that all components in the cluster can use. Any component that must access the platform also must interact with the API server. This component features GUIs that are included with OpenShift, or command-line clients (such as the `kubectl` command), or user applications that want to interact with Kubernetes objects.

► etcd

The distributed and reliable key-value store that forms the foundational Kubernetes object database is called *etcd*. Anything that must be stored is stored in the etcd cluster, which is at the core of the control plane. Protection of this database is vital for the cluster. If it is lost, the cluster is gone.

For each object the platform can manage, a representation of it exists in the etcd database. Ensure that etcd is backed up or protected by etcd snapshots. For more information about etcd backup, see [this web page](#).

The configuration information of the OpenShift Cluster is stored in this database.

► Scheduler

This component makes the ultimate decision about where to run specific workloads. It considers the requirements for a workload and the available machines with their characteristics and allocates the best fit.

- ▶ Controller-manager

This component can be seen as the workhorse in Kubernetes. It contains the built-in controllers, which are the processes that watch for changes on the database objects and perform the necessary action to have real-world status meet the wanted status in the objects. These built-in controllers include the following examples:

- Node controller: This controller acts on changes to the machines that are forming the cluster. It registers new machines that are joining, or starts recovery actions if it detects a machine failure.
- Workload controller: This class of controllers implements the workload controllers that are described in 3.1.5, “Workload controllers” on page 24.
- Endpoints controller: This controller is responsible for the connection between applications in the cluster and the network.

- ▶ The cloud-controller-manager

This component often is under the responsibility of a cloud provider. It forms the interface to the underlying cloud infrastructure that differs between different providers, such as Amazon Web Services or the IBM Cloud.

The control plane components are containerized applications, as shown in Figure 3-1 on page 21.

It is good practice to use separate infrastructure or compute nodes for the control plane for high availability, security, and reliability reasons. For example, in the IBM Cloud Kubernetes cluster offering, the control plane of customer clusters is not accessible for customers themselves. Instead, it is managed by IBM and running on separate infrastructure. Access is possible through the API server only.

### 3.1.3 Nodes

Nodes are Kubernetes’ abstraction of real computers, be it virtual or physical machines. As shown Figure 3-1 on page 21, a node consists of the characteristic entities that are running on the operating system of a machine:

- ▶ kubelet

This agent runs on every machine in the cluster. It interacts with the API server and the controllers in the control plane to fulfill its central tasks; that is, starting, stopping, monitoring, and deleting containerized workloads on the machine. This agent also includes setting up the running environment for such workloads.

- ▶ kube-proxy

An application often is useless if it cannot provide its service over the network, or use other services within or outside the cluster. The kube-proxy on each node handles establishing network connectivity for the workloads that are running on the node.

- ▶ Container runtime

The container runtime on a node can be considered as the motor of the platform. It loads the container images as the kubelet specifies them and runs them. Kubernetes can work with arbitrary run times if they fulfill the Kubernetes Container Runtime Interface (CRI); for example, Docker, CRI-O, containerd, or Kata containers.

**Note:** These components run under the operating system’s control on a machine. The kubelet and kube-proxy communicate with the platform through the apiserver, which in turn consists of a set of containers that are placed on the control plane nodes.

To build pools of nodes, the node objects in the Kubernetes database can be labeled (which also is possible for other objects). By using label selectors, specific sets of nodes can be specified, which, for example, can run a certain workload. Another possibility to characterize distinguished nodes is to add annotations.

### 3.1.4 pods

pods are the smallest possible workload entity in Kubernetes. A pod can consist of one or many containers. At first consideration, this proposition seems to be confusing because Kubernetes is a container orchestrator. However, a pod is the complete workload that is scheduled to a node for execution, as shown in Figure 3-2. Kubernetes ensures that all containers that belong to the same pod are also scheduled to the same node.

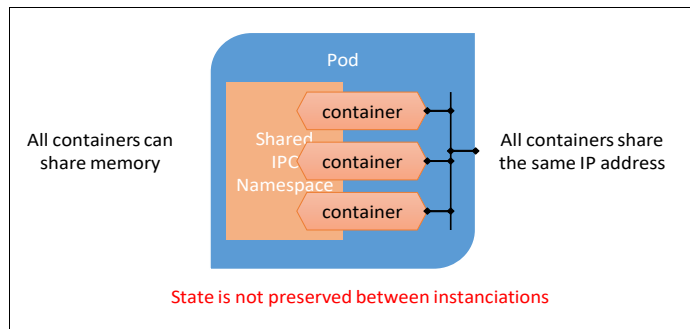


Figure 3-2 Kubernetes pod

Also, all containers in a pod share the Linux network- and Inter-Process-Communication (IPC) namespace. Therefore, all containers in a pod are bound to the same network interfaces and addresses, and they can communicate directly through IPC mechanisms, such as message queues, semaphores, or shared memory segments.

A pod can be configured so that its containers share process namespaces so that they can see themselves in the same process hierarchy. A running container resembles a process in the operating system. Its main program has process ID (PID 1) if looking at it from within the container. Also, processes that are in the container spawns are visible in the container as children of PID 1. However, if containers share the process namespace, they also see each other as processes with individual PIDs.

pods are volatile in a sense that they do not maintain state between instantiations. Whatever data a pod produces internally is lost and unrecoverable after it ends. Whenever a pod is scheduled and starting, it begins at “day zero” again. Persistent storage must be provided externally, and the CSI driver is one of the possibilities to provide it.

No start, stop, or pause commands are available for pods. They can be created (run) or deleted only. This fact reinforces the declarative approach in Kubernetes; that is, it is the database object “pod” that is created, and the platform alone does what is needed to have that wanted set of containers run somewhere in the environment. The “create pod” API call returns when the creation of the database object is queued, which does not mean that the wanted workload is running.

## Sidecars

The CSI design uses so-called *sidecar* containers, which are also relevant during the deployment of the IBM block storage CSI driver. (The name of the pattern stems from the metaphor of mounting a sidecar to a motorcycle; that is, the main application. In the context of the CSI driver architecture, we see this pattern applied.)

The sidecar pattern is a well-known design pattern in microservices development, and one of the simplest patterns that motivate the concept of pods. It assumes that a containerized main application provides the wanted basic service. Also, one or more “sidecar” containers exist that extend or improve the application container.

For example, imagine an application container that makes available an HTTP-based API. Assume that we want to extend the application with an HTTPS endpoint for secure communication. To do so, we add a sidecar container that handles the added protocol and forwards its incoming requests to the main application container.

The sidecar does not need to know about the specific information of the main application, and vice versa (the main application container does not need any modifications to handle HTTPS requests). Both of the containers are arranged in the same pod so they can directly communicate with each other without exposing any of their internal, potentially unsecured, communication.

### 3.1.5 Workload controllers

Creating and running single pods might seem enough for simple use cases, but the real power of the platform is found in its workload controllers. These controllers help describe specific characteristic workloads and let the platform enforce respective behavior.

One important aspect about the workload controllers is that they take complete ownership of the pods they start and delete. This behavior might confuse a Kubernetes novice. If an administrator deletes a pod that is controlled by a workload controller, the pod immediately is re-created according to the type of controller. This process can occur in parallel to the ending of the deleted pod.

However, turning around that principle helps fix erroring pods. It is often safe to delete or rewrite an erroring pod, or in the worst case, delete all pods that are managed by the same controller because they are re-created from their initial state.

In the following sections, we cover the basic controllers as they are used in the context of the CSI driver.

#### **ReplicaSet and deployment controller**

In this section, we described use cases for ReplicaSets and deployment controllers.

##### ***ReplicaSet***

The typical use case for a ReplicaSet (an object controlled by the ReplicaSet controller) is a workload in which multiple instances of the same pod must run in parallel. A ReplicaSet is used when the workload is characterized by the following conditions:

- ▶ The pod instances do not need to be distinguished, which means that they all provide the same service independently of each other.
- ▶ The pods do not maintain state.

The first condition means that no pods with a special role exist; for example, one distinguished leader and several secondary instances.

The second condition means that none of the instances must preserve information between restarts. the ReplicaSet workload is a number of pods where the count of pods can be increased or decreased without affecting the application's function or internal application status.

A typical use case for this type of workload is a media streaming service, which can be scaled up or down according to a request rate. The IBM block storage CSI operator is such a workload.

### ***Deployment controller***

The deployment controller's most important feature is to add a lifecycle concept to the ReplicaSet. It is for this reason that the use of deployments for the depicted use case is favored.

The deployment can be viewed as a workload controller that rolls out ReplicaSets. By doing so, it can roll out a new version of a pod and roll back a failing deployment. The deployment controller can implement different roll out strategies, such as scaling down the number of pods in an active ReplicaSet, while in parallel scaling up the number of pods in a ReplicaSet with a new pod version.

In our example, my-app is our application for which we have a container image my-app, which is tagged at version 1.3.3. Our application pod needs only to have this single container image run.

We want to scale our application to have two replicas active, and we do this in a deployment in which we also specify an update strategy of Rolling Upgrade.

After the my-app deployment object is created in the platform, the deployment controller detects no respective ReplicaSet exists and creates one, giving it a unique name with my-app- as a prefix.

The ReplicaSet controller then discovers no respective pods exist for that ReplicaSet, and create pods, again with unique names that are prefixed with the ReplicaSet name.

The scheduler then looks for nodes that can fulfill the respective resource requirements of the application. Eventually, the kubelet on the wanted nodes spin up the pods by enabling the container runtime start containers.

Figure 3-3 shows the final situation: two pods were created: one on node worker-0, the other on worker-3. Both pods run the same image in their respective containers.

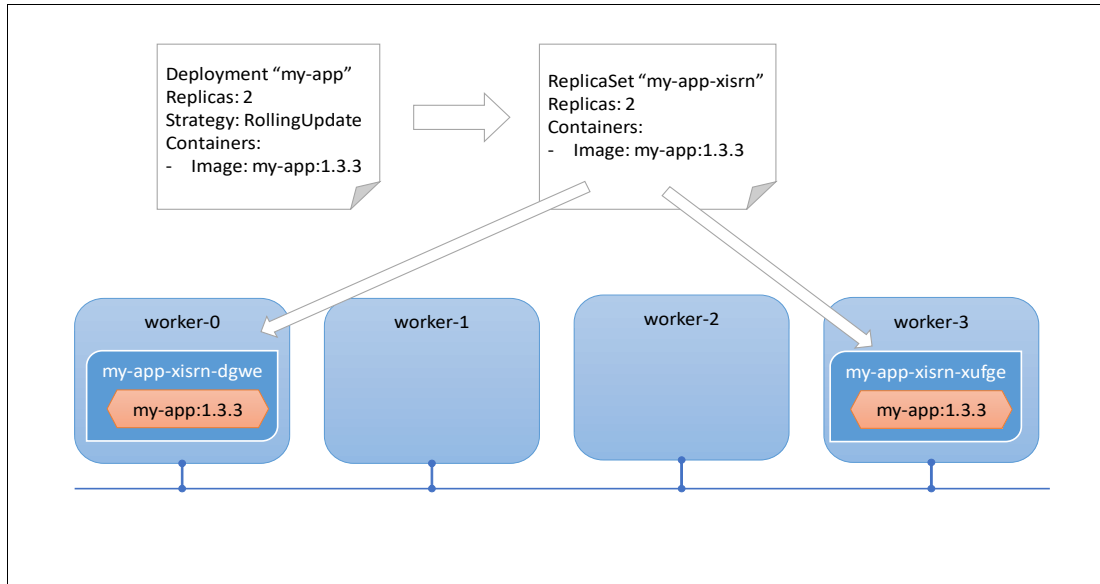


Figure 3-3 After Deployment is created

Assume that we built a new release of our application so a new `my-app` image is available, which is tagged as version 1.3.4. To deploy the new version, it is sufficient to update the image reference in the `my-app` deployment object for our new release.

The deployment controller then detects the change and starts the upgrade according to the Rolling Update strategy. It creates a ReplicaSet with the new image reference, and scales it up while it scales down the old one, which ends the pods.

Figure 3-4 shows an intermediate situation, where the ReplicaSet with version 1.3.3 that is scaled down to one replica, and the new ReplicaSet with version 1.3.4 that is scaled up to one replica.

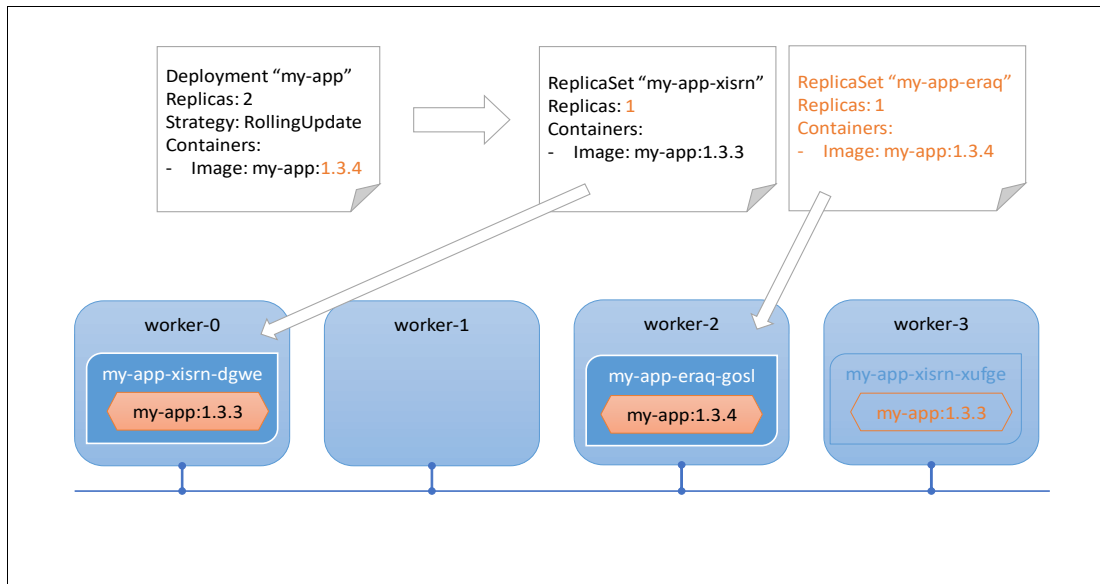


Figure 3-4 Updated Deployment in progress

The decision about where the new pods are placed is made exclusively by the scheduler that is based on the current use of the cluster. Therefore, it might happen that one of our application pods are on a different worker than before (as shown in Figure 3-4 on page 26), or the scheduler decides for the same worker again. The latter case is illustrated in Figure 3-5, which shows the final state after updating our image version in the deployment. The new pod, with our release 1.3.4, is worker-0, where we also released a 1.3.3 image running earlier.

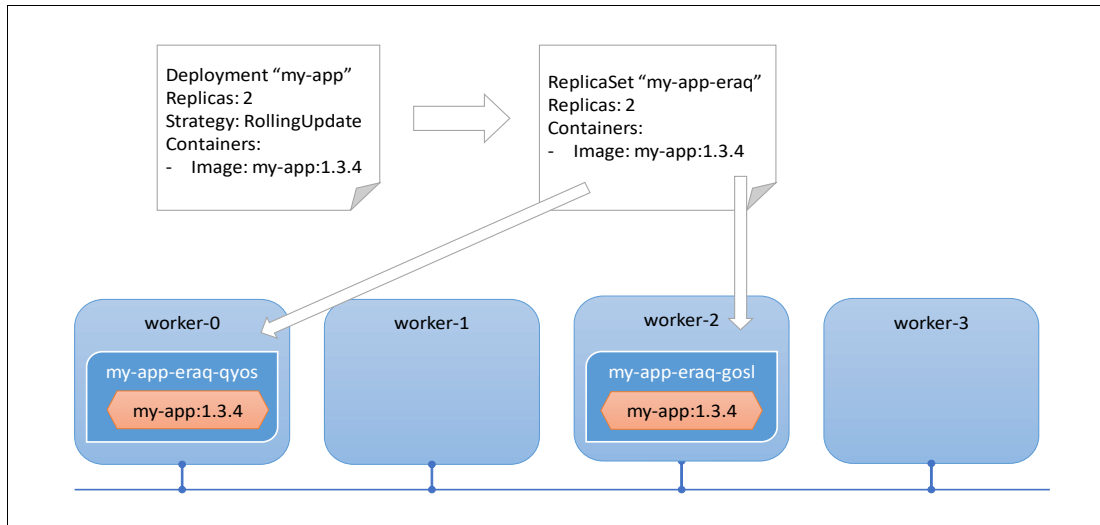


Figure 3-5 New release of “my-app” is deployed

## DaemonSet

A DaemonSet workload is characterized as a pod, which runs on all (or a distinguished set of) nodes. Therefore, whenever a new node enters the cluster, a pod is placed there. When a node exits from the cluster, the respective pod is not restarted elsewhere, as was the case with other workload controllers (see Figure 3-6 and Figure 3-7 on page 28).

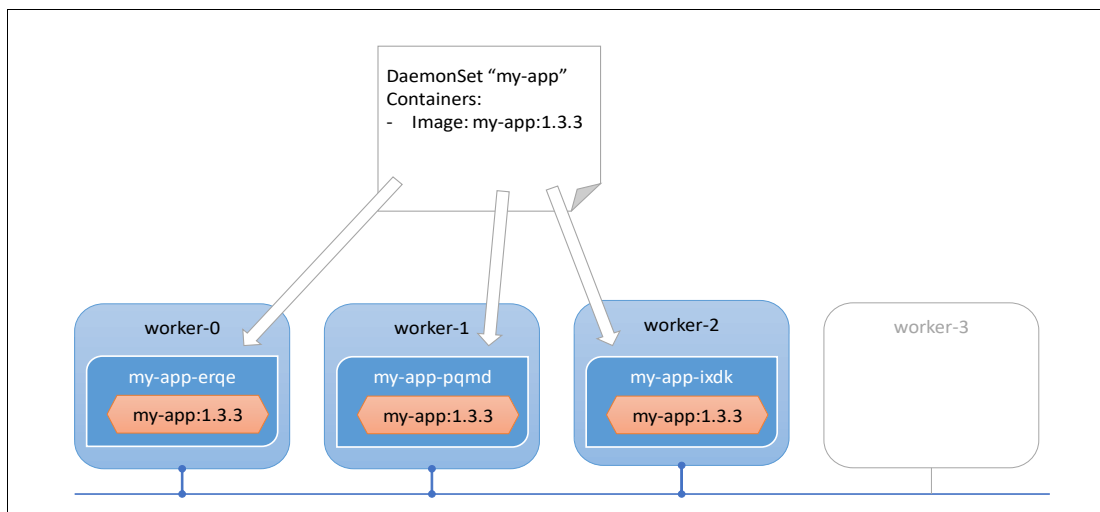


Figure 3-6 DaemonSet before worker-3 joins the cluster

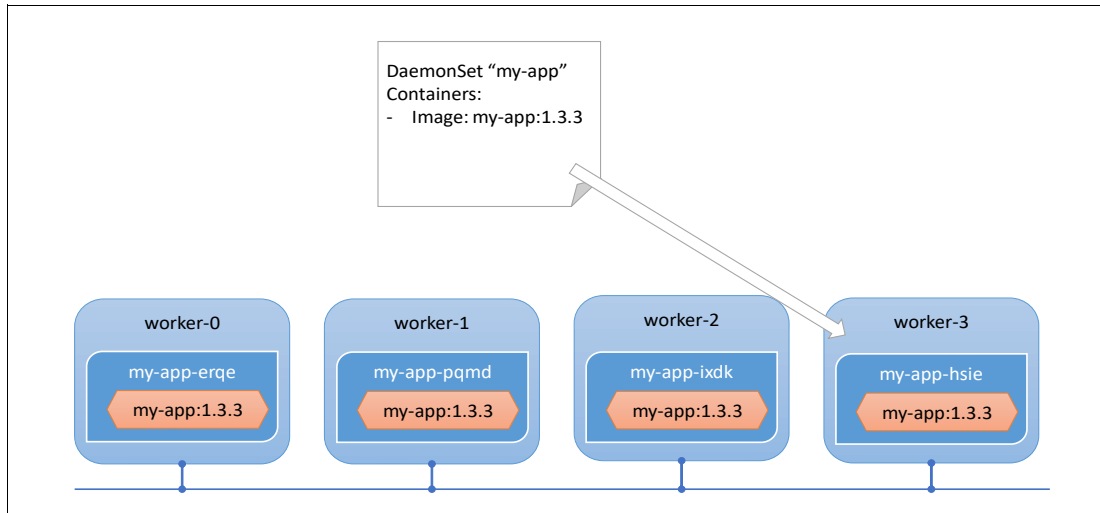


Figure 3-7 DaemonSet after worker-3 has joined the cluster

A typical use case for a DaemonSet is a monitoring application that monitors activity on all nodes that are in the cluster. Logging collections on nodes is also a popular pattern for a DaemonSet. We see the node component of the IBM block storage CSI driver being deployed in a DaemonSet.

## StatefulSet

A StatefulSet can be a complementing concept to the ReplicaSet. It manages a specific number of pods that are running the same code. However, a StatefulSet features the following set of conditions:

- ▶ The pod instances must be identifiable
- ▶ The distinguished pods maintain individual state across instantiations

*Identifiable pods* means that the pods, although running the same application, can have different roles. For example, a leader might be among them. The etcd cluster that is described in “Control plane” is an example for that use case.

If a pod from an etcd cluster is rescheduled to another node, its database also must go there and it must preserve the state it had on the former node.

Distinguishing the pods might also be an application requirement. While in a DaemonSet or a ReplicaSet, the different pod instances do not know each other, and the StatefulSet allows the different instances to identify and communicate with each other.

Our example that is shown in Figure 3-8 on page 29 shows a deployed StatefulSet with three pods. Our specification also states that each of the distinct pod instances requires 10 GB of persistent storage to keep its state. We use symbolic names for the provisioned storage parts in our figures; in reality, they look a bit more cryptic. The pod names appear as shown.

We assume that the pods in the StatefulSet were deployed to the workers nodes worker-0, worker-2, and worker-3. On worker-1, other pods might be running that are out of the scope of our StatefulSet.

We also assume that node worker-2 quits the cluster for whatever reason, be it an operating system upgrade, hardware defect, or network issue. The platform notices the loss of that node and deletes the pod objects that were scheduled to the node.



Then, the StatefulSet controller can discover that one of the wanted replicas is missing, and it creates the pod again.

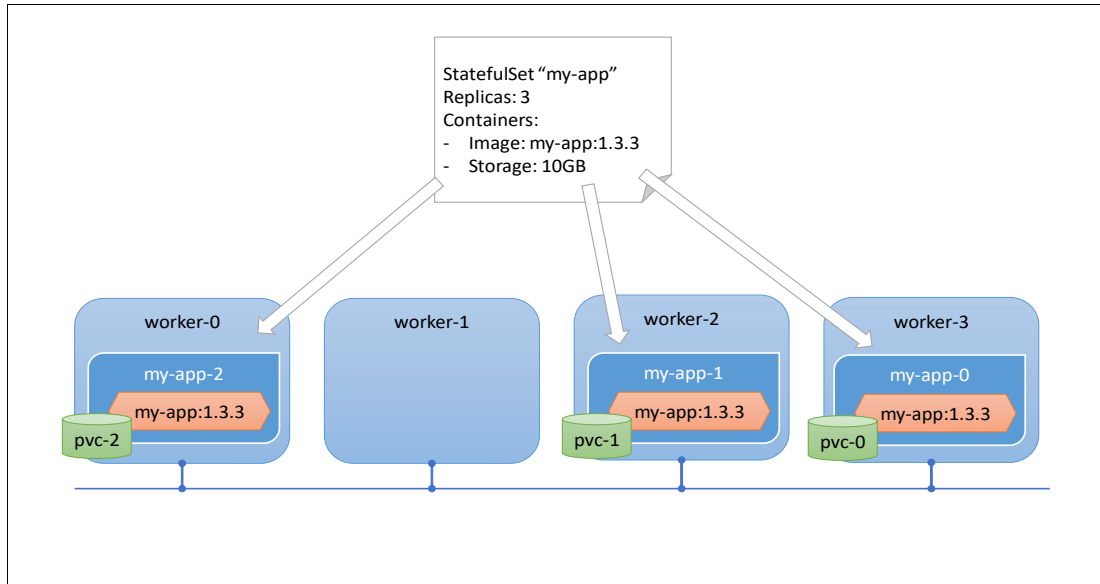


Figure 3-8 StatefulSet with three pod instances deployed to three nodes

The newly created pod then proceeds with the normal scheduling process in Kubernetes, which can result in placing the pod onto node worker-1, as shown in Figure 3-9.

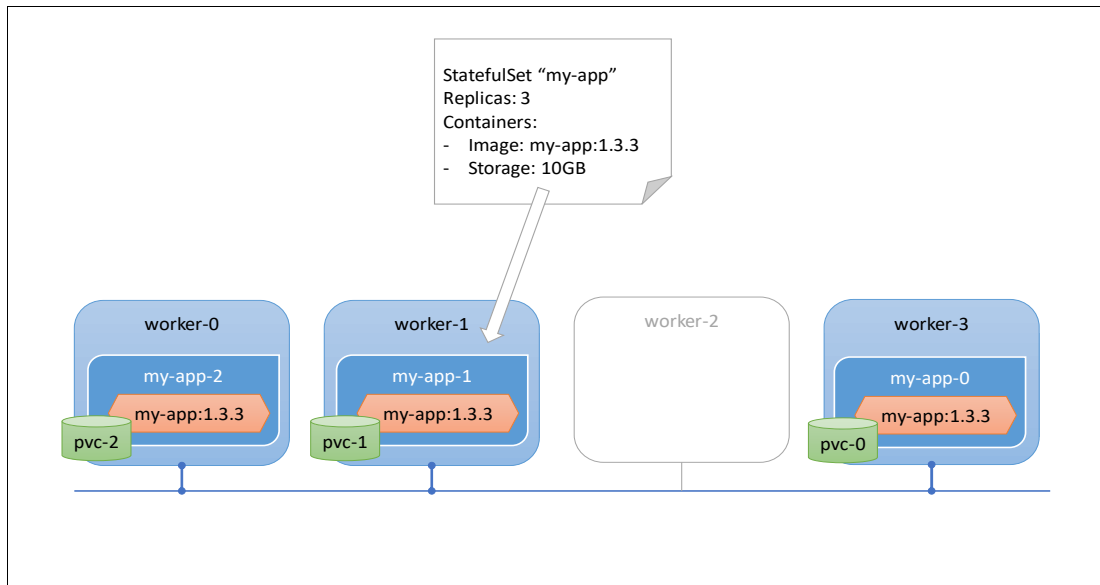


Figure 3-9 Pod and storage rescheduled in a StatefulSet after worker-2 quits the clusters

In contrast to the ReplicaSet or DaemonSet, the StatefulSet controller manages the following special circumstances:

- ▶ The exact pod instance that was lost (here, my-app-1) must be rescheduled.
- ▶ The data that the pod uses must “move with the pod” to the node where the pod is not running.

When discussing the CSI driver in “Kubernetes Container Storage Interface” on page 36, we return the respective challenges for the implementation of a persistent storage concept in a highly dynamic environment.

The users of persistent storage that is provisioned with the CSI driver are typical candidates for StatefulSet workloads.

### 3.1.6 Persistent storage

As shown in section “pods” on page 23, pods do not keep their state when they are restarted because the platform decides to reschedule them, or an administrator or user deletes and starts them again.

This principle does not allow designing stateful applications that require to maintain state, or to share persistent data among multiple pods. Kubernetes solves this problem with its concept of persistent storage.

Several objects and controllers come together to fulfill the need for storage. The architecture might seem confusingly complicated at first, but it is useful. Some evolution occurred over the Kubernetes releases. We now have a robust design with clear separation of duties as the implementation of the CSI driver demonstrates.

The concepts around persistent storage are shown in Figure 3-10. The storage-related aspects are shown on the left side of the figure; the application, or pod-related entities, are shown on the right side.

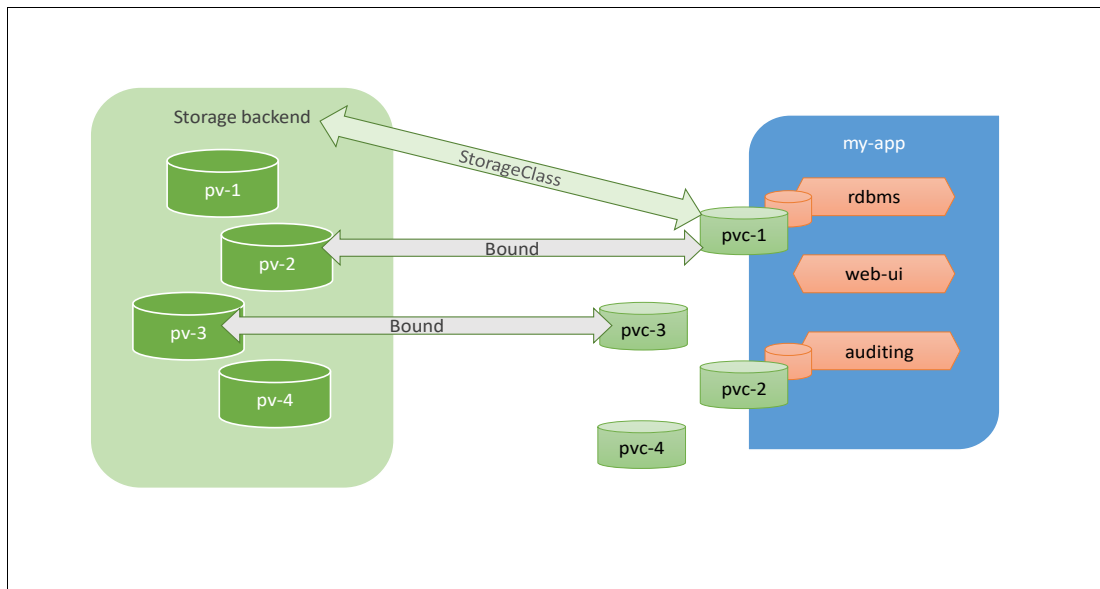


Figure 3-10 Persistent storage overview

### Volumes

At pod level, volumes are the objects providing persistent storage. The containers within the pod use volumes by mounting them into their file system hierarchy. These volumes are shown as the smaller orange disks that are shown in Figure 3-10. They are close to the containers. However, although these volumes are the object of an application’s requirement, Kubernetes adds some wrapping concepts around these mountable volumes as described next.

## PersistentVolumeClaims

Kubernetes manages volumes with persistent storage through PersistentVolumeClaim (PVC) objects. This kind of object describes the characteristics of a specific piece of storage from an application's perspective. PVCs offer an abstraction of storage that is usable for pods. They are created together with the respective pods that want to use the respective storage, and their characteristics are determined by the application:

- ▶ Size of the storage: How much space does the pod need on the volume?
- ▶ Access mode of the storage: Does it need to be read or written by one or many pods at the same time?
- ▶ Volume mode: Does this volume contain a file system or is it a block device?

In Figure 3-10 on page 30, we see the light green disk symbols close to the pod. Although PVC describes storage from a pod's perspective, PVCs also can exist independently of any pod because the storage is persistent. The data in that storage exists independently of the fact that it is actively used. This situation of having PVCs without pods referring to them is normal.

Consider the following points:

- ▶ A PVC can be referred from an arbitrary number of different pods. Therefore, zero or more instances, either active at the same point in time for ROX or RWX access, or different pods can refer to the same PVC at different points in time for RWO access. For example, the same PVC can be used by one pod for data generation, while another pod can use the generated data later for analysis.
- ▶ A pod can refer to a specific PVC that does not exist. This aspect is confusing about Kubernetes: A pod referring to a non-existent PVC can successfully be created without issues. The respective CLI call immediately returns successfully after the pod object exists in the Kubernetes database. However, the pod cannot start until the PVC exists and is bound to a PersistentVolume, what we describe next.

## PersistentVolumes

A PersistentVolume (PV) object (in our Figure 3-10 on page 30 the darker green disk symbols on the left) describes some real, available piece of persistent storage. In contrast to the PVC, the PV describes how a storage provider, or a storage back-end, looks at persistent storage.

At this point, often the question arises as why two different concepts exist that essentially describe the same thing. The answer can be found in the declarative paradigm of the platform.

A PV declares something a storage provider can offer, while a PVC declares something a storage consumer wants. We can leave it to the platform that takes the responsibility to bring the things together. This functional separation allows an application developer to define the PVCs for the application independently of the details of the underlying available storage.

However, a storage administrator does not need to “mount”, or “assign” storage specifically to any application. Both roles (application developer and storage administrator) can work fully decoupled in their respective domains, and let the platform put the pieces together.

How does the platform match PVs and PVCs? PV and PVC must provide the same access mode, the PVC's size requirement must be less or equal to the PV provided size, and the volume mode must be compatible. If the platform finds a matching PV for a newly created PVC, or vice versa, a PV appears that satisfies the demands of a PVC, and both PV and PVC are bound to each other.

The double-ended arrows that are shown in Figure 3-10 on page 30 between pv and pvc objects illustrate this concept. Bound and unbound PVs and PVCs are visible in Figure 3-10 on page 30, which is a natural state that we explain next.

## Dynamic volume provisioning

In the simplest case, a storage administrator creates some consumable pieces of storage on the storage back-end systems and makes them available for the platform by creating respective PV definitions. Applications can then claim or allocate these pieces through PVCs. However, this approach has the following flaws:

- ▶ Our storage administrator must create the storage pieces on the back-end system and then translate the “real” objects into their Kubernetes PV representation, which is a possibly error prone process.
- ▶ The storage pieces our administrator provides must fit the possible demands in the PVCs. They should fit the largest possible PVC size; however, small PVCs produce much wasted storage space if only larger PVs are available.
- ▶ Although sufficient storage is available on the back-end system, PVCs can remain unsatisfied \because the platform ran out of PVs of the needed size.

To overcome these drawbacks, Kubernetes introduced the concept of dynamic volume provisioning. With dynamic provisioning, the platform \ takes the role of our storage administrator. However, instead of creating PVs in advance for a demand that might not exist, the platform watches the creation of PVCs and follows up by creating PVs.

## StorageClass

When we look at dynamic volume provisioning, we observe that some automation must take over the storage administrator’s role. Such an automation is limited to distinct storage back-ends. This issue leads to the concept of a storage provisioner. A provisioner is some code that can allocate storage on a specific backend. Kubernetes provides an abstraction of such a provisioner in StorageClasses.

StorageClasses put together a reference to the provisioner plug-in (the code), and parameters that the plug-in needs to perform its operations. The storage administrator’s role is no longer to preallocate storage and create PVs. Instead, the role is to define the available StorageClasses and to preallocate some capacity; for example, in a dedicated storage pool that is on the storage.

It is left to the application developer to specify the StorageClass in the PVC for the application pods. The light green double-sided arrow that is shown in Figure 3-10 on page 30 connects pvc-1 with the storage back-end and shows how a StorageClass works for dynamic provisioning.

Now that we have dynamic volume provisioning and StorageClasses, binding PVCs and PVs can be done by completing the following steps:

1. A user deploys an application that specifies pods and PVCs. The PVCs reference a StorageClass that the storage administrator provided and advertised to the users.
2. The platform detects new PVCs that include no matching PVs in their StorageClass and starts the StorageClass’ provisioner.
3. The provisioner allocates a matching piece of storage on the back-end and creates a respective PV for it.
4. The platform now finds matching PVs for the PVCs and binds them.

This simple model does not cover the aspects of capacity management. However, this issue is out of scope here. The common solution is to monitor the storage back-end systems for their capacity and alert the storage administrators if capacity is running short. Also, the storage provisioning is out of scope here.

For more information, see Chapter 4, “OpenShift and Container Storage Interface deployment” on page 45.

## Volume snapshots

In addition to creating and managing persistent storage for stateful pods, Kubernetes also allows snapshots to be taken of volumes that are in use by a pod. This data, which represents a state at a specific point, can then be used as initial content for new volumes. A typical use case for this scenario can be database backups; for example, when preparing a bigger modification, such as a schema migration. Having a snapshot can serve the development and verification of the migration, and allows roll back if failures occur.

Similar to PVCs and PVs, Kubernetes is separating the concepts into an application-driven view and a storage administration view. The respective objects are:

- ▶ **VolumeSnapshots**

These are the analogy to PVCs. Applications can make use of VolumeSnapshots, without the need to understand the underlying storage backend’s details.

- ▶ **VolumeSnapshotContent**

This is the analogy to PVs. The storage backend “knows” how to map VolumeSnapshotContent objects to the respective available options on the backend.

As with the PVC/PV case, VolumeSnapshots and VolumeSnapshotContent belong to each other, although the “binding” term moved to a “ready to use” property. After a VolumeSnapshot includes a corresponding VolumeSnapshotContent, it can be used for the creation of new PVCs that refers to the VolumeSnapshot object as data source. Therefore, its bound PV’s initial content consists of the respective VolumeSnapshotContent.

VolumeSnapshots are available for Cloud Storage Interface drivers and the IBM block storage CSI driver only. It is an optional capability, which the IBM CSI driver supports starting with version 1.3 on OCP 4.4 or Kubernetes 1.17.

Similar to PVs and PVCs, a storage administrator can pre-provision VolumeSnapshotContent objects by taking snapshot of volumes on the storage system. A VolumeSnapshotContent object can be created dynamically by specifying the PVC from which to take it.

## 3.1.7 Application configuration

Although we covered the aspects of the stateless nature of pods and how persistent data is being made available for them, how to effectively bring configuration information to pods is important.

Because we expect one or more pods to implement a meaningful application from more or less generally usable container images, we do not want to have to place the required configuration information into these images.

Think of such things as, for example, parameters that are different for production, test, or development releases of an application, or access tokens or other external references to back-end systems. Not only is it inefficient to include this information into images, it is overkill to use persistent storage for these information bits.

Kubernetes offers two handy concepts for configuration information. These concepts are described next.

## ConfigMaps

With ConfigMaps, Kubernetes offers an effective way to pass configuration information to applications. These are simple key or value stores within the Kubernetes objects database that keep information together. Pods then can refer to ConfigMaps for setting up individual attributes. ConfigMaps are useful for the following purposes:

- ▶ Setting environment variables for containers in pods.
- ▶ Providing entry call parameters for containers.
- ▶ Providing configuration file content that can be used by containers.

Containers can also use the Kubernetes API programmatically to access data in a ConfigMap.

## Secrets

Although ConfigMap content is a good fit for any clear text information, it should not be used for information that is confidential. It is for this reason that Kubernetes offers Secrets.

Small pieces of confidential information (for example, usernames, login passwords, or API tokens) can be stored in Secrets. This information can be stored encrypted. Often, users do not want to have such information in clear text in a pod specification or part of container images.

Instead, a container can mount a Secret to some point in its file system hierarchy and then access the data through a file in this file system. A second possibility is to pass data from Secrets to container environment variables. Eventually, a special use case for Secrets is image pull secrets, which are the access credentials the kubelet passes to the container runtime so it can pull container images from protected image registries that require authentication.

### 3.1.8 Extension points in Kubernetes: The operator pattern

One of Kubernetes' strengths is its extensibility. By design, it offers various extension points which, according to the respective use case, allow flexible extensions to the platform. Such extensions can be more commands for the CLI, specific authentication or authentication for API requests, or more resources the platform can manage.

We concentrate on one specific way of extending the platform because it is the foundation of the value that is added with OpenShift, and the principle that is behind the CSI driver in general and the IBM block storage CSI driver in particular.

This method is the operator pattern. Its building blocks are CustomResources, their definitions and controllers, and the embracing concept of a Kubernetes operator.

#### Custom resources

Kubernetes allows to extend its API with arbitrary resources. These resources are collections of user-defined objects in the Kubernetes database. Such Custom Resources (CR) can, as with the built-in pod resource (for example) be managed by the platform.

Custom resource objects can be added to or deleted from collections on the CLI, or programmatically from within the environment. The only issue left to the CR designer is to make its structure known to Kubernetes so it can suitably handle these objects.

A common way to do this is by using a CustomResourceDefinition (CRD). CRDs are resources that are managed by Kubernetes. To add own objects to Kubernetes, a user writes a CRD and hands it over to Kubernetes. From that moment on, the platform also can manage these objects.

Just having a collection of new objects in a CR is useful, but often the intention of new resources is to make some use of them. As with the built-in pod, resources feature the semantic of letting a set of containers run somewhere in the cluster. We expect CRs to also include some semantic aspect. It is here where resource controllers come into play.

Figure 3-11 show how controllers fit into Kubernetes' programming model. A controller declares its interest in the changes (add, update, and delete) of resources (regardless if built-in or CRs) with informers. It also provides suitable code that acts on these events. These actions can then by themselves change other resources, which is done through calls to the apiserver. These changes to resources occur in the etcd database and create events, which in turn helps their controllers to take action, and so on.

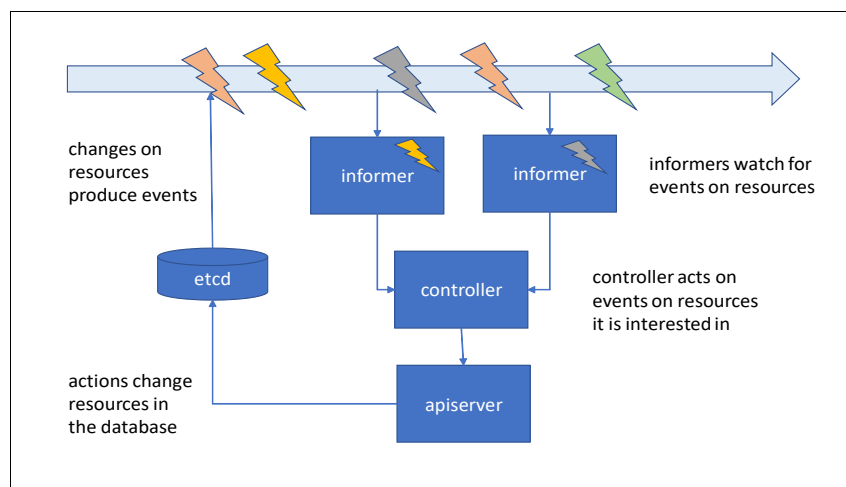


Figure 3-11 Basic principle of a resource controller

This illustration can also explain, why creating only a pod with a valid pod specification through the CLI immediately returns successfully, although at that point it is unclear if this pod can find a node in the cluster on which to run.

Several other controllers are available in the platform that watch for the creation of pod objects and take actions. However, these actions might fail or timeout. For example, the scheduler might not find a node with sufficient memory available. Because the scheduler cannot know whether this is a temporary problem, it postpones the decision, which leaves the pod in a "Pending" state.

## Operators

We described the internal mechanics of the platform in a high-level overview, and discussed how Kubernetes helps us define CRs, and how controllers for CRs work. Putting the pieces together brings us to the operator pattern. An operator is a CRD and a controller for the defined CRs. However, the question arises as to why an operator is needed.

The driving force is automation: Imagine a scenario in which an application developer creates an application and now wants to deploy it in the cluster. The application can consist of several back-end pods that must cooperate on some shared storage, and some other pods that implement a user front-end.

One possible way of deploying the application can be to manually create the respective pod specifications and then manually start the CLI to create them individually. Alternatively, a deployment for the front end, and a StatefulSet for the backend can be a more effective way. However, what if another version of the application is needed in a test version, or must be deployed into a development environment?

It is here that the operator comes into play. Instead of manually deploying the application with its varying parameters and sizing in different versions, a CRD can be set up for the application as a whole. This CRD describes the mandatory and optional deployment parameters for an application. Creating a respective controller, which performs all the steps our developer completes manually, enables the application to deploy automatically.

This pattern of a CRD together with a controller implementing the operational behavior for that type of resource is what Kubernetes calls an operator. The basic idea is that an operator performs the actions that a human operator does in an automated fashion. The OpenShift platform uses the operator pattern extensively to provide value regarding the operations of Kubernetes clusters.

## 3.2 Kubernetes Container Storage Interface

With Kubernetes 1.13, the CSI officially became a part of Kubernetes. Its primary goal is to decouple the implementation of container storage back-ends from the main Kubernetes code base.

Before CSI, the support for many storage back-end plug-ins required integration with the Kubernetes. That is, even a bug fix in a plug-in's code required rebuilds of the entire platform. These plug-ins are often called "in-tree" because they are part of the entire Kubernetes code tree.

CSI aims at overcoming this limitation. By the design of an interface between storage back-end providers and the Kubernetes platform, development can advance in a decoupled manner.

New storage back-ends can be implemented and tested independently from the Kubernetes platform, so can the CSI pieces on Kubernetes side. The CSI specification was developed with arbitrary container orchestrators in mind. Therefore, it is general enough to be used by other platforms, such as Cloud Foundry.

Together with the CSI specification in terms of the functions that a storage driver must implement come some valuable assets. One is a deployment proposal, which is a description of how the deployment of a CSI driver for a specific storage backend can be structured. The other is a set of containers that can be readily used to support simple integration with the Kubernetes platform, notably the Kubernetes side of CSI. This combination of assets underpins the intention to provide a general container orchestrator solution.

For more information, see the following GitHub resources:

- ▶ [Design proposal on CSI](#)
- ▶ [CSI specification](#)



## 3.2.1 Volume lifecycle

This section gives a short introduction to creating volumes and volume-to-node attachment.

### Creating PVCs

One of the significant advantages of the CSI driver is its ability to dynamically provision volumes. A storage administrator no longer needs to pre-provisioned pieces of storage on the back-end system that might not fit the demand and do not leave much unused capacity behind. Next, we describe how the CSI driver implements this dynamic provisioning.

The starting point is that a PVC is created from the deployment of an application into the cluster. For dynamic provisioning (in addition to its size, volume and access mode), we know a StorageClass is required (in our example, it refers to a CSI driver). The StorageClass also contains the storage pool information and the Secrets. After such a PVC is created in the cluster, the following workflow is triggered:

1. Kubernetes' persistent volume controller detects the creation of a PVC object and because it is not bound to a PV, the controller first tries to find a matching unbound PV.
2. Not having found a PV lets the controller check if the PVC provided a StorageClass. If so, it delegates PV provisioning to the respective provisioner plug-in that is in the StorageClass.
3. The provisioner allocates a suitable piece of storage on the back-end and creates a PV object from it in the database. As a side task, it also annotates the PVC with the PV's name.
4. The persistent volume controller detects that a PV was created and because the new PV includes a reference to the requesting claim in it, the controller can bind PV and PVC.

When the requested PVC is in a "Bound" state, the respective volume can be accessed by a requesting pod. Once again, all of these operations (PVC creation, PV creation, and pod creation) occur asynchronously and potentially independently from each other.

### Volume: Node attachment

Volume attachment, which means making a storage volume accessible on a node so that pods can use them, is a complex task. The attach/detach controller is the Kubernetes instance that is managing these aspects. It watches events on many resources: pods, nodes, PVCs, PVs, VolumeAttachments, and on CSI objects.

Pod updates are the most prominent triggering events. The scheduler that is placing pods onto nodes is a number one source for these events. The attach and detach controller is a central instance and must know about all the defined volumes, nodes, and pods and their relationships in the cluster. It interfaces with all storage plug-ins in-tree and external storage providers where CSI is considered one of them.

What happens if a pod is placed on a node and its volumes must be attached to the node? Consider the following points:

- ▶ Pod updates or additions with a changed node attribute let the attach/detach controller internally create a tuple (pod, volume, or node) for each volume that the pod uses. The controller maintains a private "desiredWorld" structure. It also holds a private "actualWorld" structure, where it records the assignments.
- ▶ Periodically, a reconciler loop in the controller scans the desiredWorld and the actualWorld to transform statuses into wanted statuses.

First, it triggers detachments for all (pod, volume, and node) bindings that are found in actualWorld but not in desiredWorld. Then, it triggers attachments for all (pod, volume, and node) bindings in desiredWorld but not in actualWorld.

- ▶ The respective attachment or detachment operations are implemented in the storage plug-ins for the volumes in question. These operations run in asynchronous threads or processes. Therefore, the attachment or detachment controller also must manage timing conditions or multiple repetitions of the same request. For example, it does not trigger attachment for a volume if an attach operation is in progress.
- ▶ Looking at attachment of a volume, the storage plug-in creates a VolumeAttachment object and waits for the attachment plug-in to mark the volume as attached to the node.
- ▶ After it is attached, the attachment or detachment controller no longer sees a difference between the desiredWorld and actualWorld status. The volume is now ready to be used by the pod on the targeted node.

Similarly, if a node quits the cluster or a pod ends, the desiredWorld status is updated by removing the respective references for the node or pod. Instead of attaching volumes, the storage plug-in runs detach operations.

### 3.2.2 CSI driver deployment

In this section, we review the main CSI driver building blocks, CSI controller, and CSI node (see Figure 3-12).

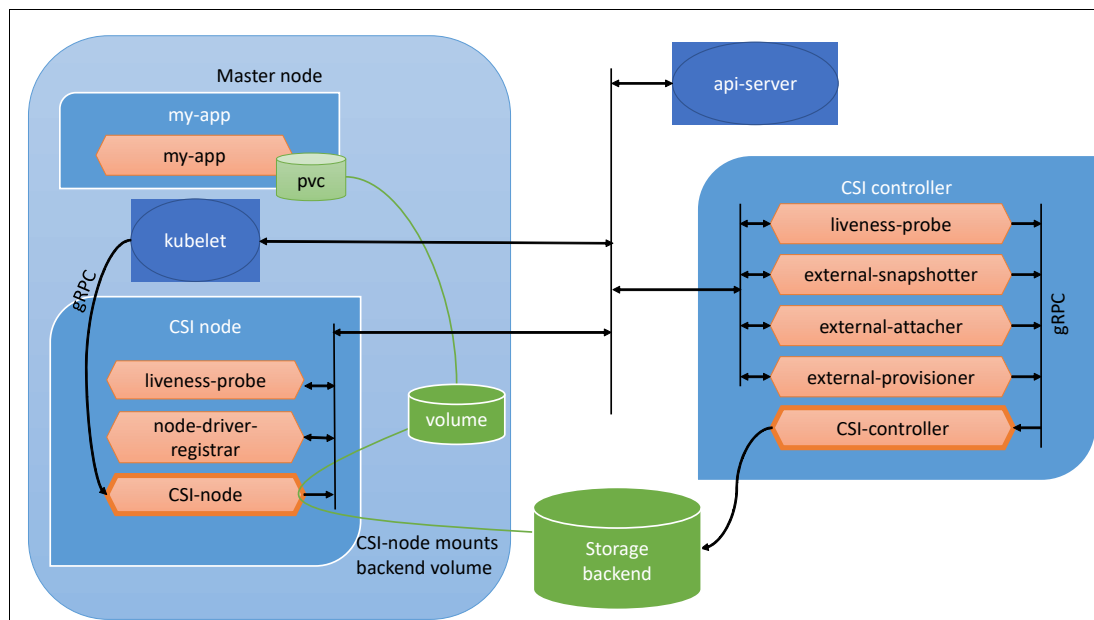


Figure 3-12 CSI components

#### CSI controller

The CSI controller’s main responsibility is to provision and deprovision storage on the back-end systems. The CSI controller pod contains some sidecars, which are responsible for the communication with the Kubernetes API server, while they communicate with the controller container through gRPC calls that are laid down in CSI specification, namely the Controller service.

Next, we describe the major calls to the controller, and leave it up to the reader to consult the documentation for more information.

### ***CreateVolume and DeleteVolume***

These calls are often made by the external-provisioner. As we saw in “Volume lifecycle” on page 37 after a PVC that is referring to a StorageClass for the CSI driver is created, the Kubernetes in-tree volume controller looks for a suitable PV, which generally does not exist yet.

However, if a PVC’s StorageClass refers to a CSI-managed volume, the respective plug-in contacts the external-provisioner, which calls `CreateVolume()`. The CSI Controller then cuts an appropriate piece of storage from the back-end, and with successful return from `CreateVolume()`, the respective PV object in the platform is created by the external-provisioner.

This process eventually allows the Kubernetes’ volume controller to bind the new PV with the PVC when it runs through its reconciliation loop. A `DeleteVolume()` is available to have the CSI controller delete the provisioned storage and the external-provisioner delete the respective PV.

At this point, we have not yet discussed the origin of volume content. In addition to provisioning a pure volume without any content, two other options are available: If the backend and driver support it, a volume can be created from another, existing volume, which ends up in creating a clone, and, a volume can be created from a VolumeSnapshot.

For more information about CSI volume cloning and CSI volume snapshots, see this [Red Hat documentation web page](#).

### ***CreateSnapshot and DeleteSnapshot***

The calls that deal with VolumeSnapshots do not differ much from the respective `CreateVolume` or `DeleteVolume` calls. They start respective procedures on the storage backend to create or remove snapshots from volumes on the storage backend.

### ***ControllerPublishVolume and ControllerUnpublishVolume***

After the Kubernetes scheduler decides to place a pod onto a specific node, the pod’s volumes also must be made accessible by that compute node. The need to attach a volume is expressed in VolumeAttachment objects that are watched for changes by the external-attacher sidecar in the CSI Controller pod. If a VolumeAttachment appears or its node reference changes, it starts the respective calls against the CSI controller.

### **CSI node**

The CSI node containers often run on each compute node (or at least those nodes where the respective storage backend can be used), which makes a perfect case for deploying them in a Kubernetes DaemonSet.

A CSI nodes main task is to bring the volumes from the storage backend and the respective storage driver of the node operating system together. This process often includes mounting the storage volume to some mount point in the node operating system. After it is assured that the mount point is available, and the correct volume is mounted there, and the container runtime can finally mount the volume into the consuming container’s file system hierarchy. However, this last step is out of the scope of this publication.

In Figure 3-12 on page 38, we see a node-driver-registrar sidecar that is collocated with the CSI node container in the CSI node pod. This sidecar allows the CSI node container to register with the kubelet on the node by providing the reference to the endpoint where its gRPC functions can be called.

After the connection is established through a UNIX Domain Socket, node-driver-registrar generally no longer plays a role. However, the CSI node container does not communicate with the kubelet other than by responding to the gRPC calls.

### ***NodeStageVolume and UnstageVolume***

For specific types of storage, it might be necessary to perform preparatory steps before some back-end volume can be made accessible on a node. The optional NodeStageVolume calls enable this accessibility. For example, in the IBM CSI node case, the NodeStageVolume call ensures that a volume that is intended to be mounted in NodePublishVolume is seen as a multipath device by the operating system.

### ***NodePublishVolume and UnpublishVolume***

The NodePublishVolume is the final step in provisioning a volume from a CSI-managed storage backend to a node from where standard Kubernetes mechanisms can present it to a pod's containers. These mechanisms and all other CSI Node procedures are directly called from the kubelet. The kubelet is the instance on a compute node that is responsible for taking pod specifications and bringing the respective containers to life on the container runtime.

This task can be accomplished successfully only if all of the requested volumes for a pod are available on the node so that they can be mounted by the containers. The CSI Controller's ControllerPublishVolume procedure is doing what must be done to make a volume available on a node from the storage system's perspective. Now, with the NodePublishVolume, the loop is closing; that is, the node accesses the dynamically provisioned and attached storage.

## **CSI identity services**

The CSI architecture is built on a plug-in pattern. Stubs are available in the platform that are complemented with concrete implementations, the CSI controller, and the CSI node. However, the platform needs information about the implemented features of these components. Therefore, the CSI components must offer some identity services. Some are common for both controller and node, and some are needed for node or controller only.

### ***Common identity services***

These services must be implemented by the controller and the node component. They provide information for the platform on the respective component. Their task is to identify the component in the platform (GetpluginInfo), and its capabilities (GetpluginCapabilities). In addition, a Probe function gives a success response after it is called so that the platform knows that the component is ready to communicate.

### ***Component identity services***

To determine the specific capabilities of the components, they provide a ControllerGetCapabilities, which is a NodeGetCapabilities call that informs about special features the component supports. For example, the ability to provision volumes from a snapshot is one of the possible controller capabilities.

An important information call is available on the node component: NodeGetInfo. This call is used to create a unique node identification for the CSI environment. As the IBM block storage CSI driver shows, this call encodes the node name and identifies the storage backend and how it can be reached. This information is important for the controller that must provision storage for a specific node on a specific backend.

## 3.3 IBM block storage CSI driver

Thus far, we covered the common CSI driver concept and how it integrates with Kubernetes as the container orchestrator of our choice. We also know of the CSI that governs the communication between the Kubernetes-provided sidecars and the storage vendor-provided CSI Controller and CSI Node component.

In this section, we review the IBM block storage CSI driver's specific parts. We see that it not only comprises the expected controller and node component, it also includes an operator. Because the IBM block storage CSI driver is an open source effort, see [this web page](#) for more information about the code.

### 3.3.1 IBM CSI Operator

In the “Extension points in Kubernetes: The operator pattern” on page 34, we explained the operator pattern. Here, we describe the CSI Operator that plays a central role in IBM block storage CSI driver deployments. For more information about its source code, see [this web page](#).

#### IBMBlockCSI resource

As described in “Extension points in Kubernetes: The operator pattern” on page 34, the operator pattern generally brings together custom resources and a controller. For the IBM CSI Operator, this kind of resource is called *IBMBlockCSI*. If we consider a single instance of a *IBMBlockCSI*, we get a basic impression of what the CSI Operator is doing. An *IBMBlockCSI* object contains the following major parts:

- ▶ Controller specification

This specification details two important aspects of the IBM block storage CSI driver's controller component: the container image that runs the controller code, and a node selection expression that further specifies the type of nodes onto which the controller pod can be scheduled.

- ▶ Node specification

Similar to the controller specification, the node specification details the image to use and a node selection criteria. However, although node specification is about the CSI node component, the node selector in this specification helps find the suitable Kubernetes nodes where to run the CSI node.

- ▶ Sidecar specifications

The CSI controller and node use the community-provided sidecars. An *IBMBlockCSI* object also specifies which container images are used for these sidecars.

From these specifications, we can derive the controller's tasks: It watches the *IBMBlockCSI* resources and creates the respective workload controllers for the deployment of a *IBMBlockCSI* driver installation. Because the image specifications (their repository paths and image tags) can be declared in the *IBMBlockCSI* resource, upgrading the driver can also be done in a declarative style; that is, an administrator then modifies the image references in the resource, which the CSI Operator picks up and triggers new deployments of whatever is needed for the declared version.

The operator is deployed to the cluster through a Deployment workload controller. This process can be done through the OpenShift GUI or on the command line.

After the operator is established in the cluster, an administrator then creates a suitable IBMBlockCSI object that specifies the wanted controller and node references. The operator triggers all necessary steps to deploy the IBM block storage CSI driver with all its components into the environment.

Final steps for an administrator include defining StorageClasses that further specify how to access the storage backend, from which storage pool to provision appropriate volumes, and so on. The volumes can then be dynamically created on the back-end by creating PVCs for applications in the defined StorageClasses.

### 3.3.2 IBM CSI controller

The IBM CSI Controller can provision storage on three different IBM block storage provider families: DS8000, Spectrum Virtualize products (such as IBM SAN Volume Controller), and the A9000/R family of products. Although these different back-ends require different adoption layers, the gRPC calls on the “CSI-side” are the same, and only the StorageClass of a PVC determine which back-end to contact.

As of this writing, the IBM Controller component is implemented in Python. It consists of the set of gRPC calls that are required by the CSI Controller and CSI Identity service, but then uses different open source Python modules to communicate with the back-end storage systems (the pyds8k module for the IBM DS8000 family, the pysvc module for IBM Storwize® and IBM FlashSystem, and the pyxcli module for IBM XIV/IBM FlashSystem A9000/R back-ends).

A thin intermediate layer forms the connection between CSI functions and the back-end modules. For more information about these modules, see [this web page](#).

### 3.3.3 IBM CSI node

Most of the functions the IBM CSI Node implements is described in “CSI node” on page 39. The IBM CSI Node driver supports the NodePublishVolume and NodeUnpublishVolume calls. After they started, the NodePublishVolume() function checks if the access for the wanted volume is valid, if the device node can be accessed, and if it appears as a multipath device in the compute node operating system.

Another task also is accomplished by the IBM CSI node on its first invocation on a node. It generates a unique identifier for the node to make itself visible and addressable for the IBM CSI controller. This identifier is called *nodeid*, and the Kubernetes Node object is annotated with it, as shown in Example 3-1.

*Example 3-1 Nodeid example*

---

```
$ oc describe node worker-0 | egrep -a -A1 -i nodeid
csi.volume.kubernetes.io/nodeid:
{"block.csi.ibm.com":"worker-0;iqn.1994-05.com.redhat:7c9b3e4fee9;c05076fff280987c:c05076fff2809e40"}
```

---

The **oc** command is the OpenShift command line client, and its **describe** operation gives a summary of the node information in the etcd database. By using the **egrep** command, the *nodeid* annotation can be filtered out.

The IBM block storage CSI driver provisions “file system” and “block” volumes, as described in “Creating a PersistentVolumeClaim” on page 80. The name of the product refers to the block storage nature of the storage back-end systems, not the driver’s capabilities.

### 3.3.4 CSI driver storage back-end communication

The instances that are communicating with the storage back-ends are distinct Python modules for each of the three supported storage families. As with all the other components of the IBM block storage CSI driver, these modules are open source.


For more information, see the following resources:

- ▶ [pyds8k \(IBM DS8000\)](#)
- ▶ [pyxcli \(IBM XIV, IBM FlashSystem A9000/R\)](#)
- ▶ [pysvc \(IBM Spectrum Virtualize\)](#)

These client modules can be used standalone in Python programs. This ability allows users to examine the interaction with a storage backend, aside from the Kubernetes environment for troubleshooting.







# OpenShift and Container Storage Interface deployment

This chapter describes a proven process for an OpenShift setup and the deployment of the IBM Container Storage Interface (CSI). The following environments are discussed:

- ▶ OpenShift Version 4.3 on IBM Power
- ▶ OpenShift Version 4.4 on IBM Z
- ▶ OpenShift Version 4.5 on x86

This chapter shows step-by-step CSI implementation examples that use the OpenShift Command-line Interface for IBM Power and IBM Z. It includes the following topics:

- ▶ 4.1, “IBM block storage Container Storage Interface” on page 46
- ▶ 4.2, “Installing Red Hat OpenShift Container Platform in an IBM Power Systems PowerVM environment” on page 52
- ▶ 4.3, “CSI deployment on IBM Power by using the command-line interface” on page 72
- ▶ 4.4, “CSI and OpenShift on IBM Z” on page 87
- ▶ 4.5, “CSI deployment on Z by using the command-line interface” on page 91
- ▶ 4.6, “Installing the CSI driver by using the OpenShift web console on x86” on page 95
- ▶ 4.7, “Updating the CSI driver” on page 110

## 4.1 IBM block storage Container Storage Interface

The examples in this chapter are based on the IBM block storage CSI driver version 1.3.0. For more information, see this [IBM Documentation web page](#).

### 4.1.1 CSI configuration overview

CSI consists of two objects within Kubernetes: the CSI operator and the CSI driver. An *operator* is a Kubernetes software extension, which uses custom resources that are outside of core Kubernetes. It also interfaces with the API Server and acts as a custom Controller. Although CSI consists of two elements, the Operator downloads the driver concurrently.

The CSI driver can be installed in OpenShift by using two methods (the choice depends on the server platform): the OpenShift Web Console or the OpenShift CLI. After the CSI driver is installed and running, relevant storage classes, constructs, and secrets must be created to use CSI. This section gives describes the needed configuration files.

#### Creating an array secret

Within Kubernetes, sensitive information, such as passwords, OAuth tokens, and ssh keys are stored within secrets. These secrets can then be used in a safe manner rather than placing information into a manifest or yaml file, pod definition, or container image.

For more information about designing Kubernetes secrets, see Chapter 3, “Container Storage Interface architectural overview” on page 19 and this [GitHub web page](#).

A storage system secret must be created to store the storage credentials (username and password) and the address of the storage system that is used by the CSI driver, as shown in Example 4-1.

*Example 4-1 Array secret file*

---

```
kind: Secret
apiVersion : v1
metadata:
  name: itso-secret
  namespace: itsons
type: Opaque
stringData:
  management_address: 9.9.9.9 # Array management address
  username: itso-user # Array username
data:
  password: Fjw95ndf0nf= # base64 array password
```

---

**Important:** If the storage service account password is changed, ensure that the passwords in the corresponding secrets are changed, especially when LDAP is used on the storage system.

## Creating CSI storage classes

A StorageClass manifest must be created to define the storage system pool name, reference the storage Secret, and set parameters such as space efficiency and file system type (see Example 4-2).

*Example 4-2 CSI StorageClass.yaml*

---

```
kind: StorageClass
apiVersion : storage.k8s.io/v1
metadata:
  name: itso-storageclass
provisioner: block.csi.ibm.com
parameters:
  SpaceEfficiency: thin # Optional. Values standard\thin. Default is standard.
  pool: itso-storageclass

  csi.storage.k8s.io/provisioner-secret-name: itso-secret
  csi.storage.k8s.io/provisioner-secret-namespace : itsons
  csi.storage.k8s.io/controller-public-secret-name: itso-secret
  csi.storage.k8s.io/controller-public-secret-namespace: itsons

# csi.storage.k8s.io/fstype : xfs # Optional. Values ext4\xfs. Default is ext4
  volume_name_prefix: itsoPVC # Optional. DS8000 maxlength = 5, all others = 20.
```

---

Consider the following points:

- ▶ The **pool** value should be the name of the pool on the storage system.  
On IBM DS8000 systems, **pool** should be the pool ID of the extent pool and not the pool name.
- ▶ The **csi.storage.k8s.io/fstype** value is optional. Values can be **ext4** or **xfs**, with **ext4** being the default
- ▶ The **volume\_name\_prefix** value is optional:
  - For IBM DS8000, the maximum length is five characters.
  - For all other IBM Storage systems, the maximum length is 20 characters.

Defining several StorageClasses is common where users need PersistentVolumes with differing properties, such as performance, standard or thin provisioning, or block or file type volumes for different types of containers. An OpenShift storage administrator must offer various PersistentVolumes that differ than only in size without having to allowing users to know how these volumes are provisioned.

The StorageClass defines only the provisioner, provisioning type (thin or standard), file system type, and an optional volume prefix.

## Creating a PersistentVolumeClaim

A PersistentVolumeClaim (PVC) yaml file must be created to bind PVs that the CSI driver creates. The PVC defines the required parameters for creating a volume on the storage subsystem by using the information that is held in the StorageClass. Combined, they have all of the information the storage subsystem needs to create and map the volume.

Example 4-3 shows a basic PVC that provisions a 10 GB volume by using the `itso-storageclass` that is shown in Example 4-2 on page 47.

*Example 4-3 PVC 10 GB raw block volume yaml*

---

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: itso-pvc-raw-block
spec:
  volumeMode: Block
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: itso-storageclass
```

---

A PVC can also be created from a volume snapshot. An example of a volume snapshot is shown in Example 4-4 where `datasource: name: itso-snapshot` is the referenced `VolumeSnapshot` Kubernetes resource that refers to the PV that is used as the source volume for the snapshot.

*Example 4-4 PVC created by using volume itso-snapshot as a source*

---

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: its-pvc-from-snap
spec:
  volumeMode: Block
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: itso-storageclass
  datasource:
    name: itso-snapshot
    kind: VolumeSnapshot
    apiGroup: snapshot.storage.k8s.io
```

---

## Creating a StatefulSet

A *StatefulSet* is a workload API object that must be defined to manage stateful applications.

The *StatefulSet* manages the deployment of a set of pods and any required scaling. It also provides assurance about the ordering and uniqueness of the pods in the *StatefulSet*.

The *StatefulSet* manages pods that are based on an identical container spec. Compared to a *Deployment* (see [this web page](#)), a *StatefulSet* maintains a persistent ID for each of their pods. The pods are created from the same spec, but are not interchangeable. Each pod includes a persistent ID that remains, even if the pod must be rescheduled.

Because of the nature of this persistence, persistent volumes are key to maintaining the integrity of the pods that are within the StatefulSet. Although individual pods that are in a StatefulSet are prone to failure, the persistent ID makes it easy to match volumes to the new rescheduled pods that replace failed ones.

Stateful sets are used when considering container applications that require one or more of the following characteristics:

- ▶ Stable, persistent storage
- ▶ Stable, unique network identifiers
- ▶ Ordered, automatic rolling updates
- ▶ Ordered, graceful deployment
- ▶ Ordered scaling

**Note:** Deleting or scaling down a StatefulSet to zero does not delete the associated volumes. This arrangement is to ensure data integrity, which is a general request other than purging data that is on ephemeral volumes.

StatefulSets can include volumes with file systems, raw block volumes, or a combination of both. When defining the StatefulSet (see Example 4-5), ensure that you define volumes according to the PVC type.

*Example 4-5 Creating a StatefulSet by using raw block storage*

---

```
kind: StatefulSet
apiVersion: app/v1
metadata:
  name: itso-statefulset-raw-block
spec:
  selector:
    matchLabels:
      app: itso-statefulset
  serviceName: itso-statefulset
  replicas: 1
  template:
    metadata:
      labels:
        app: itso-statefulset
    spec:
      containers:
      - name: itso-container
        image: registry.access.redhat.com/ubi8/ubi:latest
        command: [ "/bin/sh", "-c", "--" ]
        args: [ :;"while true; do sleep 30; done;" ]
        volumeDevices:
          - name: itso-volume-raw-block
            devicePath: "/dev/block"
      volumes:
      - name: itso-volume-raw-block
        persistentVolumeClaim:
          claimName: itso-pvc-raw-block
```

---

Example 4-6 shows a StatefulSet that is similar to Example 4-5 on page 49, but includes mounting a file system volume that is called `itso-volume-file-system` onto mount point `/dat`, which was created by using the PVC `itso-pvc-file-system`.

*Example 4-6 StatefulSet that uses raw block storage and mounting file system volume*

---

```
kind: StatefulSet
apiVersion: app/v1
metadata:
  name: itso-statefulset-raw-block
spec:
  selector:
    matchLabels:
      app: itso-statefulset
  serviceName: itso-statefulset
  replicas: 1
  template:
    metadata:
      labels:
        app: itso-statefulset
    spec:
      containers:
      - name: itso-container
        image: registry.access.redhat.com/ubi8/ubi:latest
        command: [ "/bin/sh", "-c", "--" ]
        args: [ : "while true; do sleep 30; done;" ]
        volumeMounts:
        - name: itso-volume-file-system
          mountPath: "/data"
        volumeDevices:
        - name: itso-volume-raw-block
          devicePath: "/dev/block"
      volumes:
      - name: itso-volume-file-system
        persistentVolumeClaim : itso-pvc-file-system
          claimName: itso-pvc-file-system
      - name: itso-volume-raw-block
        persistentVolumeClaim:
          claimName: itso-pvc-raw-block
```

---

## Creating volume snapshots

Since the introduction of OpenShift 4.4 and Kubernetes 1.17, volume snapshots are available. By using the CSI driver, platforms can run IBM FlashCopy® commands on the connected storage subsystems.

For more information about volume snapshot support for a specific OCP version, see [this web page](#).

To enable creating and deleting snapshots on the storage subsystem, a `VolumeSnapShotClass` yaml file must be created to define the behavior in a policy. Example 4-7 on page 51 shows an example that uses the storage subsystem `array` secret and secret namespace (as shown in Example 4-1 on page 46).

*Example 4-7 VolumeSnapshotClass block storage snapshot*

---

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshotClass
metadata:
  name: itso-snapshotclass
driver : block.csi.ibm.com
deletionPolicy: Delete
parameters:
  csi.storage.k8s.io/snapshotter-secret-name: itso-secret
  csi.storage.k8s.io/snapshotter-secret-namespace: itsons
  snapshot_name_prefix: itsoSnap # Optional.
```

---

**Note:** The VolumeSnapshotClass deletionPolicy property determines what happens to the VolumeSnapshotContent when the VolumeSnapshot bound object is deleted. This property can be Delete or Retain and this field is a required.

If Retain is specified, the underlying snapshot and VolumeSnapshotContent remain when the bound VolumeSnapshot object is deleted.

After the policy is defined in the VolumeSnapshotClass yaml file, a snapshot can be performed by defining a VolumeSnapshot yaml file, as shown in Example 4-8.

*Example 4-8 Creating a volume snapshot*

---

```
apiVersion: snapshot.storage.k8s.io/v1beta1
kind: VolumeSnapshot
metadata:
  name: itso-snapshot
spec:
  volumeSnapshotClassName: itso-snapshotclass
  source:
    persistentVolumeclaimName: itso-pvc-raw-block
```

---

## 4.2 Installing Red Hat OpenShift Container Platform in an IBM Power Systems PowerVM environment

This section describes installing Red Hat OpenShift Container Platform (OCP) in an IBM Power Systems PowerVM® environment.

At the time of this writing, the latest IBM CSI block-storage driver version 1.3 supports Red Hat OCP 4.3 on the IBM Power Systems platform, which is the versions that we used for our installation.

Figure 4-1 shows our IBM PowerVM LPAR server lab environment that we used for our Red Hat OCP installation.

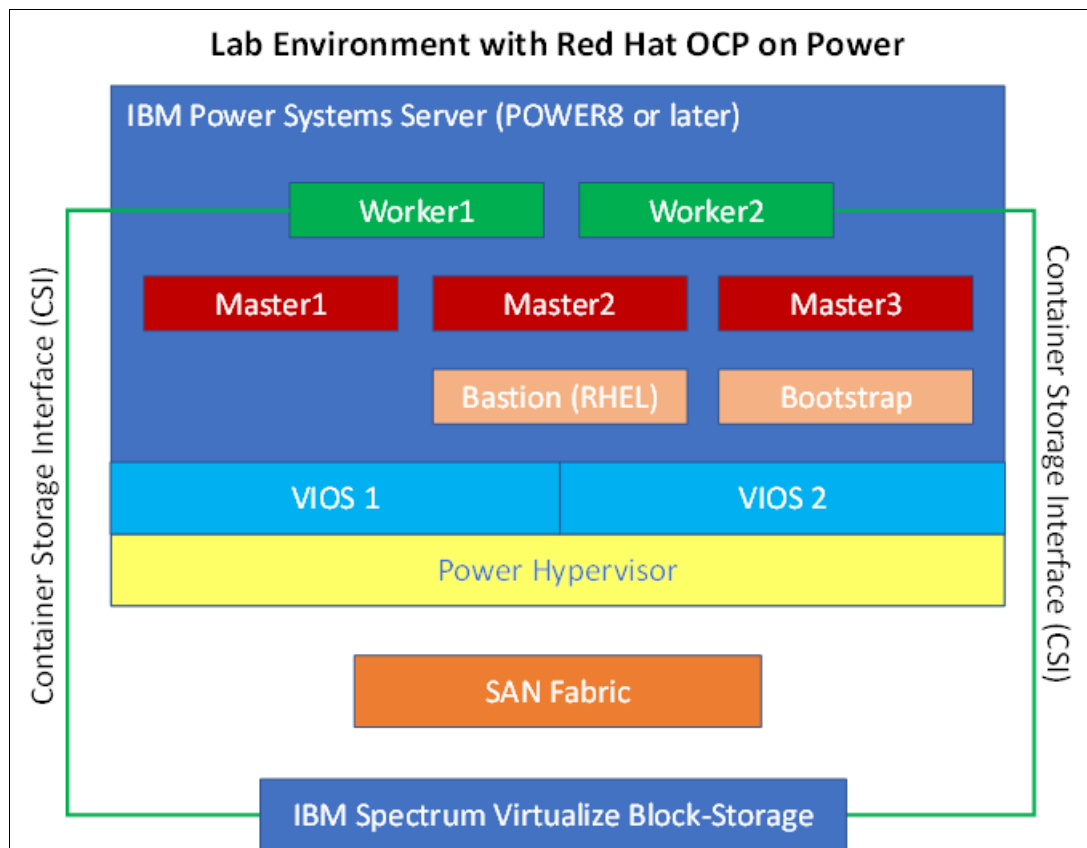


Figure 4-1 IBM Power Systems server lab environment

Red Hat OpenShift supports a PowerVM LPAR environment that was created by the PowerVM administrator where each LPAR is treated as a static resource as though it were a bare metal resource. For more information, see this [Red Hat web page](#) (log in required).

The persistent storage was provided to our Red Hat OpenShift worker nodes from an attached IBM Spectrum Virtualize-based storage system by way of the IBM block storage CSI driver.

We chose a high-performance Fibre Channel (FC) storage N\_port ID virtualization (NPIV) attachment for our worker nodes as IBM Virtual I/O Server (VIOS) client partitions. The IBM block storage CSI driver also supports iSCSI attachment when physical Ethernet cards or virtual Ethernet ports are used that are backed by SR-IOV dedicated VFs or virtualized by VIOS using vNIC technology in the worker nodes.



As a test environment, we used only a single physical server that hosted all OpenShift cluster nodes. For a production environment from an availability perspective, separate physical servers were used for hosting the redundant cluster nodes.

## 4.2.1 Red Hat OCP 4.3 on Power installation overview

The Red Hat OCP on Power installation comprises the following major steps, which we describe in the subsequent sections:

- ▶ Requirements planning
- ▶ Preparation of a RHEL bastion node
- ▶ Downloading the installation files
- ▶ Creating an installation configuration file
- ▶ Network services configuration (DNS HTTP Server, load balancer, and HTTP Server)
- ▶ Preparing the image files for booting the cluster nodes
- ▶ Creating the Kubernetes manifest and ignition files
- ▶ Creating the RHCOS cluster nodes by using the bootstrap process
- ▶ Completing and verifying a successful Red Hat OpenShift cluster installation

### Requirements planning

Red Hat OCP on Power requires a user-provisioned infrastructure (UPI) because a Red Hat OCP installer-provisioned infrastructure (IPI) is not supported on the IBM Power Systems platform as of this writing.

According to the minimum installation requirements, we created the following IBM PowerVM Virtual I/O Server (VIOS) client partitions on our IBM Power Systems server (ppc64le with POWER8® or later):

- ▶ 1x bastion node (installed with RHEL 8.2):
  - 1x CPU
  - 4 GB RAM
  - 200 GB boot volume
- ▶ 1x (temporary) bootstrap node:
  - 1x CPU
  - 16 GB RAM
  - 200 GB (minimum required: 120 GB) boot volume
- ▶ 3x master nodes (OpenShift cluster control plane):
  - 1x CPU
  - 16 GB RAM
  - 200 GB (minimum required: 120 GB) boot volume
- ▶ 2x worker nodes (OpenShift cluster compute nodes):
  - 1x CPU
  - 8 GB RAM
  - 200 GB (minimum required: 120 GB) boot volume

Consider the following points regarding the installation requirements:

- ▶ The worker nodes were configured with the minimum of 8 GB RAM. We later noticed a high memory usage that was driven by the Prometheus monitoring service, even with no user workload running. Therefore, we suggest configuring the worker nodes with at least 16 GB RAM.
- ▶ Red Hat CoreOS (RHCOS) is the required operating system for Red Hat OCP on Power for the master and the compute (worker) nodes.

- ▶ RHCOS as an immutable container host is self-managed in its configuration. This point pertains to the container orchestrator (cri-o), its authorized registries, and to over-the-air updates by the Red Hat OCP cluster machine configuration operator.
- ▶ IBM Power Systems dynamic LPAR (DLPAR) operations or storage multipathing is not supported on the OCP 4.3 RHCOS cluster nodes. For more information, see [this Red Hat web page](#).

In addition to the hardware requirements network services, such as a DNS server, a HTTP Server and network load balancer also are required for a Red Hat OCP cluster installation, which we also describe in the following sections.

For more information, see the official Red Hat OCP on Power installation documentation at [this web page](#).

## Preparing an RHEL bastion node

As a bastion node to be used for installing the Red Hat OCP cluster from and optionally for its later management by way of the OpenShift Client (oc), we prepared a Red Hat Enterprise Linux (RHEL) 8.2 partition on our IBM Power Systems server.

**Note:** Red Hat registration and subscription of the bastion node is not required; however, a later registration for OCP cluster subscription is needed after the expiration of a 60-day trial license.

In addition to the basic RHEL 8.2 installation on our bastion node, we installed utilities, such as GNU `tar`, `vim` (to allow for syntax highlighting in YAML configuration files), `wget` (for downloading installation files), and the Apache HTTP Server to serve the RAW image and ignition files to create the OCP cluster nodes (bootstrap, masters, and workers).

The Red Hat OCP installation files often are downloaded from this [Red Hat OpenShift Cluster Manager web page](#) (a registered Red Hat account is required to log in). This web page offers only the latest OCP version for download.

Because of the IBM CSI block-storage driver release compatibility with Red Hat OCP on Power, we used a prior version of Red Hat OCP on Power that we downloaded from [this web page](#).

We used the Red Hat OpenShift Cluster Manager website that is shown in Figure 4-2 on page 55 for only downloading the required pull-secret for the installation. This secret is associated with the user's Red Hat account and enables access to Red Hat container registries, such as Quay.io.

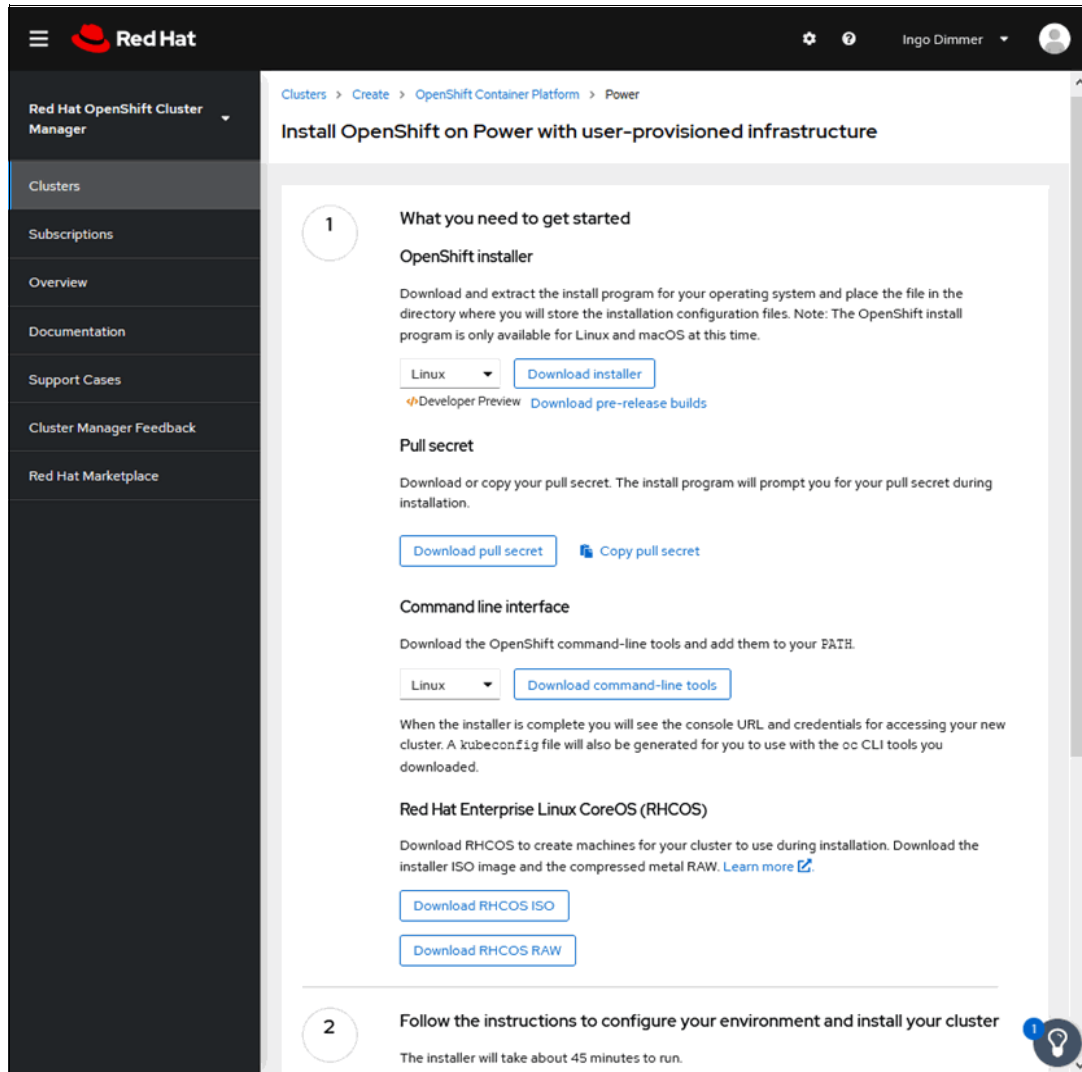


Figure 4-2 Red Hat OpenShift Cluster Manager website, reduced output for better readability

## Downloading Red Hat OpenShift installation files

We prepared the OCP on Power installation by following step 1 that is shown in Figure 4-2 by downloading the required OCP installation files (installer, OpenShift client [oc], and RHEL CoreOS [RHCOS] ISO and RAW image files) to our bastion node, as shown in Example 4-9.

### Example 4-9 Downloading the installation files

```
[root@i7PFE7 OCP_install]# wget --no-check-certificate
https://mirror.openshift.com/pub/openshift-v4/ppc64le/clients/ocp/latest-4.3/openshift-install-linux.tar.gz
[root@i7PFE7 OCP_install]# wget --no-check-certificate
https://mirror.openshift.com/pub/openshift-v4/ppc64le/clients/ocp/latest-4.3/openshift-client-linux.tar.gz
[root@i7PFE7 OCP_install]# wget --no-check-certificate
https://mirror.openshift.com/pub/openshift-v4/ppc64le/dependencies/rhcos/4.3/latest/rhcos-installer.ppc64le
.iso
[root@i7PFE7 OCP_install]# wget --no-check-certificate
https://mirror.openshift.com/pub/openshift-v4/ppc64le/dependencies/rhcos/4.3/latest/rhcos-metal.ppc64le.raw
.gz
```

From the Red Hat OpenShift Cluster Manager web page (see Figure 4-2 on page 55), we download the pull-secret text file to our local PC and transfer it by using Secure Copy to the bastion node.

By using the `tar xvf <filename>` command, we extract the OpenShift installation program and move it to a directory in the `/usr/local/bin` path.

We also extract the OpenShift client (`oc`, and `kubectl`) and move it to a location in our `/usr/local/bin` path.

By using the `oc version` command, we perform a quick check for the use of the OpenShift client:

```
[root@i7PFE7 OCP_install]# oc version
Client Version: 4.3.40
```

### **Creating an SSH key pair**

We generate an SSH public/private key to be used for SSH access by using the core user to the master nodes for debugging and disaster recovery purposes, as shown in Example 4-10.

#### *Example 4-10 SSH key pair*

---

```
[root@i7PFE7 ~]# ssh-keygen -t rsa -b 4096 -N '' -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Created directory '/root/.ssh'.
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
```

---

The generated SSH public key also must be provided to the OpenShift installation program.

### **Creating an installation configuration file**

We copy the sample `install-config.yaml` file that is provided by Red Hat on the [OCP installation web page](#) into our previously created installation directory (`/OCP_install`) on the bastion node that is holding the unique assets for each OpenShift cluster installation.

The customized `install-config.yaml` file for our specific environment includes the downloaded pull-secret and ssh-key that were copied as text into the file. The changes are highlighted in blue in Example 4-11.

#### *Example 4-11 Configuration example, example is truncated for better readability*

---

```
[root@i7PFE7 OCP_install]# cat install-config.yaml
apiVersion: v1
baseDomain: sle.kelsterbach.de.ibm.com
compute:
- hyperthreading: Enabled
  name: worker
  replicas: 0
controlPlane:
  hyperthreading: Enabled
  name: master
  replicas: 3
metadata:
  name: ocp-ats
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
```

```

hostPrefix: 23
networkType: OpenShiftSDN
serviceNetwork:
- 172.30.0.0/16
platform:
  none: {}
files: false
pullSecret: '{"auths":{"cloud.openshift.com":{"auth": ...}}'
sshKey: 'ssh-rsa ...'

```

---

We suggest creating a backup of the `install-config.yaml` file because it is deleted automatically after installation.

### Configuring the DNS server

A DNS server configuration is required to resolve Red Hat OCP cluster records, such as `<component>.<cluster_name>.<base_domain>`.

For more information about the required DNS server configuration, see the Red Hat OCP on Power installation documentation that is available at [this web page](#).

Figure 4-3 shows the required DNS configuration (from the example of our Windows DNS Manager) for our new Red Hat OCP on Power cluster environment.

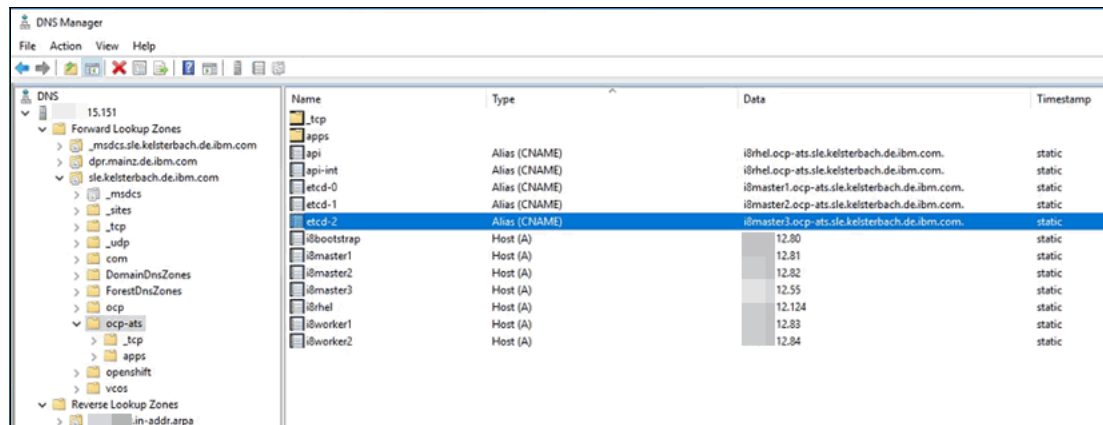


Figure 4-3 DNS-Manager

Figure 4-4 shows the DNS configuration of the `ocp-ats/_tcp` tree.

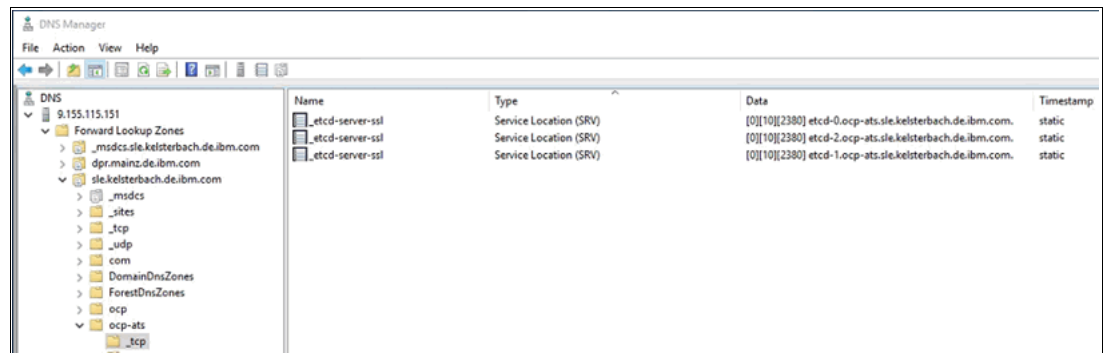


Figure 4-4 DNS ocp-ats/\_tcp information

The following steps show the configuration on our bastion node for name resolution with our previously configured existing DNS server:

1. Install `dnsmasq` as a lightweight DNS forwarder and of the `bind-utils` package with DNS query utilities, such as `nslookup`:

```
[root@i7PFE7 etc]# ibm-yum.sh install dnsmasq
[root@i7PFE7 etc]# ibm-yum.sh install bind-utils
```

2. Create the following `dnsmasq` configuration files to point to our DNS server:

```
[root@i7PFE7 dnsmasq.d]# cat /etc/NetworkManager/conf.d/ocp-ats.conf
[main]
dns=dnsmasq
```

```
[root@i7PFE7 dnsmasq.d]# cat /etc/NetworkManager/dnsmasq.d/01-ocp-ats.conf
server=/ocp-ats.sle.kelsterbach.de.ibm.com/10.11.15.151
```

3. Restart `NetworkManager` to make the changes effective and to verify the new DNS resolution:

```
[root@i7PFE7 dnsmasq.d]# systemctl reload NetworkManager
```

We verify the DNS resolution by running `ping` and `nslookup` against our DNS registration for the Kubernetes API server, as shown in Example 4-12.

#### *Example 4-12 DNS Verification*

---

```
[root@i7PFE7 ~]# ping api.ocp-ats.sle.kelsterbach.de.ibm.com
PING i8rhel.ocp-ats.sle.kelsterbach.de.ibm.com (10.11.12.124) 56(84) bytes of data.
64 bytes from i7pfe7.mainz.de.ibm.com (10.11.12.124): icmp_seq=1 ttl=64 time=0.020 ms
64 bytes from i7pfe7.mainz.de.ibm.com (10.11.12.124): icmp_seq=2 ttl=64 time=0.007 ms
64 bytes from i7pfe7.mainz.de.ibm.com (10.11.12.124): icmp_seq=3 ttl=64 time=0.011 ms

--- i8rhel.ocp-ats.sle.kelsterbach.de.ibm.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 33ms
rtt min/avg/max/mdev = 0.007/0.012/0.020/0.006 ms
```

```
[root@i7PFE7 ~]# nslookup api.ocp-ats.sle.kelsterbach.de.ibm.com
Server:          127.0.0.1
Address:         127.0.0.1#53
```

```
Non-authoritative answer:
api.ocp-ats.sle.kelsterbach.de.ibm.com canonical name =
i8rhel.ocp-ats.sle.kelsterbach.de.ibm.com.
Name:   i8rhel.ocp-ats.sle.kelsterbach.de.ibm.com
Address: 10.11.12.124
```

---

## **Configuring the load balancer**

A Red Hat OCP cluster requires a load balancer for the Kubernetes API server and machine configuration server that is running on the master nodes and for application HTTP/HTTPs ingress traffic on the worker nodes.

The following steps show the installation and configuration of a load balancer on our bastion node:

1. Install `HAProxy` as the load balancer for our test environment:

```
[root@i7PFE7 dnsmasq.d]# ibm-yum.sh install haproxy
```

For configuring the HAProxy, we referred to the OpenShift installation experience example that is available at [this web page](#).

2. To make available the HAProxy statistics by using its GUI., add a “listen stats” section to the `haproxy.cfg` configuration file with specifying port 9000 for the HAProxy GUI. The needed changes are marked in blue in Example 4-13.

*Example 4-13 Configuration file haproxy.cfg*

---

```
[root@i7PFE7 haproxy]# cat /etc/haproxy/haproxy.cfg
listen ingress-http

    bind *:80
    mode tcp

    server worker0 10.11.12.83:80 check
    server worker1 10.11.12.84:80 check

listen ingress-https

    bind *:443
    mode tcp

    server worker0 10.11.12.83:443 check
    server worker1 10.11.12.84:443 check

listen api

    bind *:6443
    mode tcp

    server bootstrap 10.11.12.80:6443 check
    server master0 10.11.12.81:6443 check
    server master1 10.11.12.82:6443 check
    server master2 10.11.12.55:6443 check

listen api-int

    bind *:22623
    mode tcp

    server bootstrap 10.11.12.80:22623 check
    server master0 10.11.12.81:22623 check
    server master1 10.11.12.82:22623 check
    server master2 10.11.12.55:22623 check

listen stats
    bind :9000
    mode http
    stats enable
    stats uri /
    monitor-uri /healthz
```

---

On our bastion node, we display the used network ports by using the `ss` command and can confirm that the wanted port 8080 for our HTTP Server is unused, as shown in Example 4-14.

*Example 4-14 Sockets information list, check that port 8080 is unused*

---

```
[root@i7PFE7 haproxy]# ss -tulnp
```

Netid	State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	users:((("dnsmasq",pid=278736,fd=14))
udp	UNCONN	0	0	0.0.0.0:42679	0.0.0.0:*	users:((("dnsmasq",pid=278736,fd=4))
udp	UNCONN	0	0	127.0.0.1:53	0.0.0.0:*	users:((("dnsmasq",pid=278736,fd=4))
udp	UNCONN	0	0	127.0.0.1:323	0.0.0.0:*	users:((("chronyd",pid=1295,fd=5))
udp	UNCONN	0	0	:::1:323	:::)*	users:((("chronyd",pid=1295,fd=6))
tcp	LISTEN	0	32	127.0.0.1:53	0.0.0.0:*	users:((("dnsmasq",pid=278736,fd=5))
tcp	LISTEN	0	128	0.0.0.0:22	0.0.0.0:*	users:((("sshd",pid=1342,fd=5))
tcp	LISTEN	0	128	:::):22	:::)*	users:((("sshd",pid=1342,fd=7))

---

Port 80 is used by our HAProxy load balancer; therefore, we modify the HTTP listening port from 80 to 8080 by modifying the `/etc/httpd/conf/httpd.conf` file:

```
[root@i7PFE7 haproxy]# vi /etc/httpd/conf/httpd.conf
...
Listen 8080
...
```

## Starting the HTTP Server

As shown in Example 4-15, we use several `systemctl` commands to start our Apache HTTP Server, check its status to confirm it is active, and enable it to automatically start at system start

*Example 4-15 Starting and enabling HTTP Server (some output lines omitted for readability)*

---

```
[root@i7PFE7 ~]# systemctl start httpd
[root@i7PFE7 ~]# systemctl status httpd
...
Active: active (running) since Thu 2020-10-15 12:43:03 CEST;
...
Nov 22 03:11:02 i7PFE7 httpd[1277]: Server configured, listening on: port 8080
...
[root@i7PFE7 ~]# systemctl enable httpd
```

---

## Starting the load balancer (HAProxy)

In our test environment, we changed the SELinux Boolean configuration for HAProxy to serve on any port and disabled the firewall:

```
[root@i7PFE7 haproxy]# setsebool -P haproxy_connect_any on
[root@i7PFE7 haproxy]# systemctl stop firewalld
[root@i7PFE7 haproxy]# systemctl disable firewalld
```

**Note:** For a production environment, disabling the firewall is not suitable; therefore, we suggest to specifically add the network ports that are used by HAProxy to the SELinux firewall configuration by using the `firewall-cmd` according to the example from the referenced OpenShift installation experience website.



To make our changes to the HAProxy configuration effective, we restart HAProxy, verify that is active, and enable its automatic start at system start by using the `systemctl` commands, as shown in Example 4-16.

*Example 4-16 Starting and enabling the proxy server (some output lines are omitted for brevity)*

```
[root@i7PFE7 ~]# systemctl start haproxy
[root@i7PFE7 ~]# systemctl status haproxy
haproxy.service - HAProxy Load Balancer
  Loaded: loaded (/usr/lib/systemd/system/haproxy.service; enabled; vendor preset:
disabled)
  Active: active (running) since Fri 2020-10-16 15:44:38 CEST;
  Main PID: 36323 (haproxy)
  Tasks: 2 (limit: 22150)
  Memory: 28.3M
  CGroup: /system.slice/haproxy.service
          └─36323 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
          └─36324 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
...
Oct 16 15:44:38 i7PFE7 systemd[1]: Started HAProxy Load Balancer.

[root@i7PFE7 ~]# systemctl enable haproxy
```

By using a web browser that is pointing to the IP address and listening port 9000 of our HAProxy server, we check whether the HAProxy GUI responds on port 9000, as shown in Figure 4-5.

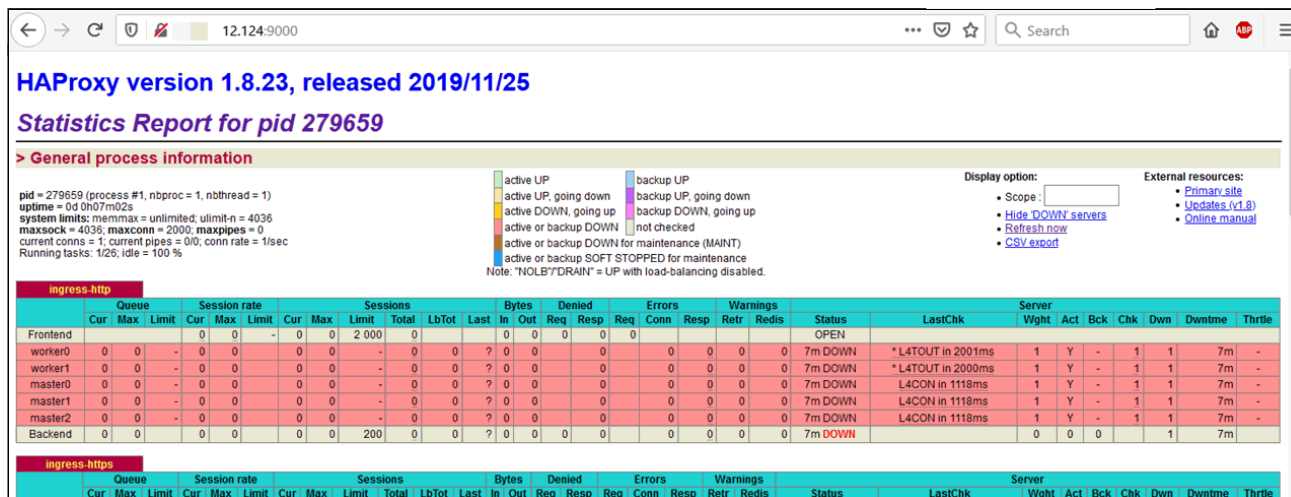


Figure 4-5 HAProxy GUI information

Similarly, we point our web browser to the IP address of our Apache HTTP Server that uses port 8080 to verify that our HTTP Server is serving on port 8080, as shown in Figure 4-6.



Figure 4-6 Apache test page

## Preparing the image files for starting the cluster nodes

We transfer the RHCOS ISO image file `rhcos-installer.ppc64le.iso` to our VIOS media repository that is hosted by the default directory `/var/vio/VMLibrary` for initial start and RHCOS installation of the cluster nodes.

For each cluster node (that is, 1 bootstrap, 3 masters, and 2 worker nodes), our LPAR configuration includes a virtual SCSI client adapter that is paired with a virtual SCSI server adapter on VIOS for hosting the ISO image. That image is loaded into a file-backed optical device by using the VIOS `loadopt` command.

Our Apache HTTP Server's root directory is `/var/www/html`, as configured in the `/etc/httpd/conf/httpd.conf` configuration file. Also, we upload the RHCOS RAW image to the Apache HTTP Server's root document directory:

```
[root@i7PFE7 OCP_install]# cp rhcos-metal.ppc64le.raw.gz /var/www/html
[root@i7PFE7 OCP_install]# ls -l /var/www/html
total 853480
-rw-r--r--. 1 root root 873960908 Oct 14 14:41 rhcos-metal.ppc64le.raw.gz
```

We verified from a browser pointing to the Apache HTTP Server (`http://10.11.12.124:8080/rhcos-metal.ppc64le.raw.gz`) that the raw image file is accessible.

## Creating the Kubernetes manifest and ignition files

Within the installation directory of our bastion node, we proceed with creating the Kubernetes manifest files:

```
[root@i7PFE7 OCP_install]# OpenShift-install create manifests
--dir=/OCP_install
INFO Consuming Install Config from target directory
WARNING Making control-plane schedulable by setting MastersSchedulable to true
for Scheduler cluster settings
```

In accordance with the official Red Hat OCP on Power installation instructions, we modify the /OCP\_install/manifests/cluster-scheduler-02-config.yml manifest file to set "mastersSchedulable: false" to prevent Pods from being scheduled on the control plane (master) nodes:

```
[root@i7PFE7 OCP_install]# sed -i 's/mastersSchedulable:
true/mastersSchedulable: false/g' manifests/cluster-scheduler-02-config.yml
```

We proceed with creating the Kubernetes ignition files that are used for creation of the cluster nodes:

```
[root@i7PFE7 OCP_install]# OpenShift-install create ignition-configs
--dir=/OCP_install
INFO Consuming Master Machines from target directory
INFO Consuming Worker Machines from target directory
INFO Consuming Common Manifests from target directory
INFO Consuming OpenShift Manifests from target directory
INFO Consuming OpenShift Install (Manifests) from target directory
```

The ignition files are copied to the Apache HTTP Server's root directory:

```
[root@i7PFE7 OCP_install]# ls *.ign
bootstrap.ign master.ign worker.ign
[root@i7PFE7 OCP_install]# cp *.ign /var/www/html
```

Read permission is added to the ignition files for "others":

```
[root@i7PFE7 OCP_install]# chmod 644 /var/www/html/*.ign
```

We also verified from a browser pointing to our HTTP Server (http://10.11.12.124:8080/bootstrap.ign) that the files are accessible.

## Creating the RHCOS nodes by using the bootstrap process

We activate our bootstrap node from the HMC command line by using the booting into System Management Services (SMS) mode:

```
chsysstate -r lpar -m Server-9119-MME-SN103E855 -o on -f default -b sms -n
i8Bootstrap
```

In SMS, we select the installation device to be the DVD drive of the virtual SCSI adapter (with the mapped ISO image) and exit SMS, as shown in Example 4-17.

### Example 4-17 System Management Services Interface

---

```
Version FW860.81 (SC860_215)
SMS (c) Copyright IBM Corp. 2000,2016 All rights reserved.
-----
Main Menu
1.  Select Language
2.  Setup Remote IPL (Initial Program Load)
3.  I/O Device Information
4.  Select Console
5.  Select Boot Options
-----
Navigation Keys:
                                           X = eXit System Management Services
-----
Type menu item number and press Enter or select Navigation key:5
```

---

We continue to select the following SMS menu options:

1. Select Install/Boot Device
2. CD/DVD
1. SCSI
1. U9119.MME.103E855-V17-C40-T1 /vdevice/v-scsi@30000028
1. - SCSI CD-ROM  
( loc=U9119.MME.103E855-V17-C40-T1-L8100000000000000 )
2. Normal Mode Boot
1. Yes

On the partition console that displays the Linux boot loader (GRUB) menu with “Install RHEL CoreOS”, we select **e** within the 60 seconds time-out limit and add the Kernel boot parameters that are specific to each node regarding the IP address and ignition file, as described next for the bootstrap node.

Kernel boot parameters added for the bootstrap node, as shown in Figure 4-7:

```
coreos.inst.install_dev=sda
coreos.inst.image_url=http://10.11.12.124:8080/rhcos-metal.ppc64le.raw.gz
coreos.inst.ignition_url=http://10.11.12.124:8080/bootstrap.ign
ip=10.11.12.80::10.11.12.1:255.255.240.0::none nameserver=10.11.15.151
```

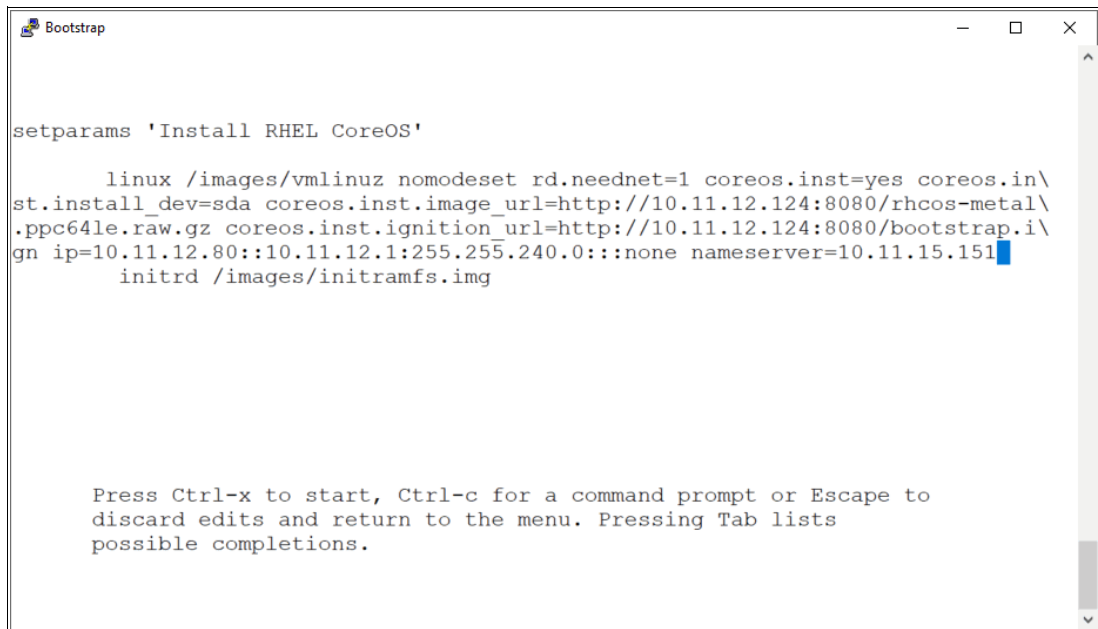


Figure 4-7 Bootstrap information

After adding the kernel boot parameters, we press **Ctrl-x** to start the RHCOS installation. A disk image is written to the node's boot volume, as shown in Figure 4-8, and the node is automatically restarted.

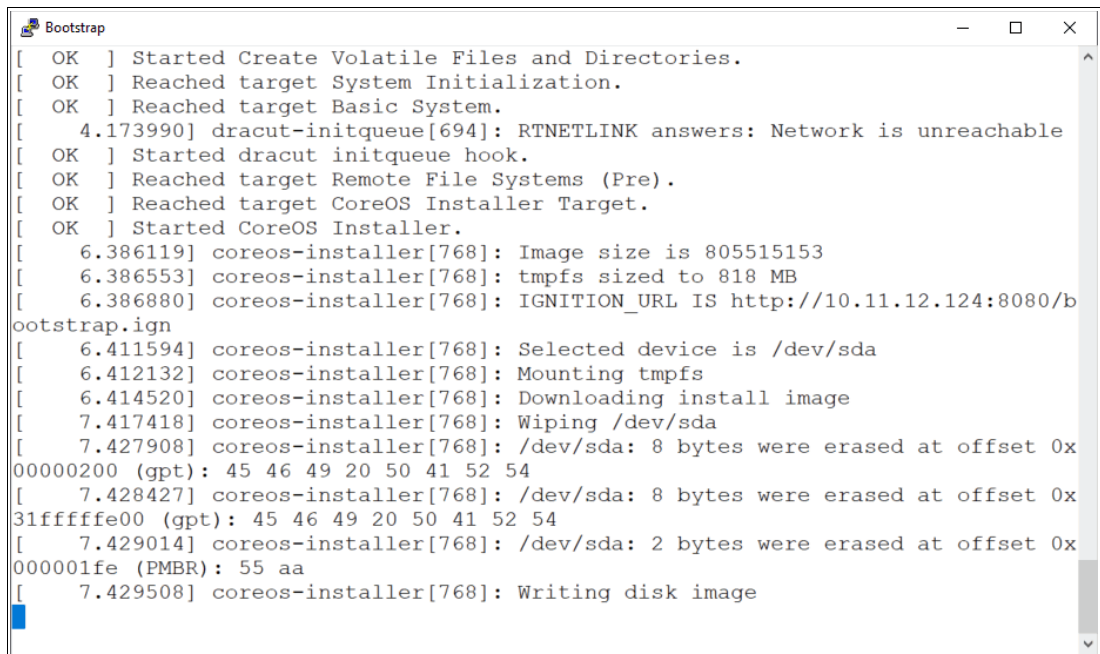


Figure 4-8 Bootstrap progress

When the partition restarts, we enter SMS by selecting **1** to change the boot list to the partition's hard disk drive with the installed RHCOS.

After the bootstrap node is completed, it restarts and we can **ssh** to log into it without a password by using the **core** user. This user uses the previously provided **ssh** key pair; for example, to display some logs.

The first **ssh** connection to the bootstrap node is shown in Example 4-18.

*Example 4-18 ssh to the bootstrap node*

```

[root@i7PFE7 html]# ssh core@10.11.12.80
The authenticity of host '10.11.12.80 (10.11.12.80)' can't be established.
ECDSA key fingerprint is SHA256:t8YEzAReBTp59fZqWkX7ymimTFtOC9KYdisUcuuuFuw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '10.11.12.80' (ECDSA) to the list of known hosts.
Red Hat Enterprise Linux CoreOS 45.82.202007151158-0
  Part of OpenShift 4.5, RHCOS is a Kubernetes native operating system
  managed by the Machine Config Operator (`clusteroperator/machine-config`).

WARNING: Direct SSH access to machines is not recommended; instead,
make configuration changes via `machineconfig` objects:

https://docs.openshift.com/container-platform/4.5/architecture/architecture-rhcos.
html

```

---  
This is the bootstrap node; it will be destroyed when the master is fully up.

The primary services are release-image.service followed by bootkube.service. To watch their status, run e.g.

```
journalctl -b -f -u release-image.service -u bootkube.service
```

In our HAProxy load balancer GUI, we can see that the bootstrap node is responding to requests because its line is marked in green, as shown in Figure 4-9.

The screenshot shows three HAProxy service status tables. The first table is for 'ingress-https', the second for 'api', and the third for 'api-int'. Each table has columns for Queue, Session rate, Sessions, Bytes, Denied, Errors, Warnings, and Server. The 'bootstrap' node in the 'api' and 'api-int' tables is highlighted in green, indicating it is operational. The 'master' and 'worker' nodes are highlighted in red, indicating they are not responding.

Figure 4-9 Bootstrap information

After the bootstrap node becomes operational, we also start our three master and two worker nodes by booting into SMS from the HMC command line, as shown in Example 4-19.

*Example 4-19 HMC commands*

```
chsysstate -r lpar -m Server-9119-MME-SN103E855 -o on -f default -b sms -n i8Master1
chsysstate -r lpar -m Server-9119-MME-SN103E855 -o on -f default -b sms -n i8Master2
chsysstate -r lpar -m Server-9119-MME-SN103E855 -o on -f default -b sms -n i8Master3
chsysstate -r lpar -m Server-9119-MME-SN103E855 -o on -f default -b sms -n i8Worker1
chsysstate -r lpar -m Server-9119-MME-SN103E855 -o on -f default -b sms -n i8Worker2
```

Form the console of each of the five nodes, we select their respective virtual SCSI adapter with the mapped RHCOS ISO image as the install/boot device and add the Kernel parameters specific for each node:

- ▶ For master1:
 

```
coreos.inst.install_dev=sda
coreos.inst.image_url=http://10.11.12.124:8080/rhcos-metal.ppc64le.raw.gz
coreos.inst.ignition_url=http://10.11.12.124:8080/master.ign
ip=10.11.12.81::10.11.12.1:255.255.240.0::none nameserver=10.11.15.151
```
- ▶ For master2:
 

```
coreos.inst.install_dev=sda
coreos.inst.image_url=http://10.11.12.124:8080/rhcos-metal.ppc64le.raw.gz
coreos.inst.ignition_url=http://10.11.12.124:8080/master.ign
ip=10.11.12.82::10.11.12.1:255.255.240.0::none nameserver=10.11.15.151
```

- ▶ For master3:
 

```
coreos.inst.install_dev=sda
coreos.inst.image_url=http://10.11.12.124:8080/rhcos-metal.ppc64le.raw.gz
coreos.inst.ignition_url=http://10.11.12.124:8080/master.ign
ip=10.11.12.55::10.11.12.1:255.255.240.0:::none nameserver=10.11.15.151
```
- ▶ For worker1:
 

```
coreos.inst.install_dev=sda
coreos.inst.image_url=http://10.11.12.124:8080/rhcos-metal.ppc64le.raw.gz
coreos.inst.ignition_url=http://10.11.12.124:8080/worker.ign
ip=10.11.12.83::10.11.12.1:255.255.240.0:::none nameserver=10.11.15.151
```
- ▶ For worker2:
 

```
coreos.inst.install_dev=sda
coreos.inst.image_url=http://10.11.12.124:8080/rhcos-metal.ppc64le.raw.gz
coreos.inst.ignition_url=http://10.11.12.124:8080/worker.ign
ip=10.11.12.84::10.11.12.1:255.255.240.0:::none nameserver=10.11.15.151
```

For each of the worker and master node, we press **Ctrl-x** to start the installer, which reaches out to our HTTP Server and to quay.io to download container images and build the etcd cluster. The requests are handled by our HAProxy load balancer and the Kubernetes API server in the background.

**Note:** You must create the bootstrap and control plane (master) machines now. If the control plane machines are not made schedulable, you must also create at least two compute (worker) machines before you install the cluster.

As we saw for installing the bootstrap node, each node automatically reboots after writing the disk image.

If a node does not reboot from the installed disk, change the boot order by using SMS to the disk so it does not restart from the DVD ISO image.

## Completing and verifying the Red Hat OpenShift cluster installation

The bootstrap process can be monitored after starting the installer.

### *Monitoring the bootstrap cluster installation progress*

We run the following OpenShift-install command and watch for the message about the API service being “up”. We also watch for the bootstrap status completion message, as shown and highlighted in Example 4-20.

*Example 4-20 OpenShift wait for bootstrap completion*

---

```
[root@i7PFE7 OCP_install]# OpenShift-install wait-for bootstrap-complete
--log-level=debug
DEBUG OpenShift Installer 4.3.40
DEBUG Built from commit 43f11cfb64edafaed1941327526efd373a829b63
INFO Waiting up to 30m0s for the Kubernetes API at
https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443...
DEBUG Still waiting for the Kubernetes API: Get
https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443/version?timeout=32s: EOF
DEBUG Still waiting for the Kubernetes API: the server could not find the
requested resource
...
```

```
DEBUG Still waiting for the Kubernetes API: the server could not find the
requested resource
DEBUG Still waiting for the Kubernetes API: Get
https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443/version?timeout=32s: EOF
INFO API v1.16.2+853223d up
INFO Waiting up to 30m0s for bootstrapping to complete...
DEBUG Bootstrap status: complete
INFO It is now safe to remove the bootstrap resources
```

---

If the bootstrap process failed, the installation logs can be gathered for more analysis:

```
[root@i7PFE7 OCP_install]# OpenShift-install gather bootstrap
--dir=/OCP_install --bootstrap 10.11.12.80 --master 10.11.12.81 --master
10.11.12.82 --master 10.11.12.55
```

With the bootstrap status complete, we can power down the bootstrap node, which is not required for setting up the worker nodes:

```
[root@i7PFE7 ~]# ssh core@10.11.12.80 sudo poweroff
```

### **Checking the login to the new cluster**

We check the login to our new OCP cluster with the default system by setting the authentication for the cluster administrator kubeadmin to use the OpenShift client by exporting the cluster's "kubeconfig" file:

```
[root@i7PFE7 OCP_install]# export KUBECONFIG=/OCP_install/auth/kubeconfig
```

**Note:** This statement can also be added to the .bashrc file in the user's home directory so that it is automatically set for each login session.

The **oc** command **whoami** shows the current user:

```
[root@i7PFE7 OCP_install]# oc whoami
system:admin
```

### **Approving all pending certificate requests**

By using the **oc get nodes** command, we can display which nodes joined the cluster, as shown in Example 4-21.

*Example 4-21 cluster node information*

---

```
[root@i7PFE7 OCP_install]# oc get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
i8master1.ocp-ats.sle.kelsterbach.de.ibm.com	Ready	master	6m54s	v1.16.2+853223d
i8master2.ocp-ats.sle.kelsterbach.de.ibm.com	Ready	master	6m47s	v1.16.2+853223d
i8master3.ocp-ats.sle.kelsterbach.de.ibm.com	Ready	master	6m47s	v1.16.2+853223d
i8worker1.ocp-ats.sle.kelsterbach.de.ibm.com	Ready	worker	7m1s	v1.16.2+853223d
i8worker2.ocp-ats.sle.kelsterbach.de.ibm.com	Ready	worker	6m19s	v1.16.2+853223d

---

**Note:** If worker nodes were missing from the cluster, we review and approve any pending certificate signing requests by using the **oc get csr** command.



Example 4-22 shows the approval process.

*Example 4-22 Approval of a node certificate*

---

```
[root@i7PFE7 OCP_install]# oc get csr
NAME      AGE      SIGNERNAME                                  REQUESTOR
CONDITION
csr-9kpz9  13m     kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:OpenShift-machine-config-operator:node-bootstrapper  Approved,Issued
csr-cd2ss  13m     kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:OpenShift-machine-config-operator:node-bootstrapper  Approved,Issued
csr-llsw4  13m     kubernetes.io/kubelet-serving
system:node:i8master2.ocp-ats.sle.kelsterbach.de.ibm.com                    Approved,Issued
csr-r8lvk  13m     kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:OpenShift-machine-config-operator:node-bootstrapper  Approved,Issued
csr-sm4d9  13m     kubernetes.io/kubelet-serving
system:node:i8master1.ocp-ats.sle.kelsterbach.de.ibm.com                    Approved,Issued
csr-v789z  2m34s   kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:OpenShift-machine-config-operator:node-bootstrapper  Pending
csr-wm6rt  13m     kubernetes.io/kubelet-serving
system:node:i8master3.ocp-ats.sle.kelsterbach.de.ibm.com                    Approved,Issued
csr-xj25d  2m42s   kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:OpenShift-machine-config-operator:node-bootstrapper  Pending
```

```
[root@i7PFE7 OCP_install]# oc get csr -o go-template='{{range .items}}{{if not
.status}}{{.metadata.name}}{"\n"}}{{end}}{{end}}' | xargs oc adm certificate approve
certificatesigningrequest.certificates.k8s.io/csr-v789z approved
certificatesigningrequest.certificates.k8s.io/csr-xj25d approved
```

---

The certification approval that is shown in Example 4-22 on page 69 might need to be repeated if worker nodes joined the cluster but did not yet reach “Ready” status.

**PowerVM users:** To prevent packet drops in a PowerVM inter-partition VLAN and SEA network environment, run the `if 1smod ... fi` command (see Example 4-23) immediately after bootstrapping completes. Running this command changes the kernel parameters with the `\MTU` settings on each master and worker node that we accessed by using ssh as core user.

*Example 4-23 MTU parameter setting (command for the first master node only is shown)*

---

```
# ssh to a node or worker
[root@i7PFE7 ~]# ssh core@i8master1

# run command
[core@i8master1 ~]$ if 1smod | grep -q 'ibmveth'; then
    sudo sysctl -w net.ipv4.route.min_pmtu=1450;
    sudo sysctl -w net.ipv4.ip_no_pmtu_disc=1;
    echo 'net.ipv4.route.min_pmtu = 1450' | sudo tee --append /etc/sysctl.d/88-sysctl.conf > /dev/null;
    echo 'net.ipv4.ip_no_pmtu_disc = 1' | sudo tee --append /etc/sysctl.d/88-sysctl.conf > /dev/null;
fi

# check the result
[core@i8master1 ~]$ if [[ -f /etc/sysctl.d/88-sysctl.conf ]]; then cat /etc/sysctl.d/88-sysctl.conf; fi
net.ipv4.route.min_pmtu = 1450
net.ipv4.ip_no_pmtu_disc = 1
```

---

We can watch the remaining steps of the cluster installation by using the `OpenShift-install wait-for install-complete` command, as shown in Example 4-24.

*Example 4-24 Wait for the installation to complete*

---

```
[root@i7PFE7 OCP_install]# OpenShift-install wait-for install-complete
--log-level=debug
DEBUG OpenShift Installer 4.3.40
DEBUG Built from commit 43f11cfb64edafaed1941327526efd373a829b63
DEBUG Fetching Install Config...
DEBUG Loading Install Config...
DEBUG   Loading SSH Key...
DEBUG   Loading Base Domain...
DEBUG   Loading Platform...
DEBUG   Loading Cluster Name...
DEBUG   Loading Base Domain...
DEBUG   Loading Platform...
DEBUG   Loading Pull Secret...
DEBUG   Loading Platform...
DEBUG Using Install Config loaded from state file
DEBUG Reusing previously-fetched Install Config
INFO Waiting up to 30m0s for the cluster at
https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443 to initialize...
DEBUG Still waiting for the cluster to initialize: Multiple errors are preventing
progress:
* Could not update oauthclient "console" (292 of 498): the server is down or not
responding
...
DEBUG Still waiting for the cluster to initialize: Working towards 4.3.40: 81%
complete
DEBUG Still waiting for the cluster to initialize: Working towards 4.3.40: 96%
complete
DEBUG Still waiting for the cluster to initialize: Working towards 4.3.40: 97%
complete
DEBUG Still waiting for the cluster to initialize: Working towards 4.3.40: 99%
complete
DEBUG Still waiting for the cluster to initialize: Working towards 4.3.40: 99%
complete, waiting on marketplace, monitoring
DEBUG Still waiting for the cluster to initialize: Working towards 4.3.40: 100%
complete
DEBUG Cluster is initialized
INFO Waiting up to 10m0s for the OpenShift-console route to be created...
DEBUG Route found in OpenShift-console namespace: console
DEBUG Route found in OpenShift-console namespace: downloads
DEBUG OpenShift console route is created
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export
KUBECONFIG=/OCP_install/auth/kubeconfig'
INFO Access the OpenShift web-console here:
https://console-openshift-console.apps.ocp-ats.sle.kelsterbach.de.ibm.com
INFO Login to the console with user: kubeadmin, password: pCjrI-7qn79-VgAgB-BvIqV
```

---

**Note:** The web console URL can also be displayed by using the `oc whoami --show-console` command.

We verify that all cluster operators are in available state by running the `oc get co` (where `co` is the short name for clusteroperators) command, as shown in Example 4-25.

Example 4-25 Cluster operator availability

```
[root@i7PFE7 ~]# oc get clusteroperators
```

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
authentication	4.3.40	True	False	False	8m15s
cloud-credential	4.3.40	True	False	False	22m
cluster-autoscaler	4.3.40	True	False	False	5m50s
console	4.3.40	True	False	False	5m57s
dns	4.3.40	True	False	False	19m
image-registry	4.3.40	True	False	False	16m
ingress	4.3.40	True	False	False	13m
insights	4.3.40	True	False	False	16m
kube-apiserver	4.3.40	True	False	False	18m
kube-controller-manager	4.3.40	True	False	False	18m
kube-scheduler	4.3.40	True	False	False	18m
machine-api	4.3.40	True	False	False	16m
machine-config	4.3.40	True	False	False	19m
marketplace	4.3.40	True	False	False	3m20s
monitoring	4.3.40	True	False	False	4s
network	4.3.40	True	False	False	21m
node-tuning	4.3.40	True	False	False	17m
OpenShift-apiserver	4.3.40	True	False	False	8m13s
OpenShift-controller-manager	4.3.40	True	False	False	18m
OpenShift-samples	4.3.40	True	False	False	5m42s
operator-lifecycle-manager	4.3.40	True	False	False	17m
operator-lifecycle-manager-catalog	4.3.40	True	False	False	17m
operator-lifecycle-manager-packageserver	4.3.40	True	False	False	16m
service-ca	4.3.40	True	False	False	21m
service-catalog-apiserver	4.3.40	True	False	False	17m
service-catalog-controller-manager	4.3.40	True	False	False	17m
storage	4.3.40	True	False	False	15m

Figure 4-10 shows the web console of our new Red Hat OCP cluster, which displays the dashboards view after logging in with the `kubeadmin` user.

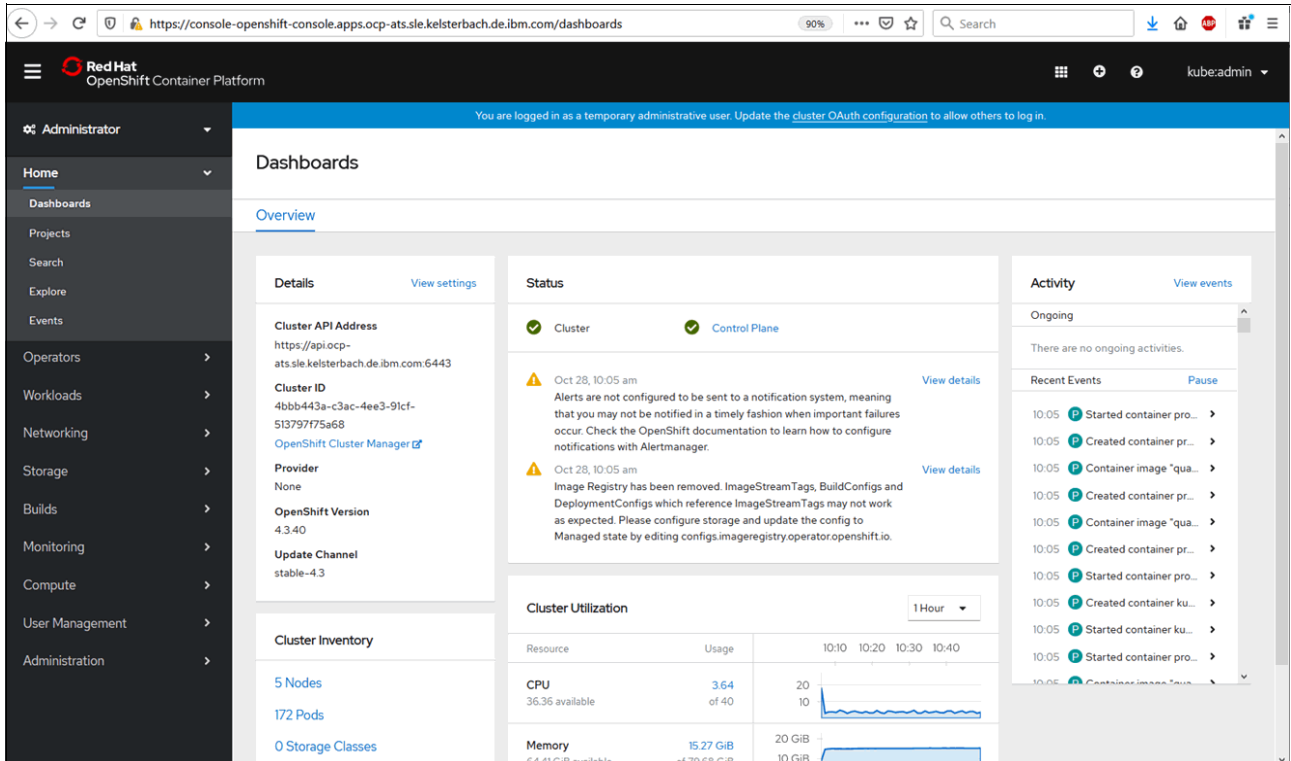


Figure 4-10 New Red Hat OCP cluster

**Note:** For building and deploying your own applications following the Red Hat OCP cluster installation, a local image registry still must be configured on persistent ReadWriteMany (RWX) storage; for example, provided by OpenShift Container Storage, object storage, or NFS.

To more information about installing the IBM CSI driver for an IBM Power Systems Red Hat OCP cluster for provisioning persistent storage from IBM Spectrum Virtualize-based SAN block storage systems for stateful containerized applications, see 4.3, “CSI deployment on IBM Power by using the command-line interface”.

## 4.3 CSI deployment on IBM Power by using the command-line interface

This section describes the installation of the IBM block storage CSI driver on IBM Power using Red Hat OpenShift version 4.3 and IBM block storage CSI driver 1.3.0. You should always check the supported operating system and the supported driver version before beginning the installation of the CSI driver as shown in Appendix B, “Container Storage Interface support matrix” on page 131. The examples for worker nodes that are used in this section use Red Hat Core OS (RHCOS).

This section describes the following steps:

1. Configuring the storage system
2. Configuring the multipath driver on the worker nodes
3. Installing the driver using CLI
4. Configuring the CSI driver using the CLI:
  - a. Creating an array secret
  - b. Creating storage classes
  - c. Creating a PersistentVolumeClaim (PVC)
  - d. Creating a StatefulSet

Before beginning the installation of the CSI driver, you should verify that you comply with the prerequisites that are listed in the section Compatibility and requirements of the CSI 1.3 documentation that is available at this [IBM Documentation web page](#).

That section describes the needed open ports on the Red Hat Core OS firewall, and packages.

### 4.3.1 Configuring the storage system

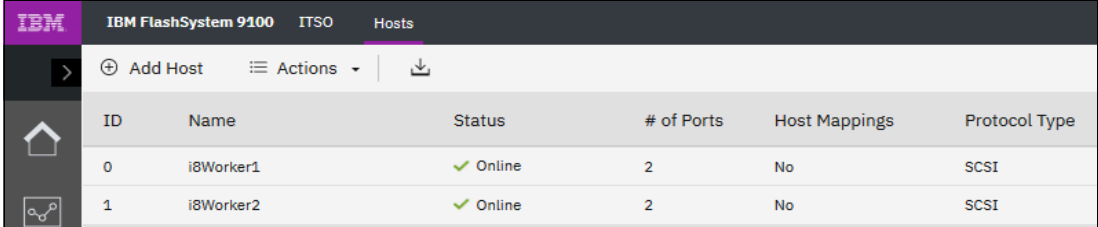
The OpenShift environment setup for our example uses two IBM Power worker nodes: i8Worker1 and i8Worker2. The FC worldwide ports names can be retrieved by using the Hardware Management Console (HMC) of the Power Systems server. In our configuration, a FlashSystem 9110 is used, as shown in Example 4-26.

*Example 4-26 Spectrum Virtualize command to create the IBM Power hosts*

```
superuser>mkhost -fcwwpn C05076082D8201AA:C05076082D8201AC -iogrp 0 -name
i8Worker1 -protocol scsi -site 1 -type generic
Host, id [7], successfully created
```

```
superuser>mkhost -fcwwpn C05076082D8201AE:C05076082D8201B0 -iogrp 0 -name
i8Worker2 -protocol scsi -site 1 -type generic
Host, id [8], successfully created
```

The created hosts can be seen in the FlashSystem 9110 GUI, as shown in Figure 4-11.



The screenshot shows the IBM FlashSystem 9110 GUI with the 'Hosts' tab selected. The table below represents the data shown in the GUI:

ID	Name	Status	# of Ports	Host Mappings	Protocol Type
0	i8Worker1	Online	2	No	SCSI
1	i8Worker2	Online	2	No	SCSI

Figure 4-11 Storage System list of IBM Power workers

The SAN configuration must be updated and the worker nodes must be zoned to the storage system.

### 4.3.2 Configuring the multipath driver on the worker nodes

Before beginning the installation of the CSI driver, the multipath configuration for the IBM storage on the OpenShift Container Platform worker nodes must be created and activated. The CSI documentation lists a configuration file that must be saved as 99-ibm-attach.yaml on the OpenShift bastion node. The file content is available at this [IBM Documentation web page](#).

**Note:** The 99-ibm-attach.yaml configuration file overrides any files that exist on your system. Use this file only if the files are not created.

If one or more files were created, edit this yaml file as necessary.

Example 4-27 lists the first 10 lines of the 99-ibm-attach.yaml file.

*Example 4-27 First 10 lines of the 99-ibm-attach.yaml file*

```
[root@i7PFE7 OCP_install]$ head 99-ibm-attach.yaml
apiVersion: machineconfiguration.OpenShift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.OpenShift.io/role: worker
```

```
name: 99-ibm-attach
spec:
  config:
    ignition:
      version: 2.2.0
```

---

A new multipath configuration is created for the worker nodes by using the `oc apply` command, as shown in Example 4-28. The machine configuration shows if the yaml file was applied.

*Example 4-28 Creation of the multipath configuration*

---

```
[root@i7PFE7 OCP_install]$ oc apply -f 99-ibm-attach.yaml
machineconfig.machineconfiguration.openshift.io/99-ibm-attach created

# Checking the Kubernetes "machineconfig" custom resource to see whether the yaml
file was accepted:

[root@i7PFE7 OCP_install]$ oc get machineconfig | egrep "NAME|99-ibm-attach"
NAME                               GENERATEDBYCONTROLLER  IGNITIONVERSION  CREATED
99-ibm-attach                       2.2.0                 15s
```

---

The next steps are to check the configuration file on the worker nodes and to check the status of the multipath daemon for one node, as shown in Example 4-29.

*Example 4-29 Checking the multipath configuration*

---

```
# check the multipath configuration
# the output is truncated to the FlashSystem 9100 entry:
[root@i7PFE7 ~]# ssh core@i8Worker1 sudo cat /etc/multipath.conf
devices {
  device {
    vendor "IBM"
    product "2145"
    path_checker tur
    features "1 queue_if_no_path"
    path_grouping_policy group_by_prio
    path_selector "service-time 0" # Used by Red Hat 7.x
    prio alua
    rr_min_io_rq 1
    rr_weight uniform
    no_path_retry "5"
    dev_loss_tmo 120
    failback immediate
  }
}

# check the status of the multipath daemon
# the output is truncated to show the status only
[root@i7PFE7 ~]# ssh core@i8Worker1 sudo systemctl status multipathd
? multipathd.service - Device-Mapper Multipath Device Controller
   Loaded: loaded (/usr/lib/systemd/system/multipathd.service; enabled; vendor
   preset: enabled)
   Active: active (running) since Thu 2020-11-12 10:17:23 UTC; 5min ago
```

---

The status is active (running); therefore, the multipath configuration was successfully enabled on the worker nodes.

### 4.3.3 Installing the driver by using CLI

This section describes the CSI operator and the CSI driver installation by using the CLI. The operator for IBM block storage CSI driver can be installed directly from GitHub. Installing the CSI driver is part of the operator installation process.

Complete the following high-level steps to install the operator and driver by using GitHub through a command-line interface. It is always good practice to install a driver in a dedicated namespace and never to use the default namespace:

- ▶ Create a namespace for the CSI driver
- ▶ Download the manifest from GitHub and update the required fields.
- ▶ Install the CSI operator
- ▶ Install the CSI driver

These steps are described next.

#### Creating a dedicated namespace for the CSI driver

The dedicated namespace that we use for our installation example of the CSI driver is `ibm-block-csi`.

**Note:** Do not install the CSI operator and driver in the default namespace. You must create a dedicated namespace for the CSI operator and driver.

Example 4-30 shows the creation of the namespace and how to change the environment to the newly generated namespace.

*Example 4-30 Dedicated namespace for the CSI driver installation*

```
[root@i7PFE7 OCP_install]# kubectl create ns ibm-block-csi
namespace/ibm-block-csi created
```

```
[root@i7PFE7 OCP_install]# oc project ibm-block-csi
Now using project "ibm-block-csi" on sever
"https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443".
```

#### Installing the CSI operator

Complete the following steps:

1. Download the manifest from GitHub and update the required fields. You can download the CSI operator and driver from GitHub, as shown in Example 4-31. Listing the first lines of the file shows the driver name `ibmblockcsis.csi.ibm.com`.

*Example 4-31 Preparing the CSI operator installation*

```
[root@i7PFE7 OCP_install]# curl
https://raw.githubusercontent.com/IBM/ibm-block-csi-operator/v1.3.0/deploy/installer/generated/ibm-block-csi-operator.yaml > ibm-block-csi-operator.yaml
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 93893  100 93893    0     0  189k      0  --:--:-- --:--:-- --:--:--  189k
```

```
[root@i7PFE7 OCP_install]# ls -l
total 404
-rw-r--r--. 1 root root 93893 Nov 2 15:08 ibm-block-csi-operator.yaml

[root@i7PFE7 OCP_install]# head -10 ibm-block-csi-operator.yaml
# Code generated by update-installer.sh. DO NOT EDIT.

apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: ibmblockcsis.csi.ibm.com
spec:
  group: csi.ibm.com
  names:
    kind: IBMBlockCSI
[root@i7PFE7 OCP_install]#
```

---

The `ibm-block-csi-operator.yaml` file contains the default namespace, which must be changed to the previously created dedicated namespace for the CSI driver. Replace every line that includes “`namespace: default`” with “`namespace: ibm-block-csi`”.

2. Install the CSI operator and check that the operator is running, as shown in Example 4-32.

*Example 4-32 CSI operator installation*

---

```
[root@i7PFE7]# kubectl -n ibm-block-csi apply -f ibm-block-csi-operator.yaml
customresourcedefinition.apiextensions.k8s.io/ibmblockcsis.csi.ibm.com created
deployment.apps/ibm-block-csi-operator created
clusterrole.rbac.authorization.k8s.io/ibm-block-csi-operator created
clusterrolebinding.rbac.authorization.k8s.io/ibm-block-csi-operator created
serviceaccount/ibm-block-csi-operator created

# Verifying that the CSI operator is running in its pod:

[root@i7PFE7]# kubectl get pod -l app.kubernetes.io/name=ibm-block-csi-operator -n
ibm-block-csi
NAME                                READY   STATUS    RESTARTS   AGE
ibm-block-csi-operator-bdfb89bdd-c76gm 1/1     Running   0           2m18s
```

---

## Installing the CSI driver

Install the IBM block storage CSI driver by creating an IBMBlockCSI custom resource. Complete the following steps:

1. Download the manifest from GitHub for the IBM Power architecture and update the required fields. You can download the CSI operator and driver from GitHub, as shown in Example 4-33. Listing the first lines of the file shows the driver name `ibm-block-csi`.

*Example 4-33 Preparing the CSI driver installation*

---

```
[root@i7PFE7 OCP_install]# curl
https://raw.githubusercontent.com/IBM/ibm-block-csi-operator/v1.3.0/deploy/crds/csi
i.ibm.com_v1_ibmblockcsi_cr_ocp.yaml > csi.ibm.com_v1_ibmblockcsi_cr.yaml
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 2211  100 2211    0     0  6263      0  --:--:--  --:--:--  --:--:--  6245

[root@i7PFE7 OCP_install]# ls -l
```



```
-rw-r--r--. 1 root root 2211 Nov 24 23:15 csi.ibm.com_v1_ibmblockcsi_cr.yaml
```

```
[root@i7PFE7 OCP_install]# head csi.ibm.com_v1_ibmblockcsi_cr.yaml
apiVersion: csi.ibm.com/v1
kind: IBMBlockCSI
metadata:
  name: ibm-block-csi
  namespace: default
```

---

The `csi.ibm.com_v1_ibmblockcsi_cr.yaml` file contains the default namespace that must be changed to the previously created dedicated namespace for the CSI driver. Replace every line that contains “namespace: default” with “namespace: ibm-block-csi”.

**Important:** The current CSI driver nodeID is limited to 128 characters. The CSI driver installation fails if the nodeID exceeds this limitation.

2. Install the CSI driver and check the operator and driver status, as shown in Example 4-34.

*Example 4-34 CSI driver installation*

---

```
[root@i7PFE7 OCP_install]# kubectl -n ibm-block-csi apply -f csi.ibm.com_v1_ibmblockcsi_cr.yaml
ibmblockcsi.csi.ibm.com/ibm-block-csi created
```

```
[root@i7PFE7 OCP_install]# oc get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/ibm-block-csi-controller-0	5/5	Running	0	1d
pod/ibm-block-csi-node-9rp6n	3/3	Running	24	1d
pod/ibm-block-csi-node-k7fmh	3/3	Running	24	1d
pod/ibm-block-csi-operator-bdfb89bdd-hrzgn	1/1	Running	0	1d

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
daemonset.apps/ibm-block-csi-node	2	2	2	2	2	<none>	1d

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/ibm-block-csi-operator	1/1	1	1	1d

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/ibm-block-csi-operator-bdfb89bdd	1	1	1	1d

NAME	READY	AGE
statefulset.apps/ibm-block-csi-controller	1/1	1d

---

For issues with a CSI driver pod not reaching a running status, its logs can be displayed for further analysis by using the `oc logs <pod_name> -c <container_name>` command. If you encounter an error message, such as “spec.drivers[0].nodeID: Invalid value: “<value>”: must be 128 characters or less”, you must change your worker node setup.

The nodeID contains the iSCSI qualified name and all the WWPNs of the FC ports. This string can exceed the 128 character size limit for the nodeID.

In our test environment, the nodeID exceeded the 128 characters; therefore, we reduced the length by removing the IQN and retained all of the WWPNs.

We safely removed the IQN part of the nodeID by keeping all our WWPNs because we did not use iSCSI. To remove the IQN, we renamed the `/etc/iscsi/initiatorname.iscsi` file on all worker nodes and then restarted them. Then, the changed length of the nodeID was fewer than 128 characters.

For more information about the worker node, run the `oc describe node` command. Example 4-35 shows how to use the command to list the nodeID.

*Example 4-35 Node detail information (shown in blue)*

---

```
[root@i7PFE7 ~]# oc describe node i8worker1 | egrep -a -A1 -i nodeid
Annotations:          csi.volume.kubernetes.io/nodeid:

{"block.csi.ibm.com":"i8worker1.ocp-ats.sle.kelsterbach.de.ibm.com;;c05076082d8201a2:c05076082d8201a4:c05076082d8201aa:c05076082d8201ac"}
```

---

### 4.3.4 Configuring the CSI driver by using the CLI

After the CSI operator and CSI driver are installed and running, the relevant storage classes and secrets must be created. You must create an array secret and storage classes to run stateful applications by using IBM block storage systems.

#### Creating an array secret

Create a storage system secret to define the storage system credentials (user name and password) and address.

**Important:** When your storage system password is changed, be sure to also change the passwords in the corresponding secrets. Otherwise, the passwords are not synchronized and a PersistentVolumeClaim cannot be created.

Create an array secret file by using a template that is available at this [IBM Documentation web page](#).

Example 4-36 shows the example file `FS9110-secret.yaml` with the correct storage system address and access credentials. You can run the `base64` command to create an encrypted password:

```
[root@i7PFE7 ~]# echo -n "<clear_text_password>" | base64
PGNsZWfYX3R1eHRfcGFzc3dvcmQ+
```

*Example 4-36 Array secret file*

---

```
kind: Secret
apiVersion: v1
metadata:
  name: fs9110-secret
  namespace: ibm-block-csi
type: Opaque
stringData:
  management_address: 10.11.22.238           # Array management addresses
  username: superuser                       # Array username
data:
  password: RG90b3REZWNvZGU=                # base64 array password
```

---

Example 4-37 shows the command that is used to create the array secret.

*Example 4-37 Creating the array secret*

---

```
[root@i7PFE7 OCP_install]# kubectl apply -f FS9110-secret.yaml
secret/fs9110-secret created
```

---

Figure 4-12 shows the newly created secret in the OpenShift GUI.

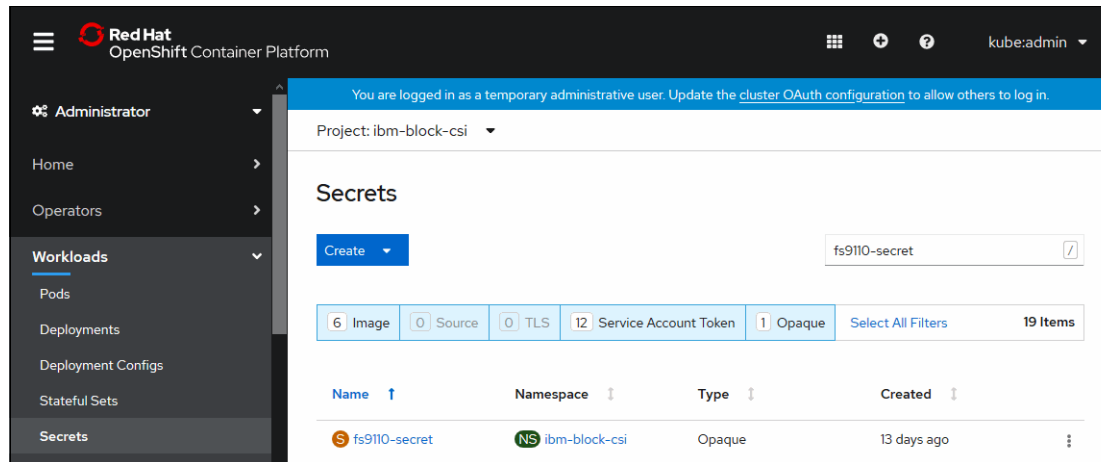


Figure 4-12 OpenShift GUI array secrets list

## Creating storage classes

Create a storage class for each storage service class to be used as defined by a pool name, secret reference, space efficiency, and file system type.

Create a storage class yaml file by using a template that is available at this [IBM Documentation web page](#).

The following space efficiency parameters for SpaceEfficiency for the IBM Spectrum Virtualize Family are available:

- ▶ Thick (default)
- ▶ Thin
- ▶ Compressed
- ▶ Deduplicated

Example 4-38 shows the example file FS9110-storageclass-thin.yaml with the correct storage system address, access credentials, and the space efficiency parameter thin.

Example 4-38 Storage class file

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fs9110-storageclass-thin
provisioner: block.csi.ibm.com
parameters:
  SpaceEfficiency: thin # Optional.
  pool: Legacy_0

  csi.storage.k8s.io/provisioner-secret-name: fs9110-secret
  csi.storage.k8s.io/provisioner-secret-namespace: ibm-block-csi
  csi.storage.k8s.io/controller-publish-secret-name: fs9110-secret
  csi.storage.k8s.io/controller-publish-secret-namespace: ibm-block-csi

  csi.storage.k8s.io/fstype: xf # Optional. Values ext4\xfs. The default is ext4.
  volume_name_prefix: ocp-ats # Optional.
```

Example 4-39 shows the command used to create the storage class.

*Example 4-39 Creating the storage class*

```
[root@i7PFE7 OCP_install]# kubectl apply -f FS9110-storageclass-thin.yaml
storageclass.storage.k8s.io/fs9110-storageclass-thin created
```

Figure 4-13 shows a list of the available storage classes in the OpenShift GUI, including the storage class that is shown in Example 4-39.

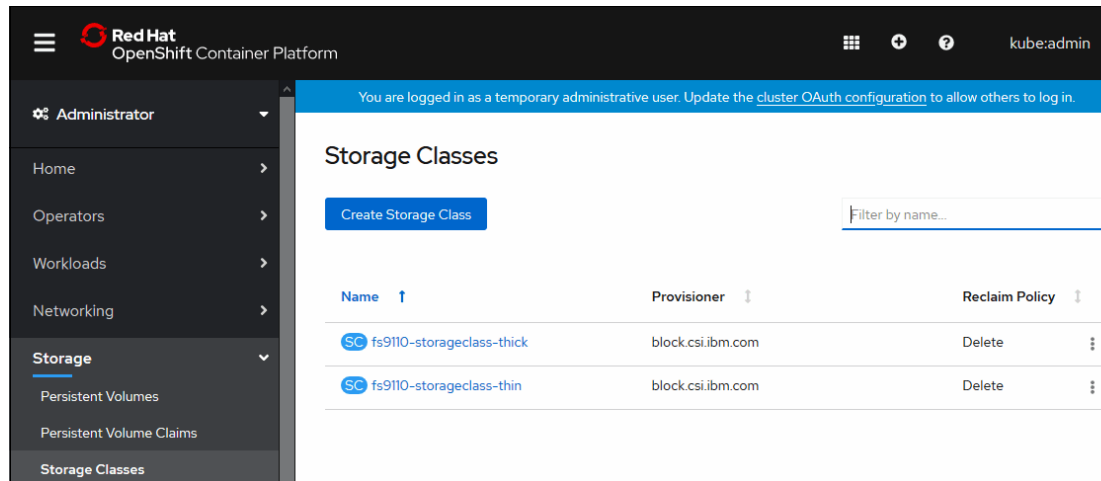


Figure 4-13 OpenShift GUI storage classes list

## Creating a PersistentVolumeClaim

Before creating the PVC, we create a namespace for the user workload. A user workload does not run in the CSI operator and driver namespace.

**Note:** Create a namespace for the user workload. You can use an existing namespace, which is different from the default or CSI driver namespace.

The dedicated namespace that is used for the volume and workload examples is fs9110.

Example 4-40 shows creating a new namespace, including automatically changing the user's environment.

*Example 4-40 New namespace for the CSI driver workload examples*

```
[root@i7PFE7 ~]# oc new-project fs9110 --description="FS9110 CSI Driver Testing"
Now using project "fs9110" on server
"https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443".
```

The IBM block storage CSI driver supports the use of file system and raw block volume persistent volume claims. By default, a PVC is for a file system type volume. The file system type to be used, such as the default ext4 file system or optionally an xfs file system, is specified within the created storage class, as shown in Example 4-38 on page 79.

If you prefer to use raw block volumes instead, you must define the type as Block in the PVC file's volumeMode specification.

Create a PersistentVolumeClaim file by using a template that is available at this [IBM Documentation web page](#).

Example 4-41 shows the example file `FS9110-secret.yaml` with the correct storage class.

*Example 4-41 PersistentVolumeClaim file*

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: fs9110-pvc-filesystem
spec:
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: fs9110-storageclass-thin
```

Example 4-42 shows the command that is used to create the PVC.

*Example 4-42 Creating the PersistentVolumeClaim*

```
[root@i7PFE7 OCP_install]# kubectl apply -f fs9110-pvc-filesystem.yaml
persistentvolumeclaim/fs9110-pvc-filesystem created
```

Figure 4-14 shows the newly created PVC and the corresponding Persistent Volume (PV).

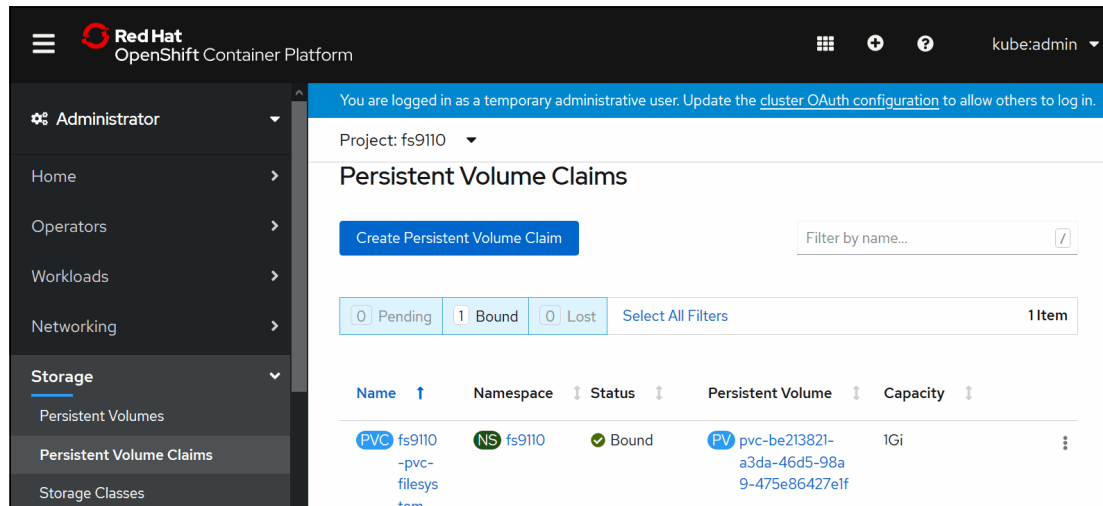


Figure 4-14 OpenShift GUI PVC and corresponding PV

The PVC is *bound* to a volume (as shown in the GUI in Figure 4-14) and can be checked in the command line, as shown in Example 4-43.

*Example 4-43 Checking the PVC and its PV, output information is truncated for better readability*

```
[root@i7PFE7 ~]# oc get pvc
NAME                STATUS    VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS          AGE
fs9110-pvc-filesystem  Bound    pvc-be213821-a3da-46d5-98a9-475e86427e1f  1Gi        RWO            fs9110-storageclass-thin  10m

[root@i7PFE7 ~]# oc get pv
NAME                CAPACITY   ACCESS MODES   STATUS    CLAIM                                STORAGECLASS          REASON   AGE
pvc-be213821-a3da-46d5-98a9-475e86427e1f  1Gi        RWO            Bound    fs9110/fs9110-pvc-filesystem        fs9110-storageclass-thin              10m
```

**Note:** If you delete the PVC, the PV and the corresponding volume on the storage system are deleted.

The storage system volume name is preceded with `ocp-ats` as defined in the storage class that is used and shown in Figure 4-15.

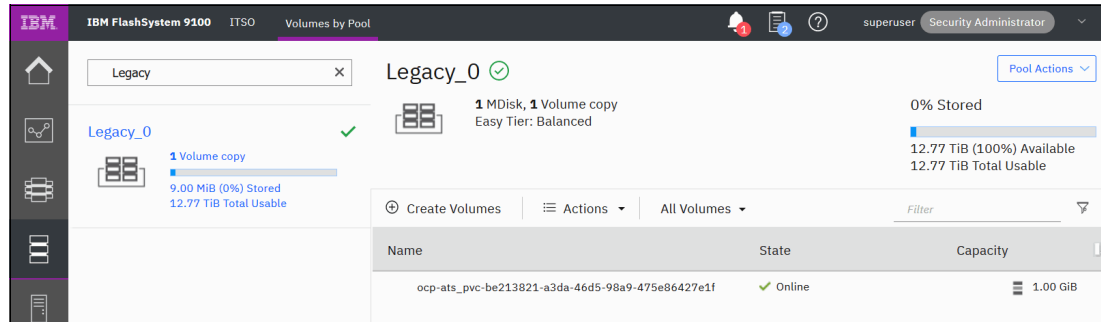


Figure 4-15 FlashSystem 9100 volume created by OpenShift

## Creating a StatefulSet

A Kubernetes *StatefulSet* is the workload API object that is used to manage stateful applications, similar to Deployments, which are used for stateless applications. They provide a persistent pod identifier that is maintained across pod rescheduling so that new pods that are replacing failed pods easily can be matched to the persistent storage volumes.

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

StatefulSets can include volumes with file systems, raw block volume systems, or both. Create a StatefulSet file by using a template that is available at this [IBM Documentation web page](#).

This template uses an PVC because the example is configured for one replica only. If you want to create two or more replicas of your StatefulSet, the volume description must be replaced by using a volume claim template that enables each replica to be provisioned with its own volume. This change is necessary because a single volume cannot be attached to multiple pods by using the IBM block storage CSI driver.

Example 4-44 shows the following example file with the volume claim template and the OpenShift universal base image (UBI):

```
fs9110-statefulset-file-system-volumeClaimTemplates.yaml
```

*Example 4-44 StatefulSet file*

```
kind: StatefulSet
apiVersion: apps/v1
metadata:
  name: fs9110-statefulset-file-system
spec:
  selector:
    matchLabels:
      app: fs9110-statefulset
  serviceName: fs9110-statefulset
  replicas: 3
  template:
    metadata:
```

```

labels:
  app: fs9110-statefulset
spec:
  containers:
  - name: fs9110-container
    image: registry.access.redhat.com/ubi8/ubi:latest
    command: [ "/bin/sh", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
    volumeMounts:
    - name: fs9110-volume-filesystem
      mountPath: "/data"
volumeClaimTemplates:
- metadata:
  name: fs9110-volume-filesystem
spec:
  accessModes: [ "ReadWriteOnce" ]
  storageClassName: "fs9110-storageclass-thin"
resources:
  requests:
    storage: 2Gi

```

Example 4-45 shows the command that is used to create the StatefulSet.

*Example 4-45 Creating the StatefulSet*

```

[root@i7PFE7]# kubectl apply -f fs9110-statefulset-file-system-volumeClaimTemplates.yaml
statefulset.apps/fs9110-statefulset-filesystem created

```

# check the created pods

```

[root@i7PFE7 OCP_install]# oc get pod
NAME                                READY   STATUS              RESTARTS   AGE
fs9110-statefulset-filesystem-0    1/1    Running             0           26s
fs9110-statefulset-filesystem-1    1/1    Running             0           20s
fs9110-statefulset-filesystem-2    0/1    ContainerCreating  0           10s

```

Figure 4-16 shows the new created StatefulSet and the corresponding number of pods.

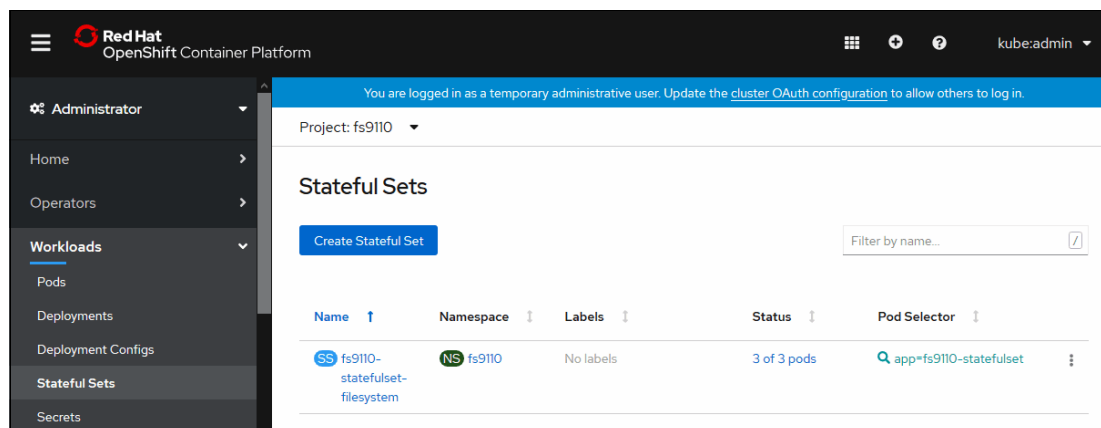


Figure 4-16 OpenShift GUI Stateful Sets

The PVC is *bound* to a volume (as shown in the GUI that is shown in Figure 4-16 on page 83) and can be checked in the command line as shown in Example 4-43.

**Example 4-46** *Checking the StatefulSets PVCs and PVs, some columns are omitted for better readability*

```
[root@i7PFE7 OCP_install]# oc get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES
fs9110-pvc-filessystem	Bound	pvc-be213821-a3da-46d5-98a9-475e86427e1f	1Gi	RWO
fs9110-volume-filessystem-fs9110-statefulset-filessystem-0	Bound	pvc-86bc5740-6218-4d01-be78-e8e0d8b9bf53	2Gi	RWO
fs9110-volume-filessystem-fs9110-statefulset-filessystem-1	Bound	pvc-29e1788a-0a26-43b7-9034-c35ced964718	2Gi	RWO
fs9110-volume-filessystem-fs9110-statefulset-filessystem-2	Bound	pvc-deea50ac-931e-4d4a-a76f-68e7f9c3960c	2Gi	RWO

```
[root@i7PFE7 OCP_install]# oc get pv
```

NAME	CAPACITY	ACCESS MODES	CLAIM
pvc-be213821-a3da-46d5-98a9-475e86427e1f	1Gi	RWO	fs9110/fs9110-pvc-filessystem
pvc-86bc5740-6218-4d01-be78-e8e0d8b9bf53	2Gi	RWO	fs9110/fs9110-volume-filessystem-fs9110-statefulset-filessystem-0
pvc-29e1788a-0a26-43b7-9034-c35ced964718	2Gi	RWO	fs9110/fs9110-volume-filessystem-fs9110-statefulset-filessystem-1
pvc-deea50ac-931e-4d4a-a76f-68e7f9c3960c	2Gi	RWO	fs9110/fs9110-volume-filessystem-fs9110-statefulset-filessystem-2

The storage system volume name is preceded with *ocp-ats* as defined in the storage class used and shown in Figure 4-15. The three volumes of the StatefulSet are mapped to a host.

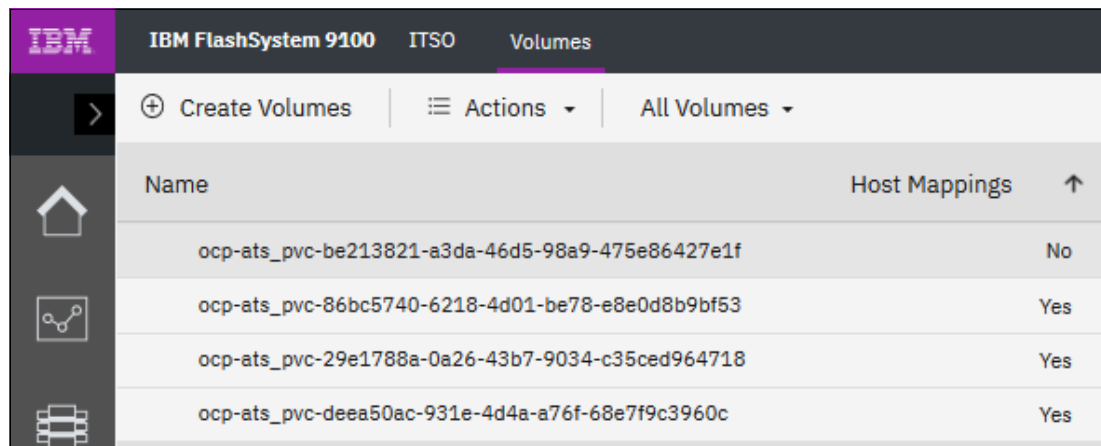


Figure 4-17 FlashSystem 9110 volumes created by the StatefulSet

The number of pods of a StatefulSet can be scaled up and down. The example StatefulSet was created with three replicas. When scaling down to two replicas, one pod is ended. When the number is scaled up again, the previously used volume is reused. Before scaling down, some data is written to the pods.

Example 4-47 on page 84 shows writing to the pods and scaling down.

**Example 4-47** *Scaling replicas down*

```
# write data to the pods
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filessystem-0 -- touch /data/fs9110-statefulset-filessystem-0
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filessystem-1 -- touch /data/fs9110-statefulset-filessystem-1
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filessystem-2 -- touch /data/fs9110-statefulset-filessystem-2

# check the data
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filessystem-0 -- ls -l /data
-rw-r--r--. 1 1000570000 1000570000 0 Nov 25 18:13 fs9110-statefulset-filessystem-0
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filessystem-1 -- ls -l /data
-rw-rw-rw-. 1 1000570000 1000570000 0 Nov 25 18:13 fs9110-statefulset-filessystem-1
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filessystem-2 -- ls -l /data
-rw-r--r--. 1 1000570000 1000570000 0 Nov 25 18:14 fs9110-statefulset-filessystem-2
```



```
# Set the number of replicas to two:
[root@i7PFE7 ~]# oc edit statefulset fs9110-statefulset-filesystem
```

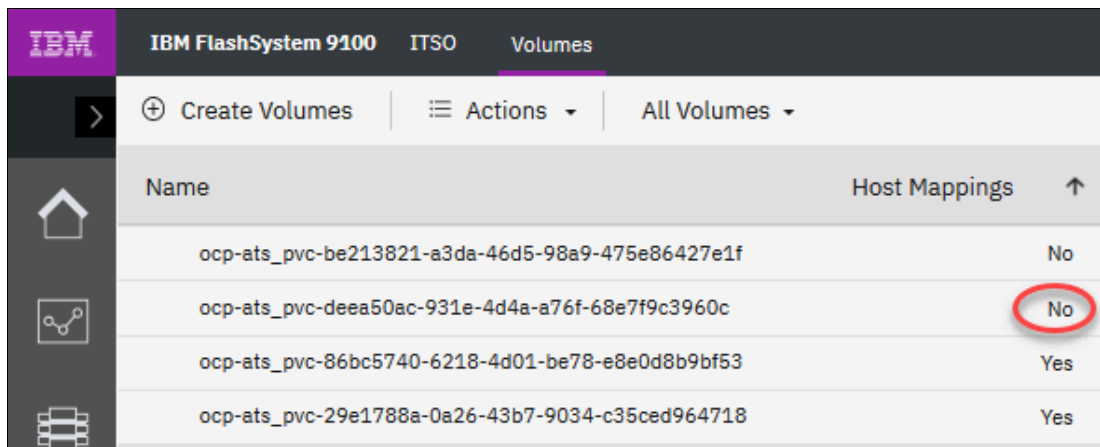
```
...
spec:
  podManagementPolicy: OrderedReady
  replicas: 2
...
```

```
statefulset.apps/fs9110-statefulset-filesystem edited
```

```
# check on pod changes
```

```
[root@i7PFE7 OCP_install]# oc get pod
NAME                                READY   STATUS    RESTARTS   AGE
fs9110-statefulset-filesystem-0    1/1    Running   0          26m
fs9110-statefulset-filesystem-1    1/1    Running   0          25m
fs9110-statefulset-filesystem-2    1/1    Terminating 0          25m
```

One pod was ended. One volume of the three StatefulSet volumes is now unmapped from the host, as shown in Figure 4-18.



Name	Host Mappings
ocp-ats_pvc-be213821-a3da-46d5-98a9-475e86427e1f	No
ocp-ats_pvc-deea50ac-931e-4d4a-a76f-68e7f9c3960c	No
ocp-ats_pvc-86bc5740-6218-4d01-be78-e8e0d8b9bf53	Yes
ocp-ats_pvc-29e1788a-0a26-43b7-9034-c35ced964718	Yes

Figure 4-18 FlashSystem 9110 one StatefulSet volume is now unmapped from the host.

When scaling up the StatefulSet replicas, this unmapped volume is reused. The node that is using this volume is not deterministic and therefore, the volume must be unmapped and are mapped to the worker node, which starts the next StatefulSet pod.

IBM Spectrum Virtualize-based storage systems have `vdiskprotection` enabled by default for pools that are created with software version 8.3.1 and up. The `vdiskprotection` prevents unmapping of the volume from the host if it experienced recent I/O activity and the volume is still mapped to the host.

The StatefulSet requires a volume in an unmapped state; otherwise, the volume provisioning fails. Therefore, the `vdiskprotectionenabled` parameter of an IBM Spectrum Virtualize-based storage pool is set to `off` to assure correct unmapping by OpenShift.

**Note:** The pool with disabled VDisk protection disabled is now exposed to a higher risk on deletion errors.

The number of pods of a StatefulSet can be scaled up and down. Example 4-47 on page 84 shows scaling our StatefulSet's replicas down by one from three to two.

Example 4-48 shows scaling our StatefulSet's replicas up by three from two to five. The third replica now uses the previously used FlashSystem 9110 volume. We can verify this use by reviewing the file system of the third pod, as shown in Example 4-48.

*Example 4-48 Scaling replicas up*

---

```
# Set the number of replicas to five:
[root@i7PFE7 ~]# oc edit statefulset fs9110-statefulset-filesystem

...
spec:
  podManagementPolicy: OrderedReady
  replicas:5
...

statefulset.apps/fs9110-statefulset-filesystem edited

# check on pod changes
[root@i7PFE7 OCP_install]# oc get pod
NAME                                READY   STATUS    RESTARTS   AGE
fs9110-statefulset-filesystem-0    1/1     Running   0           61m
fs9110-statefulset-filesystem-1    1/1     Running   0           61m
fs9110-statefulset-filesystem-2    1/1     Running   0           64s
fs9110-statefulset-filesystem-3    1/1     Running   0           43s
fs9110-statefulset-filesystem-4    1/1     Running   0           28s

# check the data, look at the third pod reusing its previously generated data
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filesystem-0 -- ls -l /data
-rw-r--r--. 1 1000570000 1000570000 0 Nov 25 18:13 fs9110-statefulset-filesystem-0
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filesystem-1 -- ls -l /data
-rw-rw-rw-. 1 1000570000 1000570000 0 Nov 25 18:13 fs9110-statefulset-filesystem-1
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filesystem-2 -- ls -l /data
-rw-rw-r--. 1 1000570000 1000570000 0 Nov 25 18:14 fs9110-statefulset-filesystem-2
[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filesystem-3 -- ls -l /data

[root@i7PFE7 ~]# oc exec -it fs9110-statefulset-filesystem-3 -- ls -l /data
```

---

The re-mapped volume and the two other volumes are shown in the IBM FlashSystem 9110 GUI, as shown in Figure 4-19.

Name	Host Mappings	Capacity
ocp-ats_pvc-be213821-a3da-46d5-98a9-475e86427e1f	No	1.00 GiB
ocp-ats_pvc-86bc5740-6218-4d01-be78-e8e0d8b9bf53	Yes	2.00 GiB
ocp-ats_pvc-29e1788a-0a26-43b7-9034-c35ced964718	Yes	2.00 GiB
ocp-ats_pvc-deea50ac-931e-4d4a-a76f-68e7f9c3960c	Yes	2.00 GiB
ocp-ats_pvc-183b5d5f-b0cd-41f3-a61a-625a84d8559f	Yes	2.00 GiB
ocp-ats_pvc-db822699-8085-4f2e-88f6-90549b4ca898	Yes	2.00 GiB

Figure 4-19 FlashSystem 9110 one unmapped StatefulSet volume is now mapped again to the host

## 4.4 CSI and OpenShift on IBM Z

The *Red Hat OpenShift on IBM Z Installation Guide*, [REDP-5605](#), publication includes more information that is needed to create an OpenShift cluster that is based on z/VM® RHCOS guests.

To use the IBM block storage CSI driver on IBM Z, some extra settings and adjustments are required. We include more information here that is not included in the Installation Guide publication.

For more information about Linux on Z and LinuxONE, see this [IBM Documentation web page](#). Regarding the CSI driver, the sections on Device Drivers, Features and Commands for the Red Hat distributions are most valuable.

### 4.4.1 Enabling FCP adapters

After an OpenShift cluster is up and running on z/VM instances, one special hurdle must be overcome: The concept of how IBM mainframes deal with I/O devices.

For the IBM block storage CSI driver's purpose, we use the SCSI-over-FibreChannel device driver that implements the FC Protocol (FCP) over virtualized adapters. These devices are called *FCP devices* or *FCP adapters*. For more information, see this [IBM Documentation web page](#).

Administrators of z/VM Linux guests are familiar with the necessity of activating the devices that are configured for the z/VM guest in the operating system. The FCP devices are no exception. They appear on the Channel Command Word (CCW) bus, which describes the data structure that is used for addressing devices in IBM Z systems.

Administrators assign (attach) devices to the z/VM guests on their virtualized CCW bus, and the operating system on the guest can then address each device through its `bus-id` on the bus. These relationships are described next.

To use a device that is attached to the z/VM guest, it must be made available for the operating system specifically. This process occurs in two steps:

1. The `cio_ignore` kernel parameter, or running the `cio_ignore` command in the running Linux operating system, determines the z/VM guest devices that can be “seen” by the operating system.
2. The operating system can then enable a device by using the `chccwdev` command. For the Linux kernel, this process results in a hot-plug event, and the device driver is connected with device. An administrator of a “traditional” Linux installation on a z/VM guest can use specific configuration files that control the process of “un-ignoring” the guest devices, and their enablement for the kernel.

However, in the OpenShift context, where the nodes are running Red Hat CoreOS (RHCOS), the situation is different. All nodes run the same base operating system image, and they are only distinguishable by their IP address and host name. A consequence of this principle is that the platform does not provide any means of a traditional Configuration Management Database (CMDB).

Detailed information does not need to be maintained about every installed operating system instance from which individual nodes with individual software stacks are recreated. The `bus-id` information for the z/VM devices for each guest is a typical piece of information that is included in such a CMDB.

Next, we describe how to set up the FCP adapters on RHCOS on IBM Z nodes.

The node is configured to use a Direct Access Storage Device (DASD), which is a standard IBM Z disk device, as operating system disk. It has several other DASDs attached, which are not used and it includes two attached FCP adapters that provide access to our CSI storage backend (which is a DS8000 family member). Some volumes are assigned for the node, but we do not use the DASDs.

Our first step is to make the bus IDs of the FCP adapters visible to RHCOS. We pass their bus IDs as kernel parameters for the worker nodes. This process is done in the worker’s `param` file, which is applied during node installation. The `cio_ignore` line in Example 4-49 is important. On our z/VM guest, we have two FCP adapters available: one with id `0.0.1923`, the other with `0.0.1963`; therefore, we “un-ignore” them.

*Example 4-49 A worker’s kernel parameters file*

---

```
$ cat worker-0.param
```

```
rd.neednet=1 coreos.inst=yes
coreos.inst.install_dev=dasda
coreos.inst.image_url=http://172.18.142.30:8088/coreos-s390/rhcos-4.4.9-s390x-dasd
.s390x.raw.gz
coreos.inst.ignition_url=http://172.18.142.30:8088/clusters/zvm/worker.ign
ip=172.18.142.35::172.18.0.1:255.254.0.0:worker-0::none
nameserver=172.18.142.30
rd.znet=qeth,0.0.bdf0,0.0.bdf1,0.0.bdf2,layer2=1,portno=0
cio_ignore=all,!condev,!0.0.1923,!0.0.1963,!0.0.6689,!0.0.668a
rd.dasd=0.0.6688
```

---

After the worker node is installed with this param file, we `ssh` to the node as user `core` and review our available devices, as shown in Example 4-50.

*Example 4-50 Available devices on z/VM guest*

```
[core@worker-0 ~]$ lsdev
TYPE          ID                      ON  PERS  NAMES
dasd-eckd    0.0.6688                yes no   dasda
dasd-eckd    0.0.6689                no  no
dasd-eckd    0.0.668a                no  no
zfcplib      0.0.1923                no  no
zfcplib      0.0.1963                no  no
qeth         0.0.bdf0:0.0.bdf1:0.0.bdf2  yes no   encbdf0
generic-ccw  0.0.0009                yes no
```

We can see our FCP adapters on 0.0.1923 and 0.0.1963, but they are still offline. The DASD on 0.0.6688 is the system disk. Other DASDs also are configured on the system, but they are not activated. To see all available devices, you can also review `/sys/bus/ccw/devices`, as shown in Example 4-51.

*Example 4-51 Available devices listing in /sys/bus/ccw/devices*

```
[core@worker-0 devices]$ ls -l
total 0
lrwxrwxrwx. 1 root root 0 Nov  4 09:16 0.0.0009 -> ../../../../devices/css0/0.0.0007/0.0.0009
lrwxrwxrwx. 1 root root 0 Nov  4 09:16 0.0.1923 -> ../../../../devices/css0/0.0.0004/0.0.1923
lrwxrwxrwx. 1 root root 0 Nov  4 09:16 0.0.1963 -> ../../../../devices/css0/0.0.0005/0.0.1963
lrwxrwxrwx. 1 root root 0 Nov  4 09:16 0.0.6688 -> ../../../../devices/css0/0.0.0000/0.0.6688
lrwxrwxrwx. 1 root root 0 Nov  4 09:16 0.0.6689 -> ../../../../devices/css0/0.0.0001/0.0.6689
lrwxrwxrwx. 1 root root 0 Nov  4 09:16 0.0.668a -> ../../../../devices/css0/0.0.0002/0.0.668a
lrwxrwxrwx. 1 root root 0 Nov  4 09:16 0.0.bdf0 -> ../../../../devices/css0/0.0.0008/0.0.bdf0
lrwxrwxrwx. 1 root root 0 Nov  4 09:16 0.0.bdf1 -> ../../../../devices/css0/0.0.0009/0.0.bdf1
lrwxrwxrwx. 1 root root 0 Nov  4 09:16 0.0.bdf2 -> ../../../../devices/css0/0.0.000a/0.0.bdf2
```

We can use the `chccwdev` command, or we can write to the `online` file in the driver's `/sys/bus/ccw/devices` directory to activate FCP devices temporarily, at first. Example 4-52 shows both possibilities.

*Example 4-52 Activation of FCP adapters (two variants)*

```
[core@worker-0 devices]$ echo 1 | sudo tee /sys/bus/ccw/devices/0.0.1923/online
1
[core@worker-0 devices]$ lsdev
TYPE          ID                      ON  PERS  NAMES
dasd-eckd    0.0.6688                yes no   dasda
dasd-eckd    0.0.6689                no  no
dasd-eckd    0.0.668a                no  no
zfcplib      0.0.1923                yes no
zfcplib      0.0.1963                no  no
zfcplib-lun  0.0.1923:0x500507630910d435:0x408240f100000000  yes no   sda sg0
zfcplib-lun  0.0.1923:0x500507630910d435:0x408340f100000000  yes no   sdb sg1
zfcplib-lun  0.0.1923:0x500507630910d435:0x408440f100000000  yes no   sdc sg2
zfcplib-lun  0.0.1923:0x500507630910d435:0x408540f100000000  yes no   sdd sg3
qeth         0.0.bdf0:0.0.bdf1:0.0.bdf2  yes no   encbdf0
generic-ccw  0.0.0009                yes no

[core@worker-0 devices]$ sudo chccwdev -e 0.0.1963
Setting device 0.0.1963 online
Done
```

```
[core@worker-0 devices]$ lszdev
```

TYPE	ID	ON	PERS	NAMES
dasd-eckd	0.0.6688	yes	no	dasda
dasd-eckd	0.0.6689	no	no	
dasd-eckd	0.0.668a	no	no	
zfcplib	0.0.1923	yes	no	
zfcplib	0.0.1963	yes	no	
<b>zfcplib</b>	<b>0.0.1923:0x500507630910d435:0x408240f100000000</b>	<b>yes</b>	<b>no</b>	<b>sda sg0</b>
zfcplib	0.0.1923:0x500507630910d435:0x408340f100000000	yes	no	sdb sg1
zfcplib	0.0.1923:0x500507630910d435:0x408440f100000000	yes	no	sdg sg2
zfcplib	0.0.1923:0x500507630910d435:0x408540f100000000	yes	no	sdd sg3
zfcplib	0.0.1963:0x500507630914d435:0x408240f100000000	yes	no	sde sg4
zfcplib	0.0.1963:0x500507630914d435:0x408340f100000000	yes	no	sdf sg5
zfcplib	0.0.1963:0x500507630914d435:0x408440f100000000	yes	no	sdg sg6
zfcplib	0.0.1963:0x500507630914d435:0x408540f100000000	yes	no	sdh sg7
qeth	0.0.bdf0:0.0.bdf1:0.0.bdf2	yes	no	encbdf0
generic-ccw	0.0.0009	yes	no	

After the FCP adapters are enabled, we now can see all pre-configured logical units identified with their Logical Unit Number (LUN)<sup>1</sup>, which is reachable through them. Reviewing the highlighted line in the Example 4-52 on page 89, we can see this information for a specific LUN in the second column. The colon-separated values are the adapter's bus-id, its worldwide port name (WWPN), and the LUN.

However, what was done so far is not persistent. After the node reboots, it has the FCP adapter available, but it is not active. We make the activation permanent by creating a `/etc/zfcplib.conf` file.

As of this writing, the implementation requires fully specified lines with the CCW bus-id, WWPN, and LUN. We are interested in activating the adapter in general, regardless of the existing LUNs because they are created dynamically later. Therefore, we use a dummy LUN entry: the well-known REPORT LUN ID, as shown in Example 4-53.

*Example 4-53 Persistent activation of FCP adapters using a dummy REPORT LUN entry*

```
[core@worker-0 ~]$ cat /etc/zfcplib.conf
0.0.1923 0x500507630910d435 0xc101000000000000
0.0.1963 0x500507630914d435 0xc101000000000000
```

Although our nodes survive reboots with these settings, we must reconfigure them if they are reinstalled.

It is possible (but out of scope of this book) to supply a MachineConfig object that sets all available adapters online, regardless of their bus-id. The MachineConfig's ignition file definition creates a systemd unit that activates all visible adapters (cio\_ignore kernel parameter). A second unit that handles FCP devices creates `/etc/zfcplib.conf`, as shown Example 4-53. This second unit is triggered only if `/etc/zfcplib.conf` does not exist.

<sup>1</sup> Following common practice, we use the LUN acronym as synonym for the logical unit itself.

## 4.5 CSI deployment on Z by using the command-line interface

The IBM block storage CSI driver installation is described in its respective users guide. Here, we want to give some examples and helpful tips for Kubernetes novices.

The deployment of the driver can be structured into four main steps and one optional step:

1. Deploying of the IBM CSI Operator. The operator can be seen as the launch pad for the driver installation.
2. Creating an IBMBlockCSI resource. The IBMBlockCSI object tells the IBM CSI Operator how a release of the IBM CSI driver is to be installed in the cluster.
3. Creating a StorageClass for CSI provisioned volumes. Other storage parameters, such as the storage pool from which to create volumes, or a reference to the secret to use for accessing the backend, is tied together in this StorageClass.
4. Creating a secret holding for the access information for the storage backend. This secret holds the management address, user name, and password that are needed to access the storage backend.
5. If VolumeSnapshots are used, a VolumeSnapshot Class and optionally a other secret is needed. This requirement is similar to the StorageClass and describes how to use the storage backend for VolumeSnapshots.

### 4.5.1 Deploying the operator

Common practice in Kubernetes and OpenShift is to create Kubernetes resources from yaml files. First step is to download the yaml manifest for the IBM CSI operator. For our example, we follow the instructions that are available are found at this [IBM Documentation web page](#).

Examples were taken on a Linux bastion host with the OpenShift Client (oc) installed and working that uses a Kubernetes admin user.

After downloading the operator manifest from GitHub to `ibm-block-csi-operator.yaml`, we noticed that some namespace objects in the manifest must be changed to the namespace we choose for the operator and driver components - `ibm-csi-operator`. This change affects the namespace for the deployment, the ServiceAccount, and the reference to the ServiceAccount in the ClusterRoleBinding, (see Example 4-54).

*Example 4-54 Namespace changes (lines are omitted for brevity)*

---

```
...  
  
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: ibm-block-csi-operator  
  namespace: ibm-csi-operator  
  labels:  
    product: ibm-block-csi-driver  
    csi: ibm  
    app.kubernetes.io/name: ibm-block-csi-operator  
    app.kubernetes.io/instance: ibm-block-csi-operator  
    app.kubernetes.io/managed-by: ibm-block-csi-operator  
  
...
```

```

subjects:
- kind: ServiceAccount
  name: ibm-block-csi-operator
  namespace: ibm-csi-operator
roleRef:
  kind: ClusterRole
  name: ibm-block-csi-operator
  apiGroup: rbac.authorization.k8s.io

...

apiVersion: v1
kind: ServiceAccount
metadata:
  name: ibm-block-csi-operator
  namespace: ibm-csi-operator
  labels:
    product: ibm-block-csi-driver
    csi: ibm

```

With these updates, the IBM CSI operator is deployed by applying the manifest as the User Guide tells us: `oc apply -f ibm-block-csi-operator.yaml`.

After a few seconds, we see the specified deployment and a ReplicaSet are created; then, the respective operator pod comes up (see Example 4-55).

*Example 4-55 Verifying successful deployment*

---

```

$ oc get all --selector=app.kubernetes.io/instance=ibm-block-csi-operator
NAME                                                    READY   STATUS    RESTARTS   AGE
pod/ibm-block-csi-operator-bdfb89bdd-56wg4            1/1     Running   0           8h

NAME                                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ibm-block-csi-operator                 1/1     1             1           29d

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/ibm-block-csi-operator-bdfb89bdd      1         1         1       29d
$ oc get crd ibmblockcsis.csi.ibm.com
NAME                               CREATED AT
ibmblockcsis.csi.ibm.com          2020-10-07T09:03:23Z

```

---

## 4.5.2 Deploying the driver

The operator we installed is responsible for the deployment of the driver throughout the cluster. As described in 3.1.8, “Extension points in Kubernetes: The operator pattern” on page 34, the operator is implementing a controller for the IBMBlockCSI CustomResource (CR) type. The CustomResourceDefinition (CRD) for that type of resources is part of the application manifest. Therefore, to install the driver, we must create only a respective IBMBlockCSI CR in the cluster. The operator installs the driver pieces to where they belong.

As the User Guide explains, we must download the correct version of the CR. Confirm that you downloaded the correct version. Deploy the driver by using `oc apply -f csi.ibm.com_v1_ibmblockcsi_cr_amd64.yaml`.



It takes some time until all parts and pieces are installed. However, after the deployment completes, the situation is similar to what is shown in Example 4-56.

*Example 4-56 Check successful driver deployment*

```

$ oc get all --selector=app.kubernetes.io/instance=ibm-block-csi
NAME                                READY   STATUS    RESTARTS   AGE
pod/ibm-block-csi-controller-0      5/5     Running   0           8h
pod/ibm-block-csi-node-97qzw        3/3     Running   0           8h
pod/ibm-block-csi-node-gj9lk        3/3     Running   0           30h
pod/ibm-block-csi-node-jwqds        3/3     Running   5           30h

NAME                                DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/ibm-block-csi-node   3         3         3       3         3
<none>          30h

NAME                                READY   AGE
statefulset.apps/ibm-block-csi-controller 1/1     30h

```

A StatefulSet for the controller part and a DaemonSet for the node parts are available. Check that all pods are in status Running, and that they do not accumulate restarts. If the latter is the case, something is wrong with the deployment. For more information about debugging, see Chapter 5, “Maintenance and troubleshooting” on page 113.

### 4.5.3 Configuring the storage backend

To provision storage on the backends, our configuration must answer the following questions for the platform:

- ▶ Which backend should be used to provision storage?
- ▶ How can the backend be accessed?

To answer these questions, we must create a StorageClass that ties the required information together. It contains a reference to the provisioner to use the IBM block storage CSI driver, and a reference to the access information for the backend; that is, a Secret object.

The yaml file that is shown in Example 4-57 on page 94 specifies a StorageClass for our purposes. Because we want to access a DS8000 family member, the ds8k naming is used.

#### Example 4-57 StorageClass specification

---

```
$ cat ds8k-storage-class.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ds8k-csi
parameters:
  SpaceEfficiency: thin
  csi.storage.k8s.io/controller-publish-secret-name: ds8k-secret
  csi.storage.k8s.io/controller-publish-secret-namespace: ibm-csi-operator
  csi.storage.k8s.io/provisioner-secret-name: ds8k-secret
  csi.storage.k8s.io/provisioner-secret-namespace: ibm-csi-operator
  pool: P4
provisioner: block.csi.ibm.com
reclaimPolicy: Delete
volumeBindingMode: Immediate
```

---

The answers to our questions can be found in the following file:

- ▶ The provisioner is `block.csi.ibm.com`.
- ▶ Access to the backend for the controller-publish service is stored in Secret `ds8k-secret` in namespace `ibm-csi-operator`.
- ▶ Access to the backend for the provisioning service is stored in the same Secret.

Again, a simple `oc apply -f ds8k-storage-class.yaml` enables the platform to create the respective StorageClass. The remaining process creates the Secret that we specified. This process can be done on the command line, as shown in Example 4-58.

#### Example 4-58 Creating Secret

---

```
$ oc create secret generic ds8k-secret \
  --from-literal=management_address=back-end-address(es) \
  --from-literal=username=back-end-username \
  --from-literal=password=back-end-password \
  --dry-run -o yaml > secret.yaml
$ oc apply -f secret.yaml
```

---

With these items set up, the driver is ready to dynamically provision storage on the back-end, and make it available for the application Pods throughout the cluster.

## 4.6 Installing the CSI driver by using the OpenShift web console on x86

In this section, we provide an example deployment and installation of the IBM block storage CSI driver that uses the OpenShift web console on x86 architecture. For more information about installing the IBM block storage CSI driver, see the following resources:

- ▶ [GitHub](#)
- ▶ [IBM Documentation](#)

The demonstration environment that used in this section is based on the architecture shown that is in Figure 4-20.

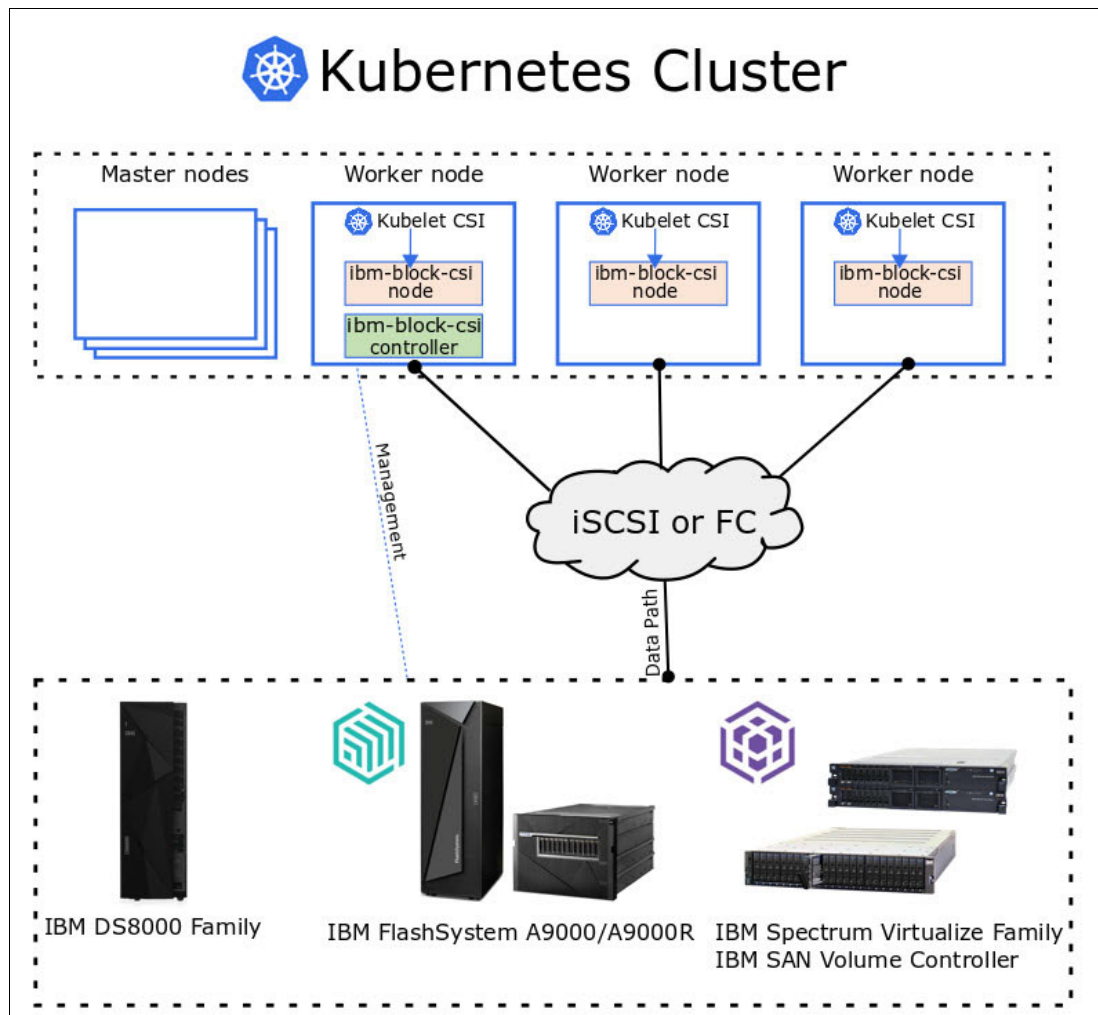


Figure 4-20 OpenShift Cluster architecture overview

The master nodes, also known as the control plane, are configured unschedulable. That is, the `masterSchedulable` setting in the scheduler's configuration of the OpenShift cluster is set to `false`. The command that is shown in Example 4-59 configures this setting.

*Example 4-59 Setting control plane to unschedulable*

```
oc patch schedulers.config.OpenShift.io cluster -p  
'{"spec":{"mastersSchedulable":false}}' --type=merge
```

The decision to set the control plane as unschedulable is a best practice for performance and security reasons. In our demo environment, it also allows us to showcase the decision that must be made before configuring the IBM block storage CSI driver.

We show a bare metal OpenShift cluster that is attached to the storage subsystems with FC adapters. Therefore, all nodes that have the IBM block storage CSI driver pods that are running on them need a physical adapter installed with connectivity to the storage subsystem.

The use of an iSCSI connection to the storage subsystem eliminates the adapter requirement, but it imposes the same considerations as for FC with regard to throughput and load.

You can run the command that is shown in Example 4-60 to interactively configure the schedulability of the master nodes.

*Example 4-60 Interactive configuration of control plane schedulability*

---

```
oc edit schedulers.config.OpenShift.io cluster
```

---

All worker nodes are schedulable and can have an IBM block storage CSI driver running on them.

Even with the use of FC communication to the storage backend an IP connectivity is needed for the communication between the OpenShift cluster and the respective storage subsystems. Administrative tasks, such as LUN creation and LUN masking, are done by using IP. Do not forget to provide both communication channels, IP and FC, if you plan to use the IBM block storage CSI driver.

## 4.6.1 Fullfilling installation prerequisites

In comparison to the command-line installation method or the deployment on other platforms, the installation of the IBM block storage CSI driver on x86 platform needs only little preparations and customizations. Opening firewall ports or installing more packages is not required on the CoreOS systems.

### OpenShift project for the IBM block storage CSI driver deployment

We follow the OpenShift best practices and create a dedicated project for the deployment of the IBM block storage CSI driver. As example project name `ibm-block-csi-driver` is chosen. A new project can be created with the CLI, as shown in Example 4-61.

*Example 4-61 Creating a new OpenShift project with the CLI*

---

```
[root@ocp4-helpnode ~]# oc new-project ibm-block-csi-driver
Now using project "ibm-block-csi-driver" on server
"https://api.openshift.sle.kelsterbach.de.ibm.com:6443".
```

---

You can add applications to this project with the `'new-app'` command. For example, try:

```
oc new-app ruby~https://github.com/sclorg/ruby-ex.git
```

to build a new example application in Python. Or use `kubectl` to deploy a simple Kubernetes application:

```
kubectl create deployment hello-node
--image=gcr.io/hello-minikube-zero-install/hello-node
```

---

As an alternative to the CLI way of creating an OpenShift project, the GUI can be used as shown in Figure 4-21.

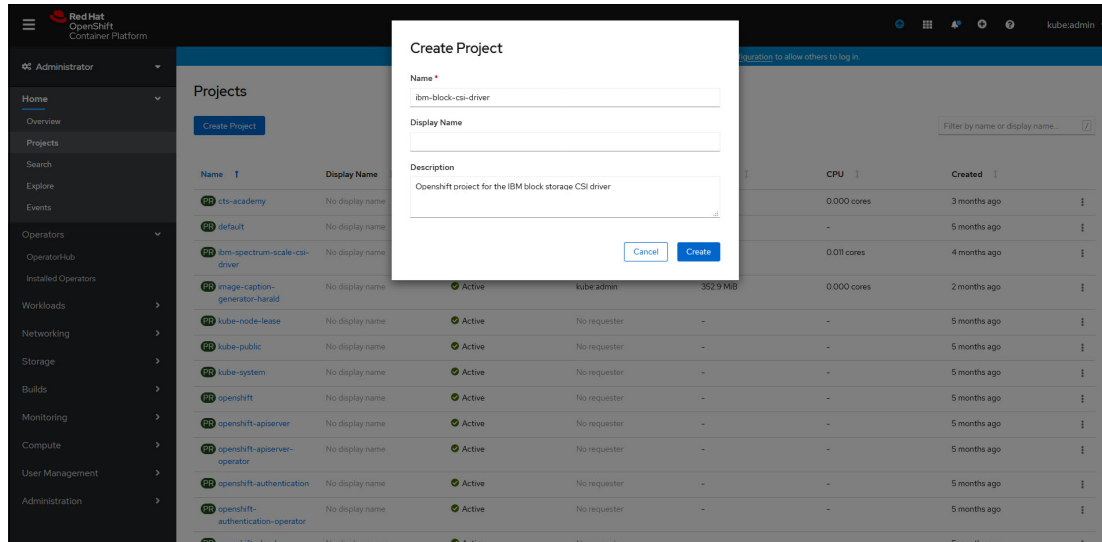


Figure 4-21 Creating an OpenShift project with the GUI

## Preparing CoreOS for Fibre Channel and iSCSI configurations

One of the key features of CoreOS is the controlled immutability. Management is performed remotely from the OpenShift Container Platform cluster. When you set up your RHCOS machines, you can modify only a few system settings. This controlled immutability allows OpenShift Container Platform to store the latest state of RHCOS systems in the cluster so that it always can create machines and perform updates that are based on the latest RHCOS configurations.

To enable the storage system support and suitable multipathing configuration, we use the mechanism of a machine configuration. The yaml file that is shown in Example 4-62 provides the necessary changes to the OpenShift cluster to introduce the needed storage subsystem configuration for multipathing. It also enables the multipathing daemon, and optionally enables and starts the iSCSI subsystem.

Example 4-62 *99-ibm-attach.yaml*

```

apiVersion: machineconfiguration.OpenShift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.OpenShift.io/role: worker
  name: 99-ibm-attach
spec:
  config:
    ignition:
      version: 2.2.0
    storage:
      files:
        - path: /etc/multipath.conf
          mode: 384
          filesystem: root
          contents:

```





- After these preparations are complete, the installation of the IBM block storage CSI driver can be started. In the OperatorHub, choose **IBM block storage CSI driver** for installation, as shown in Figure 4-24.

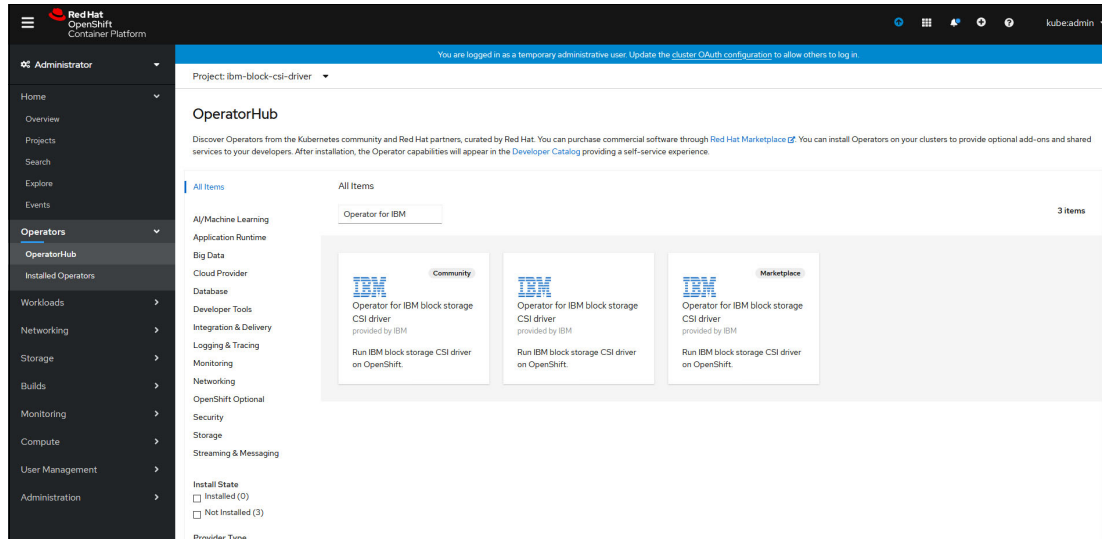


Figure 4-24 Choosing the IBM block CSI driver

At the time of the writing, the code that is delivered through the three individual items is the same. Expect a diversified offering in the future to allow for up-stream access to the community version of the code or similar. For this example, we choose the middle icon for installation.

- Click the middle icon and then, verify the information that is displayed (see Figure 4-25).

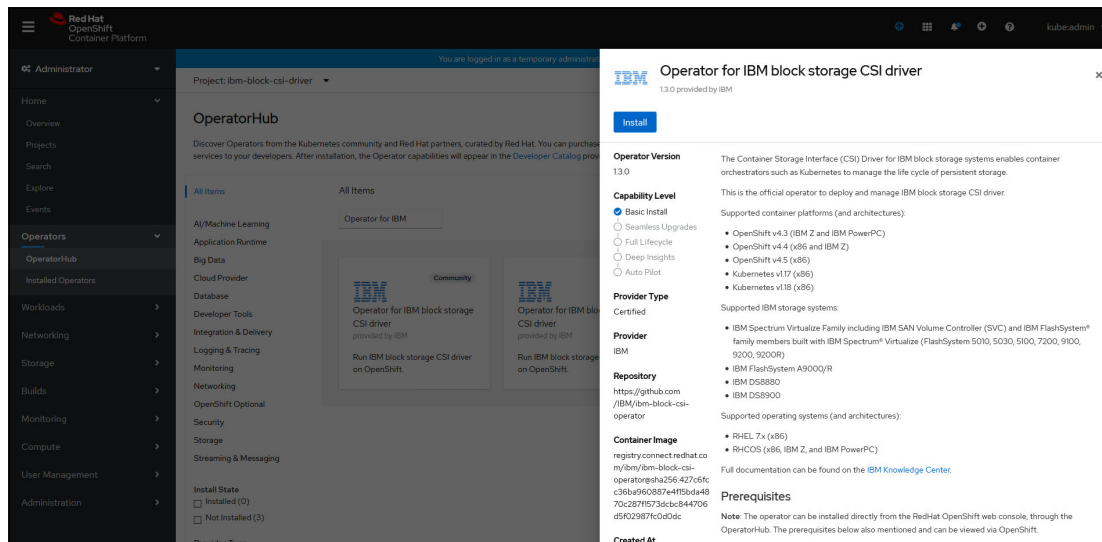


Figure 4-25 Installing the IBM block CSI driver with the web GUI



- Click **Install** to finalize and start the installation. The next window allows you to choose an Installation Mode, a target Namespace, an Update Channel, and an Approval Strategy, as shown in Figure 4-26.

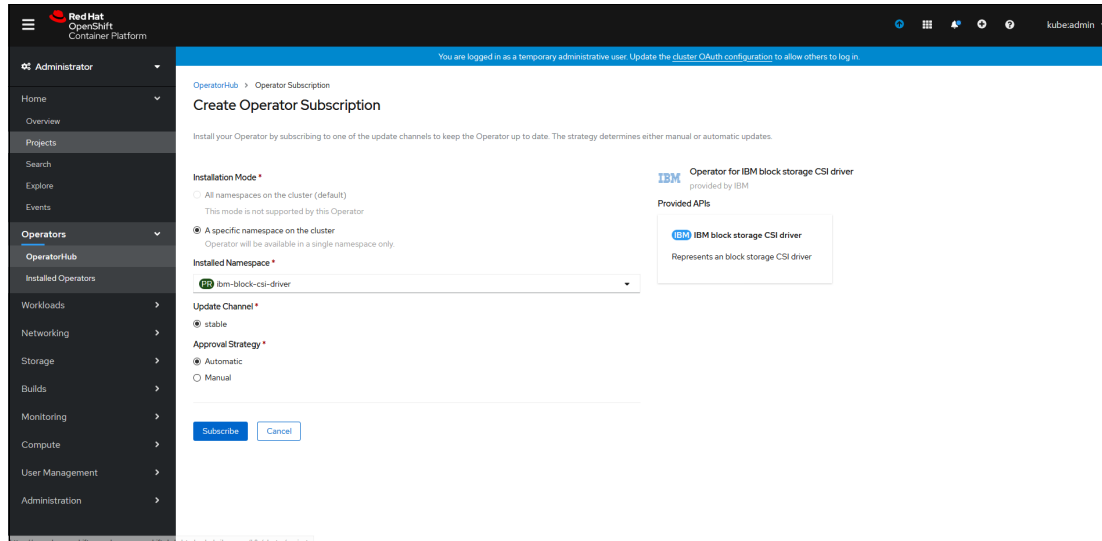


Figure 4-26 Creating an Operator Subscription in the web GUI

Following the preparations that are described this chapter, the acceptance of the default values leads to a successful IBM block CSI driver deployment.

- Click **Subscribe** to start the installation. The successful installation presents you with the overview that is shown in Figure 4-27.

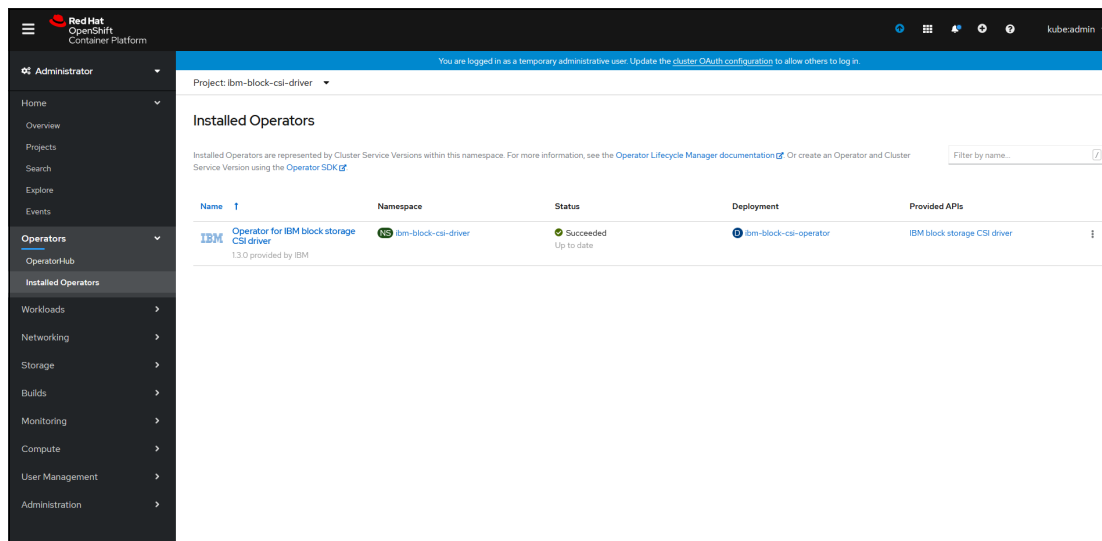


Figure 4-27 Successful IBM block CSI driver installation

- An instance of the IBM block storage CSI driver must be created. Click **Create IBMBlockCSI** in the Operator for IBM block storage CSI driver view, as shown in Figure 4-28. You might need to choose the **IBM block storage CSI driver** tab in the overview if you do not see it.

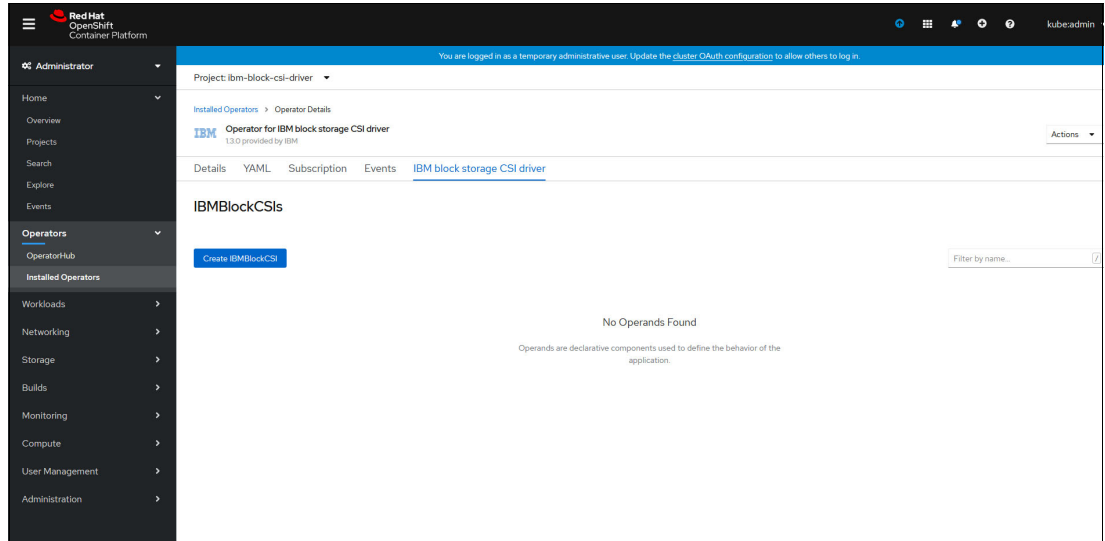


Figure 4-28 Creating an *IBMBlockCSI* instance

The Create *IBMBlockCSI* overview displays an overview of the instance to be created. The `yaml` is prepared and can be left unchanged (see Figure 4-29).

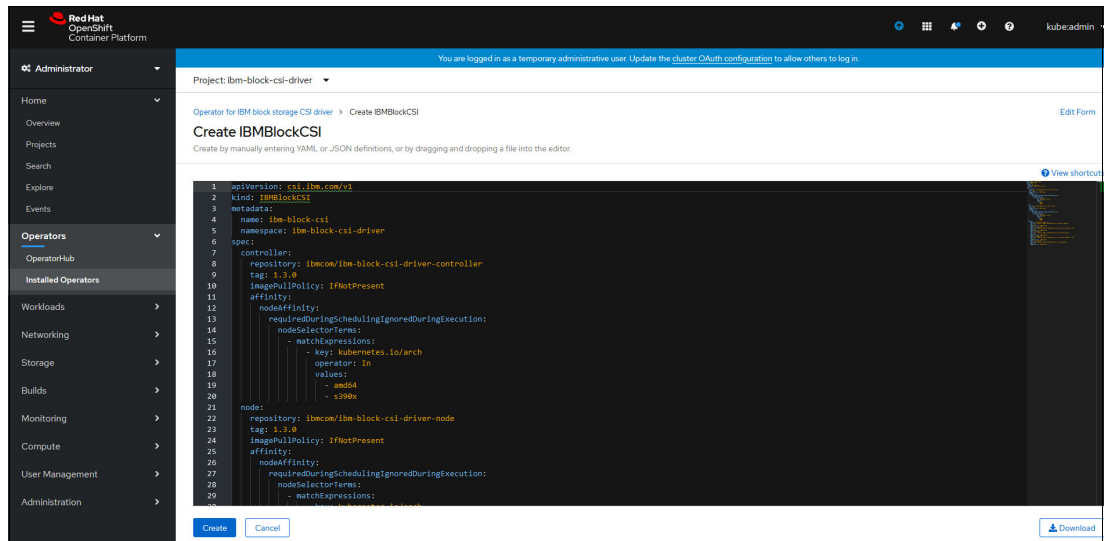


Figure 4-29 Creating an “*IBMBlockCSI*” instance

After some time, the process is finished and displays a successful result with the state as **Running**, as shown in Figure 4-30.

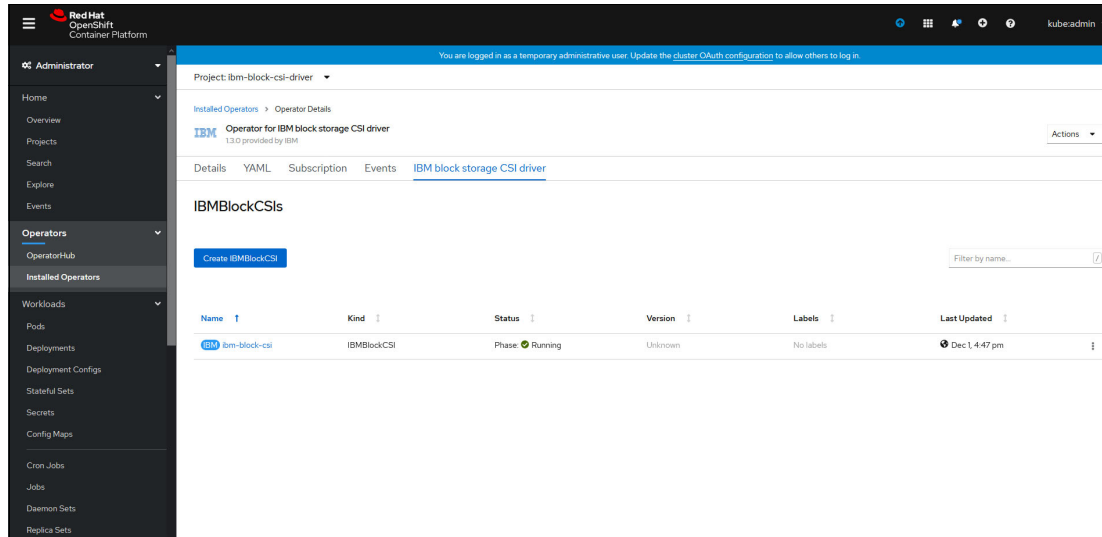


Figure 4-30 BM block CSI is running

The IBM block CSI installation is now complete. The next step is configuring the IBM block CSI. In the next section, we describe creating an array secret, storage class, and persistent volume claim (PVC).

### Creating an array secret

The array secret is used to define the storage system credentials, user name and password, and the storage subsystem address to the OpenShift cluster. The array secret is created with the GUI.

On the left side of the OpenShift web GUI, browse to **Workloads** → **Secrets** to start the array secret creation process, as shown in Figure 4-31. Click **Create** and choose **Key/Value Secret** to start the array secret creation process.

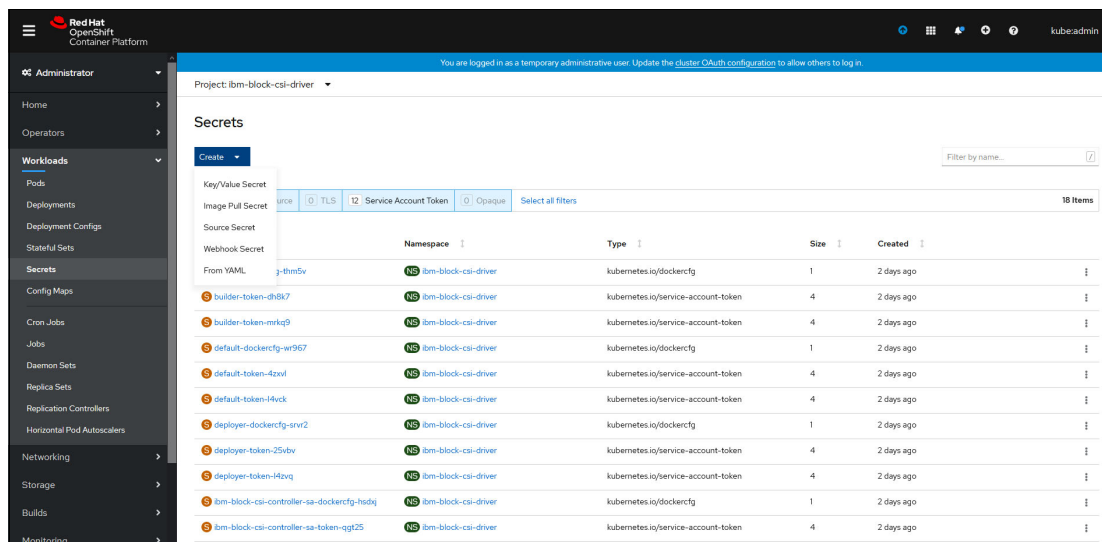


Figure 4-31 Secrets as part of the Workloads web GUI window

Enter the information for `management_address`, `username`, and `password` into the secret by entering the values and using **Add Key/Value** to create the other entries, as shown in Figure 4-32. Click **Create** to create the secret.

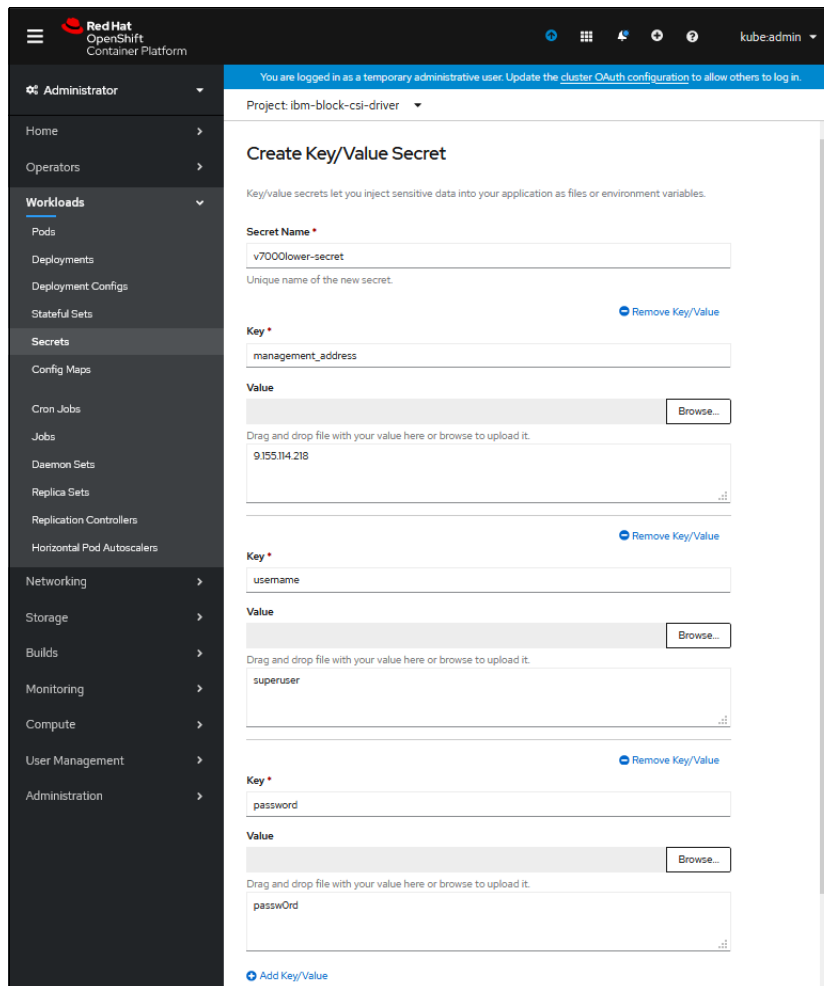


Figure 4-32 Creating the array secret

## Creating a storage class

The storage class is used to define the storage system pool name, secret reference, SpaceEfficiency type, and the `fstype` setting to the OpenShift cluster.

Complete the following steps:

1. Browse to **Storage** → **Storage Classes** of the OpenShift web GUI and click **Create Storage Class** to create a storage class. Click **Edit YAML** to manually enter the yaml file, as shown in Figure 4-33.

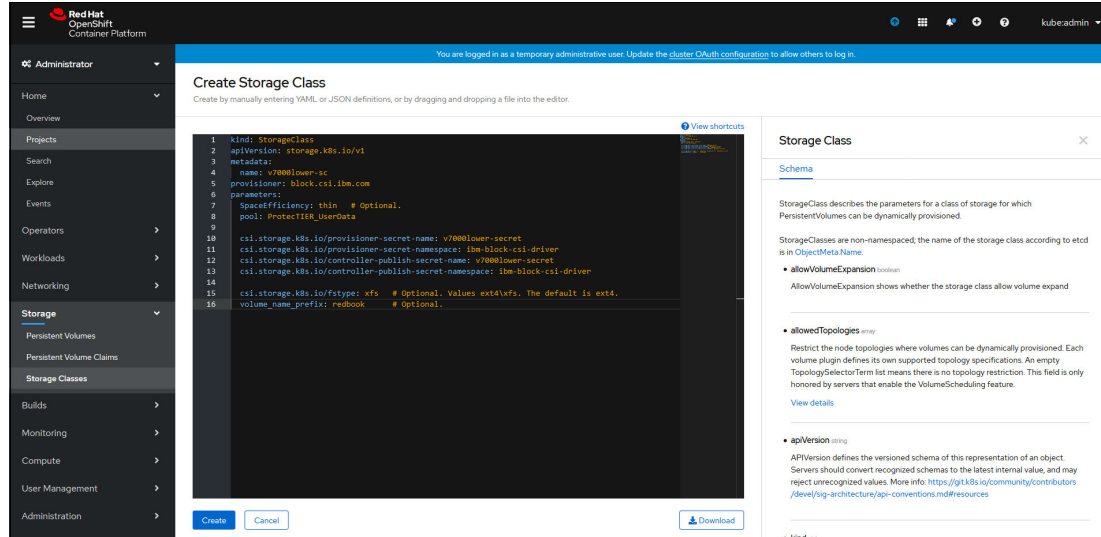


Figure 4-33 Creating a storage class with the OpenShift web GUI

2. Use the example YAML file that is shown in Example 4-64 as an example for the settings that are needed in the environment. The storage class name is set to a descriptive value that allows for easy identification of the storage in the OpenShift environment. The storage pool on the storage subsystem is selected and the respective secret and namespace values are selected.

The optional selection of the file system and the volume name prefix are used.

#### Example 4-64 Storage class example yaml file

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: v7000lower-sc
provisioner: block.csi.ibm.com
parameters:
  SpaceEfficiency: thin # Optional.
  pool: ProtecTIER_UserData

csi.storage.k8s.io/provisioner-secret-name: v7000lower-secret
csi.storage.k8s.io/provisioner-secret-namespace: ibm-block-csi-driver
csi.storage.k8s.io/controller-publish-secret-name: v7000lower-secret
csi.storage.k8s.io/controller-publish-secret-namespace: ibm-block-csi-driver

csi.storage.k8s.io/fstype: xfs # Optional. Values ext4\xfs. The default is
ext4.
volume_name_prefix: red # Optional.
```

3. Click **Create** to apply the yaml and create the storage class.

## Sample storage usage

With the creation of the storage class, the preparation for storage provisioning in the OpenShift cluster is finished. The storage can now be used.

The next example describes the process of accessing storage with the IBM block storage CSI driver.

To follow the process of storage allocation, a PVC is manually created and assigned to an example pod. If you configure your deployments to a storage class instead of a PVC, the process occurs automatically.

Complete the following steps:

1. Browse to **Home** → **Projects** and create an example project by clicking **Create Project**, as shown in Figure 4-34.

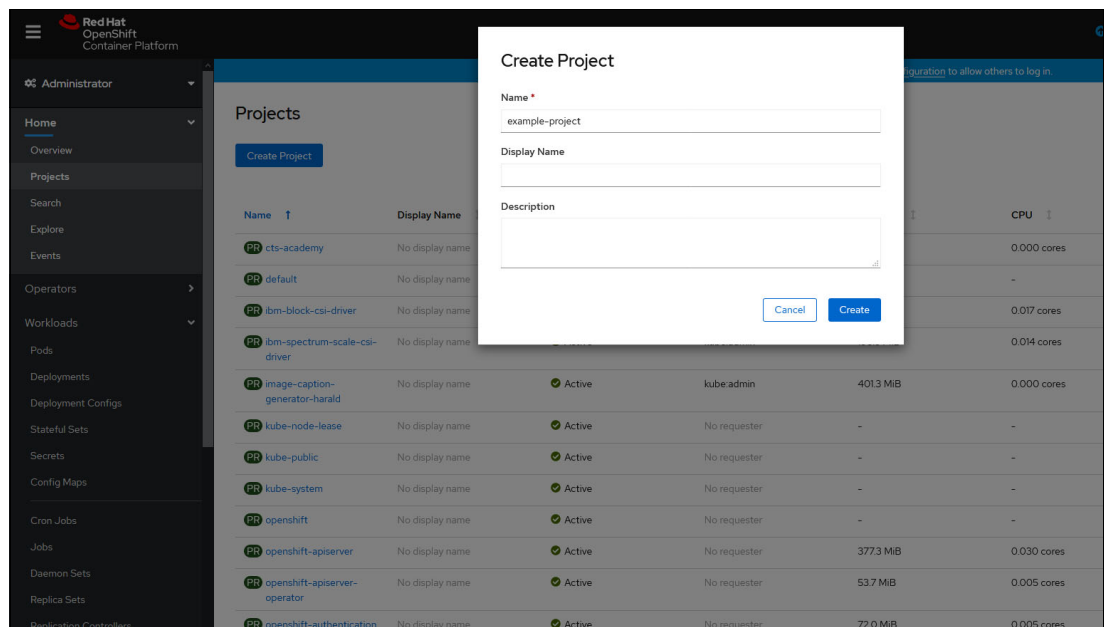


Figure 4-34 Create example project

- From within the project context of the newly created example-project, click **Storage** → **Persistent Volume Claims** and then, click **Create Persistent Volume Claim** to create a PVC. Enter example values as shown in Figure 4-35.

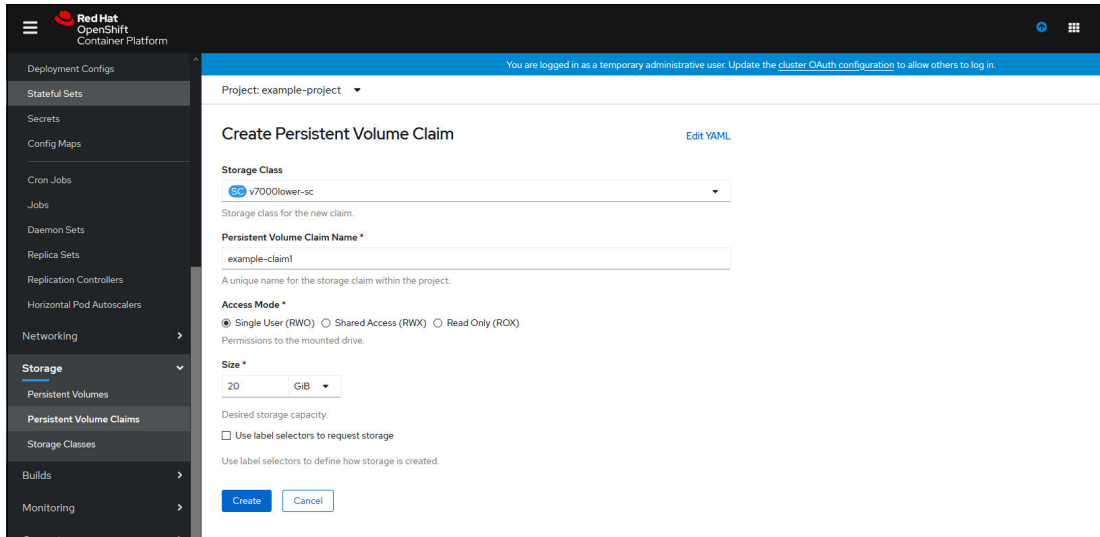


Figure 4-35 Create example PVC

- Cross-check the LUN creation process with the PVC overview. The OpenShift PVC overview shows that the two newly created PVCs are bound to a PV from the storage class (see Figure 4-36).

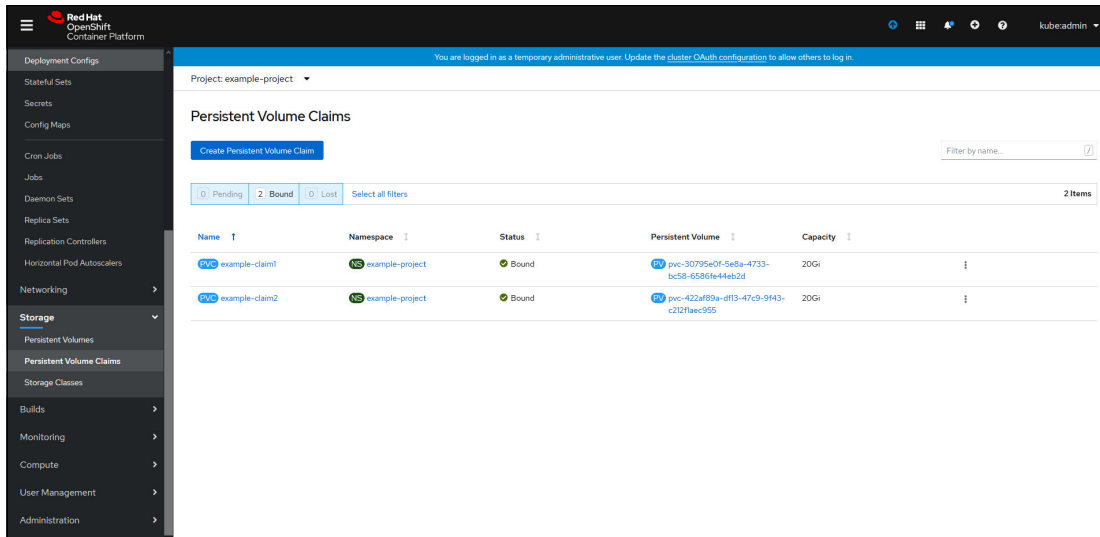


Figure 4-36 OpenShift PVC overview for the example-project

In comparison to the overview that is shown Figure 4-36, we see the LUNs that were created on the storage subsystem. The LUNs are not yet mapped to a host, as shown in Figure 4-37.

Name	State	Synchronized	Pool	UID	Host Mappings	Capacity
openshift data	✓ Online		MetaData	60050768028190358000000000000106	Yes	1.00 TiB
openshift images	✓ Online		UserData	60050768028190358000000000000107	Yes	4.00 TiB
red_pvc-422af89a-df13-47c9-9f43-c212faec955	✓ Online		UserData	60050768028190358000000000000110	No	20.00 GiB
red_pvc-30795e0f-5e6a-4713-bc58-6566fe44eb2d	✓ Online		UserData	6005076802819035800000000000010F	No	20.00 GiB
rhev0	✓ Online		UserData	60050768028190358000000000000108	Yes	2.00 TiB
rhev1	✓ Online		UserData	60050768028190358000000000000109	Yes	2.00 TiB

Figure 4-37 Storage subsystem overview with LUNs that are not yet mapped

- To use the storage and have the IBM block storage CSI map the LUNs to the hosts, a workload (for example, a pod), is needed. Create an example workload by using the YAML statement that is shown in Example 4-65.

*Example 4-65 Example pod to use storage*

```

apiVersion: v1
kind: Pod
metadata:
  name: my-site
spec:
  containers:
  - name: mysql
    image: mysql
    volumeMounts:
    - mountPath: /var/lib/mysql
      name: db-data
      subPath: mysql
  - name: php
    image: php
    volumeMounts:
    - mountPath: /var/www/html
      name: www-data
      subPath: html
  volumes:
  - name: db-data
    persistentVolumeClaim:
      claimName: example-claim1
  - name: www-data
    persistentVolumeClaim:
      claimName: example-claim2
  nodeSelector:
    csi: blockcsi

```



- Browse to **Home** → **Projects** and click your example project to ensure that the correct project context is entered. Then, click the **+** icon that is in the upper right corner of the window to enter and deploy the example (see Figure 4-38 on page 109). Paste the example or create your own and click **Create** to start the deployment.

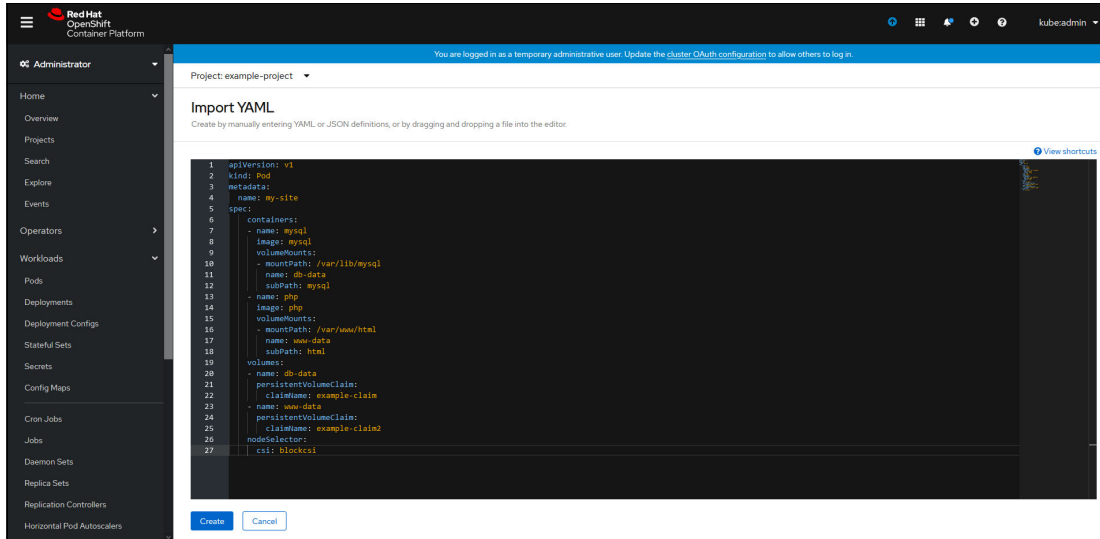


Figure 4-38 Deploy example workload; node selectors are used

**Note:** Our example is modified by using nodeSelector statements. Remove those statements or use them to your advantage in your own environment.

With the pod created, you can monitor the LUN state and see that the LUNs change from Host Mappings no to Host Mappings yes, as shown in Figure 4-39 on page 109.

Name	State	Synchronized	Pool	UID	Host Mappings	Capacity
openshift data	✓ Online		MetaData	60050768028100368000000000000106	Yes	1.00 TiB
openshift images	✓ Online		UserData	60050768028100368000000000000107	Yes	4.00 TiB
red_pvc-422af93a-df13-47c9-9f43-c2121faec955	✓ Online		UserData	60050768028100368000000000000110	Yes	20.00 GiB
red_pvc-30795e0f-5e8a-4733-bc58-8580fe44eb2d	✓ Online		UserData	6005076802810036800000000000010F	Yes	20.00 TiB
rhev0	✓ Online		UserData	60050768028100368000000000000108	Yes	2.00 TiB
rhev1	✓ Online		UserData	60050768028100368000000000000109	Yes	2.00 TiB

Figure 4-39 Host Mappings occurred

## 4.7 Updating the CSI driver

The CSI driver can be updated from a previous version by using one of the following methods:

- ▶ Use a subscription (as of this writing, x86 only)
- ▶ Uninstall the driver and then, install the newer version if no subscription exists

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

### 4.7.1 Updating the CSI driver by using a subscription

This example shows the x86 environment as discussed in 4.6, “Installing the CSI driver by using the OpenShift web console on x86” on page 95. You can use **Operators** → **Installed Operators** → **Operator for IBM block storage CSI driver** in the GUI to check and change the subscription, as shown Figure 4-40.

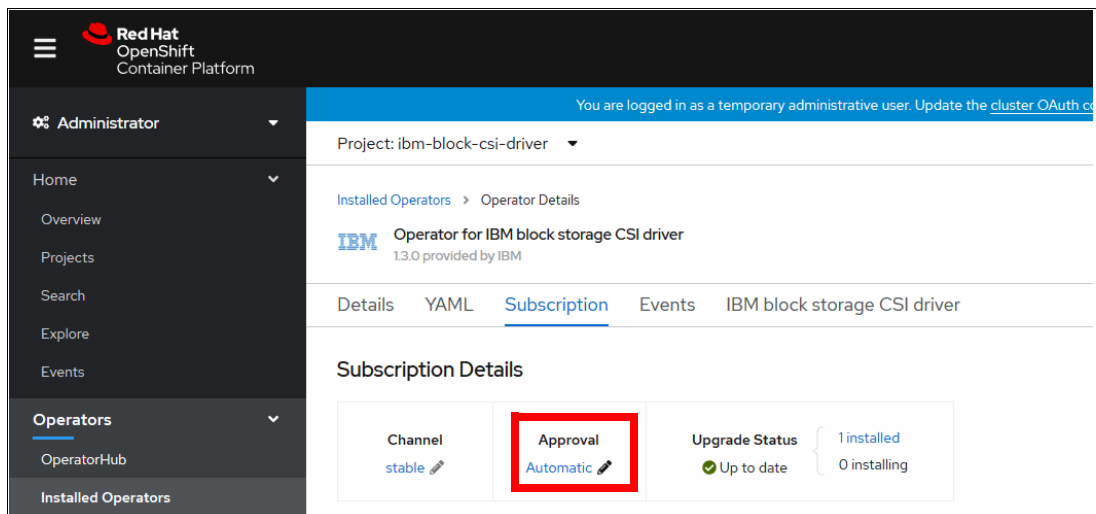


Figure 4-40 Operator Details CSI subscription, automatic mode

Determine whether the installed CSI driver includes an automatic or manual subscription type:

- ▶ If the subscription type is manual, you must confirm the update.
- ▶ If the subscription type is automatic, the CSI driver is automatically updated after it is available.

## 4.7.2 Updating the CSI driver manually

Determine whether the installed CSI driver does not use a subscription, as shown in Example 4-66. This example shows the IBM Power environment as discussed in 4.3, “CSI deployment on IBM Power by using the command-line interface” on page 72.

*Example 4-66 Check, if a subscription exists for the CSI driver*

---

```
[root@i7PFE7 ~]# # change to the CSI driver project
[root@i7PFE7 ~]# oc project ibm-block-csi
Now using project "ibm-block-csi" on server
"https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443".

[root@i7PFE7 ~]# # check on CSI driver subscription
[root@i7PFE7 ~]# oc get subscription
No resources found in ibm-block-csi namespace.
```

---

Example 4-66 on page 111 also shows that the current CSI driver does not have a subscription. Therefore, the upgrade must be done by uninstalling and reinstalling the new version. These processes are described next.

### Uninstalling the CSI driver

Complete the following steps to uninstall the CSI driver by using the installation files:

1. Delete the IBMBlockCSI custom resource:

```
kubectl delete -f csi.ibm.com_v1_ibmblockcsi_cr.yaml
```

2. Delete the operator:

```
kubectl delete -f ibm-block-csi-operator.yaml
```

### Installing the new CSI driver version

For more information about installing the CSI driver, see 4.3, “CSI deployment on IBM Power by using the command-line interface” on page 72.





## Maintenance and troubleshooting

In this chapter, we provide maintenance and troubleshooting tips and suggestions.

This chapter includes the following topics:

- ▶ 5.1, “General hints” on page 114
- ▶ 5.2, “Studying the good case” on page 114
- ▶ 5.3, “CSI operator issues” on page 117
- ▶ 5.4, “CSI driver CustomResource issues” on page 118
- ▶ 5.5, “CSI driver snapshot feature” on page 119
- ▶ 5.6, “CSI driver operation issues” on page 120
- ▶ 5.7, “Searching the CSI related log entries” on page 124
- ▶ 5.8, “Changing the CSI driver subscription by using the CLI” on page 125

## 5.1 General hints

Because Kubernetes follows a declarative paradigm, it might take some time until changes are picked up by the responsible controllers and propagated through the platform.

Consider the following points:

- ▶ All CSI driver parts are deployed through workload controllers. It is safe to delete pods that exhibit unusual behavior. Sometimes, this method is the simplest and most effective fix.
- ▶ Occasionally, timing issues or deadlocks can occur that also often can be solved easily by deleting single pods.
- ▶ If fixing an error does not yield immediate effects, controllers that are detecting errors on their managed entities put exponential time backoffs for retries. It can take a long time until a controller performs a reconciliation on erroring components. Deleting erroring components can speed up recovery.
- ▶ Configuration changes (for example, CSI secrets) are not necessarily detected by running pods. At times, they must be deleted and restarted to use their new configuration.
- ▶ Setting up logging is can be done. However, logs are available by using `oc logs` if a pod can start.
- ▶ Erroring pods can be inspected by using `oc describe`, which shows the latest events.
- ▶ It might be helpful to `oc exec` into running pods to get more information.
- ▶ Also, `oc debug` can help debug pods.

## 5.2 Studying the good case

Because it is difficult to anticipate every possible failure scenario, we demonstrate a “perfect” case from the command-line perspective as a guideline for where to dig deeper if anything goes wrong.

The starting point is a simple application: An image that waits for 10000 seconds and then exits and is redeployed. However, to use the CSI driver, we mount a file system at `/tmp/csi-volume`, which we provision on a DS8000 backend by using the IBM block storage CSI driver. We start with our application manifest `demo-deploy.yaml`, as shown in Example 5-1.

*Example 5-1 Simple application manifest*

---

```
$ cat demo-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: csi-demo-02
spec:
  replicas: 1
  selector:
    matchLabels:
      app: csi-demo-02
  template:
    metadata:
      labels:
        app: csi-demo-02
    spec:
```

```

containers:
- name: shell
  image: registry.access.redhat.com/ubi7/ubi:latest
  command:
  - "bin/bash"
  - "-c"
  - "sleep 10000"
  volumeMounts:
  - name: csi-vol
    mountPath: "/tmp/csi-vol"
volumes:
- name: csi-vol
  persistentVolumeClaim:
    claimName: csi-claim-002
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: csi-claim-002
spec:
  storageClassName: "ds8k-csi"
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi

```

---

This application manifest contains two objects: A Deployment that is called `csi-demo-002` for our application, and a `PersistentVolumeClaim csi-claim-002`, which refers to the `StorageClass` for our DS8000 backend. We deploy our application by using `oc apply -f demo-deploy.yaml` and wait for some time until we check the progress and success.

We start with the expected Deployment for our application as shown in Example 5-2.

*Example 5-2 Listing Deployment for the application*

```

$ oc get deployment.apps/csi-demo-02
NAME          READY  UP-TO-DATE  AVAILABLE  AGE
csi-demo-02  1/1    1            1           89m

```

---

The `READY`, `UP-TO-DATE` and `AVAILABLE` column show us that all required `ReplicaSets` are deployed, current, and that their pods are alive. We check the `ReplicaSet` and notice its name, which refers to the controlling Deployment, as shown in Example 5-3.

*Example 5-3 Listing Replicaset and pod for the application*

```

$ oc get all --selector=app=csi-demo-02
NAME                                     READY  STATUS    RESTARTS  AGE
pod/csi-demo-02-f6cc59d78-xtqw4        1/1    Running   1          4h6m

NAME                                     DESIRED  CURRENT  READY  AGE
replicaset.apps/csi-demo-02-f6cc59d78  1        1        1      4h6m

```

---

We see that the `ReplicaSet` wants one pod to be running, and this one pod is also `READY` to serve requests. It is in status `Running`, and we see it was restarted once.

However, that issue does not concern us because we know it end after 10000 seconds. Because it existed for 4 hours and 6 minutes, it exited and was then restarted by the controlling ReplicaSet after approximately 2 hours and 47 minutes, or 10000 seconds.

By seeing the pod in the Running state, we assume that the Persistent Volume Claim (PVC) for our pod is bound to an accessible Persistent Volume (PV). To be sure, we check the PVC and PV status, as shown in Example 5-4.

*Example 5-4 Listing PVC and PV*

```

$ oc get pvc csi-claim-002
NAME                STATUS    VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS    AGE
csi-claim-002    Bound    pvc-e1c0274c-6a20-428c-919e-4548f21a4b4f  1Gi        RWO
ds8k-csi        90m

$ oc get pv pvc-e1c0274c-6a20-428c-919e-4548f21a4b4f
NAME                CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM              STORAGECLASS  REASON    AGE
pvc-e1c0274c-6a20-428c-919e-4548f21a4b4f  1Gi        RWO          Delete            Bound
ibm-csi-operator/csi-claim-002  ds8k-csi                90m

```

In section 3.2.1, “Volume lifecycle” on page 37, we discussed the concept of VolumeAttachment objects. These objects track the relation between volumes and nodes where they are needed. Our PV must be mounted on the node where the pod was scheduled. Reviewing all VolumeAttachments gives us the information that we need, as shown in Example 5-5.

*Example 5-5 Listing all VolumeAttachments*

```

$ oc get volumeattachment
NAME                ATTACHER
PV                NODE    ATTACHED    AGE
csi-509e1b6a3252892b2beee5b2eababd96079423c601cea65b70b1006182f78003
block.csi.ibm.com  pvc-e1c0274c-6a20-428c-919e-4548f21a4b4f  worker-0  true
90m

```

We see that the PV should be attached to node worker-0, and the attachment occurred. Because we have a single VolumeAttachment in our namespace, we can determine the correct one. In practical environments, we **grep** the output for the PV name. As an administrator, we can also check the situation on the node by debugging it, as shown in Example 5-6. A new pod with the name <node>-debug is created (a process that takes several minutes).

*Example 5-6 Accessing a node for debugging*

```

$ oc debug node/worker-0
Starting pod/worker-0-debug ...
To use host binaries, run `chroot /host`
Pod IP: 172.18.142.35

# Open a second command windows and check on the new created debug container.
# This may take some minutes until the “sh-4.2” command prompt will be shown.

sh-4.2# cat /host/proc/mounts | grep pvc-e1c0274c-6a20-428c-919e-4548f21a4b4f
/dev/mapper/mpathg
/host/var/lib/kubelet/pods/9efce832-94bc-4aaa-a5ff-0103acdaa81e/volumes/kubernetes

```



```
.io~csi/pvc-e1c0274c-6a20-428c-919e-4548f21a4b4f/mount ext4 rw,seclabel,relatime 0
0
```

---

We see the device mapper `/dev/mapper/mpathg` device, which is the provisioned multipath LUN on our DS8000, is mounted to the node operating system under a kubelet-related path. This device is mapped to `/host/var/lib/kubelet/pods` in the debug environment.

Finally, we examine our pod to verify that the file system that is mounted on the node is also available in our pod on the expected path, as shown in Example 5-7.

*Example 5-7 Running commands in a running pod*

---

```
$ oc exec -it pod/csi-demo-02-f6cc59d78-xtqw4 /bin/bash
bash-4.2$ df /tmp/csi-vol
Filesystem          1K-blocks  Used Available Use% Mounted on
/dev/mapper/mpathg  999320    2564   980372    1% /tmp/csi-vol
bash-4.2$ cd /tmp/csi-vol/
bash-4.2$ ls -ltr
total 16
drwxrws---. 2 root 1000570000 16384 Nov  5 08:31 lost+found
bash-4.2$ date > .touched
bash-4.2$ cat /tmp/csi-vol/.touched
Thu Nov  5 08:43:35 UTC 2020
bash-4.2$ exit
```

---

Problems can occur. However, now that we know which parts must come together, we can systematically debug any issues. However, we first want to ensure that the driver is suitably deployed in the cluster.

## 5.3 CSI operator issues

How to deploy the CSI operator is explained in the respective User Guide. Installation from OpenShift's Operator Hub on the OpenShift Console is straightforward, and installation from the CLI on platforms that do not support a GUI-based installation also are not too complicated by following the tips that are provided in 4.3, "CSI deployment on IBM Power by using the command-line interface" on page 72. On successful installation, we find the assets that belong to the operator by selecting those assets that are labeled with `app.kubernetes.io/instance=ibm-block-csi-operator`, as shown in Example 5-8.

*Example 5-8 Successfully deployed operator*

---

```
$ oc get all --selector=app.kubernetes.io/instance=ibm-block-csi-operator
NAME                                READY  STATUS   RESTARTS  AGE
pod/ibm-block-csi-operator-bdfb89bdd-56wg4  1/1    Running  0         3h3m

NAME                                READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/ibm-block-csi-operator  1/1    1           1         29d

NAME                                DESIRED  CURRENT  READY  AGE
replicaset.apps/ibm-block-csi-operator-bdfb89bdd  1        1        1     29d
```

---

We then expect the operator pod to be ready and running, and not showing a fast-growing restart count.

As starting point for more debugging, we might perform the following tasks:

- ▶ Get more information about the status and recent events regarding the pod:

```
oc describe pod/ibm-block-csi-operator-bdfb89bdd-56wg4
```

- ▶ Gather logs from the operator pod:

```
oc logs pod/ibm-block-csi-operator-bdfb89bdd-56wg4
```

More examinations depend on the findings from these commands.

## 5.4 CSI driver CustomResource issues

As we explained in Chapter 4, “OpenShift and Container Storage Interface deployment” on page 45, the CSI driver components are deployed by the operator. To do so, we create a CustomResource type of IBMBlockCSI. From this resource, the operator creates a StatefulSet for the CSI controller component, and a DaemonSet for the CSI node component.

The operator labels all resources that belong to that specific IBMBlockCSI CR with its name so that we can easily find them. Example 5-9 shows the output after a successful installation.

*Example 5-9 Checking driver installation from specific IBMBlockCSI*

---

```
# first switch to the CSI driver namespace
$ oc project ibm-block-csi
Now using project "ibm-block-csi"

$ oc get ibmblockcsi
NAME          AGE
ibm-block-csi 27h

$ oc get all --selector=app.kubernetes.io/instance=ibm-block-csi
NAME                                READY   STATUS    RESTARTS   AGE
pod/ibm-block-csi-controller-0      5/5     Running   0           5h9m
pod/ibm-block-csi-node-97qzw        3/3     Running   0           5h21m
pod/ibm-block-csi-node-gj91k        3/3     Running   0           27h
pod/ibm-block-csi-node-jwqds        3/3     Running   5           27h

NAME                                DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/ibm-block-csi-node    3         3         3       3         3
<none>          27h

NAME                                READY   AGE
statefulset.apps/ibm-block-csi-controller 1/1     27h
```

---

Issues with the CSI Controller or CSI Node deployment can then be tracked by reviewing the respective pod’s status and logs.

## 5.5 CSI driver snapshot feature

You can use the `oc` command to check whether the CSI driver supports the snapshot feature, as shown in Example 5-10. The snapshot feature is highlighted in red in the example.

*Example 5-10 Listing all pod resources in the namespace that include the label `csi`*

---

```
[root@i7PFE7]# # change to project of the CSI driver, in this exampe ibm-block-csi
[root@i7PFE7 OCP_install]# oc project ibm-block-csi
Now using project "ibm-block-csi" on server "https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443".

[root@i7PFE7]# oc get all -l csi -o wide
NAME                                READY STATUS  RESTARTS  AGE  IP              NODE
NOMINATED NODE  READINESS GATES
pod/ibm-block-csi-controller-0      5/5    Running    0         24h  10.128.0.207
i8worker1.ocp-ats.sle.kelsterbach.de.ibm.com <none> <none>
pod/ibm-block-csi-node-6cb6q        3/3    Running    0         24h  9.155.112.84
i8worker2.ocp-ats.sle.kelsterbach.de.ibm.com <none> <none>
pod/ibm-block-csi-node-mxv9n        3/3    Running    0         24h  9.155.112.83
i8worker1.ocp-ats.sle.kelsterbach.de.ibm.com <none> <none>
pod/ibm-block-csi-operator-bdfb89bdd-hrzgn 1/1    Running    0         33d  10.128.0.11
i8worker1.ocp-ats.sle.kelsterbach.de.ibm.com <none> <none>

NAME                                DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR
AGE CONTAINERS                      IMAGES
SELECTOR
daemonset.apps/ibm-block-csi-node  2        2        2      2            2           <none>
24h ibm-block-csi-node,node-driver-registrar,liveness-probe ibmcom/ibm-block-csi-driver-node:1
.3.0,registry.redhat.io/openshift4/ose-csi-driver-registrar:v4.3,registry.redhat.io/openshift4/ose-csi-live
nessprobe:v4.4  app.kubernetes.io/component=csi-node

NAME                                READY  UP-TO-DATE  AVAILABLE  AGE  CONTAINERS
IMAGES                               SELECTOR
deployment.apps/ibm-block-csi-operator 1/1    1            1          43d  ibm-block-csi-operator
ibmcom/ibm-block-csi-operator:1.3.0  app.kubernetes.io/name=ibm-block-csi-operator

NAME                                DESIRED  CURRENT  READY  AGE  CONTAINERS
IMAGES                               SELECTOR
replicaset.apps/ibm-block-csi-operator-bdfb89bdd 1        1          1      43d  ibm-block-csi-operator
ibmcom/ibm-block-csi-operator:1.3.0 app.kubernetes.io/name=ibm-block-csi-operator,pod-templ
ate-hash=bdfb89bdd

NAME                                READY  AGE  CONTAINERS
IMAGES
statefulset.apps/ibm-block-csi-controller 1/1    24h
ibm-block-csi-controller,csi-provisioner,csi-attacher,csi-snapshotter,liveness-probe
ibmcom/ibm-block-csi-driver-controller:1.3.0,registry.redh
at.io/openshift4/ose-csi-external-provisioner-rhel7:v4.4,registry.redhat.io/openshift4/ose-csi-external-att
acher:v4.4,registry.redhat.io/openshift4/ose-csi-external-snapshotter-rhel7:v4.4,registry.redhat
.io/openshift4/ose-csi-livenessprobe:v4.4
```

---

Another option to check the snapshot support of the CSI driver is to check a snapshot image in the CSI operator, as shown in Example 5-11. The snapshot image is highlighted in red in the example.

*Example 5-11 Listing the csi operator POD images*

---

```
[root@i7PFE7]# # change to project of the CSI driver, in this example ibm-block-csi
[root@i7PFE7 OCP_install]# oc project ibm-block-csi
Now using project "ibm-block-csi" on server "https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443".

[root@i7PFE7]# # get the POD name of ibm-block-csi-controller
[root@i7PFE7 OCP_install]# oc get pods | grep ibm-block-csi-controller
ibm-block-csi-controller-0          5/5      Running    0          25h

[root@i7PFE7]# oc get pod ibm-block-csi-controller-0 -o jsonpath="{..image}" | tr -s '[:space:]' '\n' |
sort -u | grep snapshot
registry.redhat.io/openshift4/ose-csi-external-snapshotter-rhel7:v4.4
```

---

## 5.6 CSI driver operation issues

In this section, we assume that the IBM CSI Operator was successfully deployed, and that the CSI driver installation with a IBMBlockCSI CR finished successfully. Now, we review common issues regarding the driver configuration and typical symptoms that are used daily.

### 5.6.1 Configuration issues

The following prerequisites must be met so that the CSI driver works correctly:

- ▶ IBM CSI Operator is deployed and running.
- ▶ A valid IBMBlockCSI CR was created and successfully deployed.
- ▶ The storage backend access information is a secret.
- ▶ A StorageClass for provisioning volumes and an optional VolumeSnapshotClass are defined.

The first two points were discussed in “CSI operator issues”.

To check the configuration, we start with the last point, as shown in Example 5-12. Does a correct StorageClass (or VolumeSnapshotClass respectively) exist?

*Example 5-12 Inspection of a StorageClass for CSI provisioned volumes (shortened)*

---

```
$ oc get StorageClass
NAME          PROVISIONER          RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION  AGE
ds8k-csi     block.csi.ibm.com   Delete         Immediate         false
8d
$ oc get StorageClass/ds8k-csi -o yaml
...
parameters:
  SpaceEfficiency: thin
  csi.storage.k8s.io/controller-publish-secret-name: ds8k-secret
  csi.storage.k8s.io/controller-publish-secret-namespace: ibm-csi-operator
  csi.storage.k8s.io/provisioner-secret-name: ds8k-secret
  csi.storage.k8s.io/provisioner-secret-namespace: ibm-csi-operator
```

```
pool: P4
provisioner: block.csi.ibm.com
...
```

---

Check the values for the references to the secrets. Notably, if they exist in the specific namespace. Use the command that is shown in Example 5-13. Regarding the namespace (`-n` option) and the secret name, it is best to paste the values from the StorageClass to ensure that no typos exist. VolumeSnapshotClasses contains respective references and checking or fixing them follows the same principles that we show in Example 5-13.

*Example 5-13 Verification of a secret reference*

---

```
$ oc -n ibm-csi-operator get secret/ds8k-secret
NAME          TYPE    DATA  AGE
ds8k-secret   Opaque  3      3d5h
```

---

If the references are accurate, issues might exist with the keys and values in the secret. Example 5-14 shows how to list its content.

*Example 5-14 Content of a secret (shortened, modified)*

---

```
$ oc get secret/ds8k-secret -o yaml
apiVersion: v1
data:
  management_address: ZHM4ay1*****
  password: eU5*****Fo=
  username: bW*****=
kind: Secret
metadata:
...
```

---

The keys in the data field provide the access information for the storage backend. These values are base64 encoded, which can be decoded by using `echo "Y3NpLXVzZXIK" | base64 -d`, as an example. If the values are correct, the secret can safely be deleted and recreated on the GUI, or on the CLI. Example 5-15 shows this process on the CLI.

*Example 5-15 Creating a Secret on the CLI*

---

```
$ oc create secret generic secret-name \
  --from-literal=management_address=back-end-address \
  --from-literal=username=back-end-user \
  --from-literal=password=back-end-password \
  --dry-run -o yaml > secret.yaml
$ oc apply -f ./secret.yaml
```

---

The use of the `--dry-run` option allows us to create a yaml file that can be inspected before it is applied. After a new secret is provided, the CSI controller pod is restarted so that it ends any connections that might still use the older access information and fail:

```
oc delete pod/ibm-block-csi-controller-0
```

## Linux for System z considerations

As described in Chapter 4, “OpenShift and Container Storage Interface deployment” on page 45, some specifics exist regarding the FCP devices.

A common symptom if the FCP adapters are not active on a node is that a so-called *nodeid* is not being created. This issue leads to a CSI node pod, which is continuously restarting and put into Error status, as shown in Example 5-16.

*Example 5-16 Pod restarting on a node after fresh installation*

```
[schaefm@t8360030 block-csi]$ oc get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/ibm-block-csi-controller-0      5/5    Running   0           3m14s
pod/ibm-block-csi-node-dntqc        3/3    Running   0           3m14s
pod/ibm-block-csi-node-gj91k       3/3    Running   1           3m14s
pod/ibm-block-csi-node-jwqds        3/3    Running   0           3m14s
pod/ibm-block-csi-operator-bdfb89bdd-6gr8l  1/1    Running   0           28d
```

---

```
NAME                                DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE
daemonset.apps/ibm-block-csi-node  3         3         3       3         3
<none>          3m15s
```

---

```
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ibm-block-csi-operator  1/1     1             1           28d
```

---

```
NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/ibm-block-csi-operator-bdfb89bdd  1         1         1       28d
```

---

```
NAME                                READY   AGE
statefulset.apps/ibm-block-csi-controller  1/1     3m15s
```

To verify whether this issue is the case, check the pod's logs, as shown in Example 5-17.

*Example 5-17 Checking CSI node pod logs*

```
$ oc logs pod/ibm-block-csi-node-gj91k -c ibm-block-csi-node
...
2020-11-04 10:06:02,11410 DEBUG [210] [-] (node.go:594) - >>>> NodeGetInfo: called
with args {XXX_NoUnkeyedLiteral:{} XXX_unrecognized:[] XXX_sizecache:0}
2020-11-04 10:06:02,11410 DEBUG [210] [-] (node.go:605) - <<<< NodeGetInfo
2020-11-04 10:06:02,11410 ERROR [210] [-] (driver.go:83) - GRPC error: rpc error:
code = Internal desc = Unsupported connectivity type : {fc}
```

The CSI node cannot provide any information about its Fibre Channel connectivity. Therefore, the node annotation `csi.volume.kubernetes.io/nodeid` is missing for that node, as shown in Example 5-18.

*Example 5-18 Check nodeid annotation for a Node*

```
$ oc describe pod/ibm-block-csi-node-gj91k | grep Node:
Node:          worker-0/172.18.142.35
$ oc describe node/worker-0 | grep 'kubernetes.io/nodeid'
Showing no result
```

This issue can be solved by following the instructions that are provided in 4.4, “CSI and OpenShift on IBM Z” on page 87. After the adapters are active, the CSI node pod can start and annotate the node object correctly.

## 5.6.2 PVC not binding

A common symptom is that the PVC that we created does not bind to PVs. In that case, we observe a PVC that stays in Pending state for a longer time, and no corresponding PV is appearing. Another issue can be that a newly created pod stays in Pending state for an unexpectedly long time. We review the pod's events by using `oc describe pod/pod-name` because we see some of its volumes do not mount.

Consider the following hints for debugging the issue:

- ▶ Give the platform a bit more time because it might be under stress.
- ▶ Check CSI controller logs to confirm whether the controller allocated the requested storage on the backend:
  - Does it determine the right back-end type for the new volume: DS8K, XIV, or IBM SAN Volume Controller?
  - Are management addresses correct and can the controller log in to the backend?
  - Does the backend refuse to create the volume?
- ▶ Check the storage system:
  - Are the worker nodes correctly configured on the storage system?
  - Is its capacity sufficient?
  - Does the request reach to the back-end?
  - Are volumes created?

## 5.6.3 Volume not mounting

An issue can occur in which the CSI controller successfully created the PV on the backend and the PVC and PV are bound. However, a pod that uses the PVC does not start. Instead, it stays in the Pending state and possibly goes into the Error state after some time.

Consider the following debugging hints:

- ▶ Check VolumeAttachment objects:
  - Does VolumeAttachment for the PV on the targeted node exist?
  - Are other VolumeAttachments for the same PV active, but on a different node?
  - Is a terminating pod on a node possibly not finishing?
- ▶ Check the targeted node of the VolumeAttachment.
- ▶ Check the storage system:
  - Are the worker nodes correctly configured on the storage system?
  - Is the volume mounted to a worker?
  - Check the note about IBM FlashSystem `vdiskprotectionenabled` that is discussed in “Creating a StatefulSet” on page 82.

## 5.7 Searching the CSI related log entries

Example 5-19 shows the steps to list the CSI operator log information.

*Example 5-19 Listing the CSI operator log information only*

---

```
[root@i7PFE7 ~]# # select the CSI project
[root@i7PFE7 ~]# oc project ibm-block-csi
Now using project "ibm-block-csi" on server "https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443".
[root@i7PFE7 ~]# oc get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/ibm-block-csi-controller-0      5/5     Running   0           2d
. . .

[root@i7PFE7 ~]# oc logs pod/ibm-block-csi-controller-0
Error from server (BadRequest): a container name must be specified for pod ibm-block-csi-controller-0, choose one of:
[ibm-block-csi-controller csi-provisioner csi-attacher csi-snapshotter liveness-probe]

[root@i7PFE7 ~]# oc logs pod/ibm-block-csi-controller-0 -c ibm-block-csi-controller
/opt/app-root/lib/python3.6/site-packages/cryptography/hazmat/bindings/openssl/binding.py:177: CryptographyDeprecationWarning: OpenSSL
version 1.0.2 is no longer supported by the OpenSSL project, please upgrade. The next version of cryptography will drop support for it.
  utils.CryptographyDeprecationWarning,
2020-11-29 15:37:53,369 DEBUG [140735520723024] [MainThread] (csi_controller_server.py:start_server:649) - Listening for connections on
endpoint address: unix:///var/lib/csi/sockets/pluginproxy/csi.sock
2020-11-29 15:37:53,370 DEBUG [140735520723024] [MainThread] (csi_controller_server.py:start_server:652) - Controller Server running ...
2020-11-29 15:37:53,487 INFO [140735430783408] [ThreadPoolExecutor-0_0] (csi_controller_server.py:GetPluginInfo:558) - GetPluginInfo
. . .

[root@i7PFE7 ~]#
```

---

Example 5-20 shows the steps to collect the CSI driver log information.

*Example 5-20 Listing the CSI driver log information*

---

```
[root@i7PFE7 ~]# # select the CSI project
[root@i7PFE7 ~]# oc project ibm-block-csi
Now using project "ibm-block-csi" on server "https://api.ocp-ats.sle.kelsterbach.de.ibm.com:6443".

[root@i7PFE7 ~]# # create a directory to store the collected log information
[root@i7PFE7 ~]# mkdir logs

[root@i7PFE7 ~]# # collect csi-controller logs
[root@i7PFE7 ~]# oc logs ibm-block-csi-controller-0 --all-containers > logs/ibm-block-csi-controller

[root@i7PFE7 ~]# # collect csi-node logs, first get the nodes and then gather all nodes information
[root@i7PFE7 ~]# nodepods=`oc get pod -l app.kubernetes.io/component=csi-node
--output=jsonpath={.items..metadata.name}`
[root@i7PFE7 ~]# for nodepod in $nodepods; do oc logs --all-containers $nodepod > logs/$nodepod; done

[root@i7PFE7 ~]# # collect csi-operator log, first get the operator and the collect its log information
[root@i7PFE7 ~]# oc get pod -l app.kubernetes.io/instance=ibm-block-csi-operator
NAME                                READY   STATUS    RESTARTS   AGE
ibm-block-csi-operator-74c99d777c-fsj4j  1/1     Running   0           80m

[root@i7PFE7 ~]# oc logs ibm-block-csi-operator-74c99d777c-fsj4j > logs/operator

[root@i7PFE7 ~]# # describe details of all csi component
[root@i7PFE7 ~]# oc describe all -l csi > logs/describe_csi

[root@i7PFE7 ~]# # describe details of PVC with issues, get unbound PVCs of your namespace/project
[root@i7PFE7 ~]# oc get pvc --no-headers -n fs9110 | grep -v Bound
fs9110-child-3    Pending    fs9110-storageclass-child    5m12s

[root@i7PFE7 ~]# oc describe pvc fs9110-child-3 -n fs9110 > logs/pvc_not_bounded
```



```

[root@i7PFE7 ~]# # describe details of PODs with issues, get not running PODs of your namespace/project
[root@i7PFE7 ~]# oc get pod --no-headers -n fs9110 | grep -v Running
hello-node-854bb954fd-r49kp      0/1   ImagePullBackOff   0      143m
[root@i7PFE7 ~]# oc describe pod hello-node-854bb954fd-r49kp -n fs9110 > logs/pod_not_running

[root@i7PFE7 ~]# # list the created log files
[root@i7PFE7 ~]# ls -l logs

-rw-r--r--. 1 root root   36631 Dec 18 11:45 describe_csi
-rw-r--r--. 1 root root 6130625 Dec 18 10:38 ibm-block-csi-controller
-rw-r--r--. 1 root root   65318 Dec 18 11:09 ibm-block-csi-node-8wh99
-rw-r--r--. 1 root root   53769 Dec 18 11:09 ibm-block-csi-node-chd4q
-rw-r--r--. 1 root root    7331 Dec 18 11:12 operator
-rw-r--r--. 1 root root    2980 Dec 18 11:20 pod_not_running
-rw-r--r--. 1 root root    1712 Dec 18 11:15 pvc_not_bounded

```

---

## 5.8 Changing the CSI driver subscription by using the CLI

You can set the upgrade mode of the CSI driver to `automatic` or `manual` by using the `oc edit` command:

```
# oc edit subscriptions ibm-block-csi-operator
```

Figure 5-1 shows the line to be edited. Change the `installPlanApproval` value to `automatic` or `manual`, depending on your needs.

```

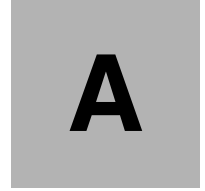
spec:
  channel: stable
  installPlanApproval: Automatic
  name: ibm-block-csi-operator
  source: certified-operators
  sourceNamespace: openshift-marketplace
  startingCSV: ibm-block-csi-operator.v1.3.0
status:

```

Figure 5-1 Editing the CSI driver installation plan approval

After the changes are saved, the new value is automatically applied.





# Terminology

This appendix provides a list of terms and acronyms and their definitions that are used with OpenShift and the IBM CSI driver.

## Term and acronyms

### Block Volume

A volume that will appear as a block device inside the container.

### Ceph

Ceph is an open-source software storage platform. Software defined storage (SDS) that implements object storage on a distributed computer cluster, and provides interfaces for object-, block-, and file-level storage.

### Container

This is a technology for packaging an application along with its runtime dependencies.

### CRI-O

The Container Runtime Interface (CRI) is a plug-in interface that gives kubelets the ability to use different Open Container Initiative (OCI) compliant container runtimes. The CRI-O project allows you to run containers directly from Kubernetes. Each CRI-O version is compatible with the Kubernetes version that has the same version number. CRI-O is the default in OpenShift runtime interface.

### CSI

Container Storage Interface (CSI): an industry standard API to create and use storage services.

### CO

A Container Orchestration system (CO) manages containers automatically. Kubernetes is an open source Container Orchestration system. RPCs Remote Procedure Calls (RPCs) are used for the communication.

### Docker

Docker is a software technology providing containers, promoted by the company Docker, Inc. Docker provides an additional layer of abstraction and automation of the operating system level virtualization on Windows and Linux.

### etcd

etcd is a consistent and highly-available key value store used as Kubernetes' store for all cluster data.

### gRPC

gRPC is a Remote Procedure Call (RPC) platform developed by Goggle.

### Kubernetes

Kubernetes (commonly referred to as *K8s*) is an open source container orchestration system for automating deployment, scaling and management of containerized applications that was originally designed by Google and donated to the Cloud Native Computing Foundation. It aims to provide a platform for automating deployment, scaling, and operations of application containers across clusters of hosts.

### kubelet

The Kubernetes kubelet is an agent running on every machine in the cluster. It interacts with the API server and the controllers in the control plane to fulfil its central tasks: starting, stopping, monitoring and deleting containerized workloads on the machine.

## kube-proxy

The Kubernetes network proxy runs on each node. It reflects services as defined in the Kubernetes API on each node and can do simple TCP, UDP, and SCTP stream forwarding or round robin TCP, UDP, and SCTP forwarding across a set of back-ends. Service cluster IPs and ports are currently found through Docker-links-compatible environment variables specifying ports opened by the service proxy.

## Mounted Volume

A volume that appear as a directory inside the container. It will be mounted using the specified file system type.

## Node

Nodes are Kubernetes' abstraction of real computer. Kubernetes runs your workload by placing containers into Pods to run on nodes. A node may be a virtual or physical machine, depending on the cluster. Each node contains the services necessary to run Pods, managed by the control plane.

## oc

OpenShift CLI (OC) contains commands for managing your applications, as well as lower level tools to interact with each component of your system. You can download and unpack the Command Line Interface (CLI) from the Red Hat Customer Portal for use on Linux, Mac OS X, and Windows clients.

## OCP

Red Hat OpenShift container platform (OCP). An application platform with full stack automation.

## OCS

Red Hat OpenShift Container Storage (OCS)

## OpenShift

OpenShift is developed by Red Hat is an open source development platform. It enables developers to develop and deploy their applications on cloud infrastructure. It is very helpful in developing cloud-enabled services by using the cloud development Platform as a Service (PaaS).

## Plug-in

A plug-in also called "plug-in implementation", is a google remote procedure call (gRPC) endpoint that implements the CSI Services.

## POD

PODs are the smallest deployable units of computing that can be created and managed in Kubernetes and consists of one or more containers.

## PV

PersistentVolume (PV).

## PVC

A Persistent Volume Claim (PVC) object describes the characteristics of a piece of storage from an application's point of view. PVCs offer an abstraction of storage.

## quay.io

Quay.io is a container image repository. It includes features for building images and scanning for security vulnerabilities. It is available as a standalone component or in conjunction with OpenShift.

## ROX

This is the storage volume mode ReadOnlyMany (ROX): the volume can be mounted as read-only by many pods.

## RPC

Remote Procedure Call (RPC).

## RWO

This is the storage volume mode ReadWriteOnce (RWO): the volume can be mounted as read-write by a single pod).

## RWX

This is the storage volume mode ReadWriteMany (RWX): the volume can be mounted as read-write by many pods.

## Sidecars

Sidecar containers extend and enhance a container. They exist in the same pod as the container sharing storage and network with it.

## StorageClass

A StorageClass describes the options an administrator can choose to provide storage to OCP. Different classes might map to quality-of-service levels, or to backup policies, or to arbitrary policies determined by the cluster administrators. Storage classes are transparent for Kubernetes.

## SP

A Storage Provider (SP) is the vendor of a CSI plug-in implementation.

## SpectrumScale

IBM SpectrumScale is a cluster file system that provides concurrent access to a single file system or set of file systems from multiple nodes. The nodes can be SAN attached, network attached, a mixture of SAN attached and network attached, or in a shared nothing cluster configuration.

## Volume

A volume is a unit of storage inside of a CO-managed container. It will be made available via the CSI.

## Workload

A workload is an application running on Kubernetes which you run inside a set of Pods. Your workload can be a single component or several that work together.

## Yaml

Yaml is a human readable data serialization language, typically used for systems configuration files.



# Container Storage Interface support matrix

This appendix provides information about the IBM Container Storage Interface (CSI) driver compatibility and requirements. The supported Red Hat OpenShift access modes for persistent storage are also listed.

This matrix reflects the supported environment at the time of this writing.

This appendix includes the following topics:

- ▶ “IBM block storage CSI driver compatibility and requirements” on page 132
- ▶ “Red Hat OpenShift access modes for persistent storage” on page 133
- ▶ “Red Hat OpenShift CSI volume snapshot support” on page 133

# IBM block storage CSI driver compatibility and requirements

More information about IBM block storage CSI driver compatibility and requirements are listed in the IBM block storage CSI driver documentation, which is available at this [IBM Documentation web page](#).

This information specifies the compatibility and requirements of version 1.3.0 of IBM block storage CSI driver for the following topics:

► Supported storage systems

IBM block storage CSI driver 1.3.0 supports different IBM storage systems as listed in Table B-1.

*Table B-1 CSI driver supported storage systems*

Storage system	Microcode version
IBM FlashSystem A9000/R	12.x
IBM Spectrum Virtualize Family	7.x, 8.x
IBM Spectrum Virtualize as software only	7.x, 8.x
IBM DS8000 Family	8.x and higher with same API interface

► Supported operating systems

The operating systems that are required for deployment of the IBM block storage CSI driver are listed in Table B-2.

*Table B-2 CSI driver supported operating systems*

Operating system	Microcode version
Red Hat Enterprise Linux (RHEL) 7.x	x86
Red Hat CoreOS (RHCOS)	x86, IBM Z
Red Hat CoreOS (RHCOS)	<ul style="list-style-type: none"> <li>► IBM Power</li> <li>► IBM Power architecture is supported on Spectrum Virtualize Family storage systems only</li> </ul>

► Supported orchestration platforms

The orchestration platforms that are suitable for deployment of the IBM block storage CSI driver are listed in Table B-3.

*Table B-3 CSI driver supported orchestration platforms*

Orchestration platform	Version	Architecture
Kubernetes	1.17	x86
Kubernetes	1.18	x86
Red Hat OpenShift	4.3	IBM Z IBM Power with Spectrum Virtualize Family storage systems
Red Hat OpenShift	4.4	x86, IBM Z
Red Hat OpenShift	4.5	x86,



For more information about the release notes and user guide, see this [IBM Documentation web page](#).

Always check this web page for the most current CSI driver updates and new compatibility and requirement information.

## Red Hat OpenShift access modes for persistent storage

A PersistentVolume can be mounted on a host having one of the access modes that are listed in Table B-4.

Table B-4 Access modes

Access mode	CLI abbreviation	Volume can be mounted as
ReadWriteOnce	RWO	Read-write by a single node
ReadOnlyMany	ROX	Read-only by many nodes
ReadWriteMany	RWX	Read/write by many nodes

This paper describes the attachment for Fibre Channel (FC) attached storage. Therefore, the information that is listed in Table B-5 is reduced to the FC support. OpenShift Container Platform versions 4.3, 4.4, and 4.5 support the same access mode, as shown in Table B-5.

Table B-5 Supported access modes for PersistentVolumes

Volume plug-in	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
Fibre Channel	Yes	Yes	No

ReadWriteOnce (RWO) volumes cannot be mounted on multiple nodes. If a node fails, the volume is still assigned to the failed node and cannot be mounted on another node. To make the volume available to another node, you might need to recover or delete the failed node.

For more information, see the OpenShift documentation that is available at [this web page](#).

## Red Hat OpenShift CSI volume snapshot support

CSI volume snapshot is an Red Hat OpenShift Container Platform *Technology Preview* feature only. For more information, see [this web page](#).



# 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 paper.

## IBM Redbooks

The following IBM Redbooks publications provide more information about the topic in this document. Note that some publications that are referenced in this list might be available in softcopy only:

- ▶ *IBM Spectrum Scale CSI Driver for Container Persistent Storage*, REDP-5589
- ▶ *Red Hat OpenShift V4.3 on IBM Power Systems Reference Guide*, REDP-5599
- ▶ *Red Hat OpenShift on IBM Z Installation Guide*, REDP-5605

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

[ibm.com/redbooks](https://ibm.com/redbooks)

## Other publications

The following publications are also relevant as further information sources:

- ▶ *OpenShift OKD on IBM LinuxONE, Installation Guide*, REDP-5561
- ▶ *IBM Storage for Red Hat OpenShift Container Platform V3.11 Blueprint Version 1 Release 1*, REDP-5564
- ▶ *IBM Storage for Red Hat OpenShift Blueprint Version 1 Release 4*, REDP-5565
- ▶ *Red Hat OpenShift and IBM Cloud Paks on IBM Power Systems: Volume 1*, SG24-8459

## Online resources

The following websites are also relevant as further information sources:

- ▶ IBM block storage CSI driver documentation:  
[https://www.ibm.com/support/knowledgecenter/SSRQ8T/landing/IBM\\_block\\_storage\\_CSI\\_driver\\_welcome\\_page.html](https://www.ibm.com/support/knowledgecenter/SSRQ8T/landing/IBM_block_storage_CSI_driver_welcome_page.html)
- ▶ OpenShift Container Platform 4.5 Documentation:  
<https://docs.openshift.com/container-platform/4.5/welcome/index.html>  
Other platforms, like platform 4.4, can be selected on above page.
- ▶ OpenShift Container Platform 4.5 Documentation on persistent storage:  
<https://docs.openshift.com/container-platform/4.5/storage/understanding-persistent-storage.html>
- ▶ Kubernetes concepts:  
<https://kubernetes.io/docs/concepts>

- ▶ Container Storage Interface (CSI) Specification:  
<https://github.com/container-storage-interface/spec>
- ▶ Container Storage Interface (CSI) (industry standard definition):  
<https://github.com/container-storage-interface/spec/blob/master/spec.md>
- ▶ IBM CSI driver source code:  
<https://github.com/IBM/ibm-block-csi-driver>
- ▶ IBM CSI operator source code:  
<https://github.com/IBM/ibm-block-csi-operator>
- ▶ Open source distributed key-value store *etcd* project and IBM detailed information:  
<https://github.com/etcd-io/etcd>  
<https://www.ibm.com/cloud/learn/etcd>:
- ▶ Kubernetes StatefulSet description:  
<https://kubernetes.io/docs/concepts/workloads/controllers/statefulset>
- ▶ How to edit MTU in OpenShift 4 install:  
<https://access.redhat.com/solutions/5092381>

## IBM Knowledge Center Cloud Pak documentation

These websites give information on the IBM Cloud Paks:

- ▶ Cloud Pak for Applications:
  - Storage requirements (primarily ReadWriteOnce (RWO)):  
[https://www.ibm.com/support/knowledgecenter/SSCSJL\\_4.3.x/install-prerequisites.html#storage-access-mode](https://www.ibm.com/support/knowledgecenter/SSCSJL_4.3.x/install-prerequisites.html#storage-access-mode)
  - CodeReady Workspaces and Codewind (ReadWriteMany (RWX) example):  
[https://www.ibm.com/support/knowledgecenter/SSCSJL\\_4.2.x/docs/ref/general/installation/installing-codeready-and-codewind.html](https://www.ibm.com/support/knowledgecenter/SSCSJL_4.2.x/docs/ref/general/installation/installing-codeready-and-codewind.html)
  - Mobile Foundation:  
[https://www.ibm.com/support/knowledgecenter/SSCSJL\\_4.2.x/install-prerequisites-mf.htm](https://www.ibm.com/support/knowledgecenter/SSCSJL_4.2.x/install-prerequisites-mf.htm)
- ▶ Cloud Pak for Data (RWO and RWX):
  - Storage requirements:  
[https://www.ibm.com/support/knowledgecenter/en/SSQNUZ\\_3.0.1/cpd/plan/storage\\_considerations.html](https://www.ibm.com/support/knowledgecenter/en/SSQNUZ_3.0.1/cpd/plan/storage_considerations.html)
  - Informix (RWO example):  
[https://www.ibm.com/support/knowledgecenter/en/SSQNUZ\\_3.0.1/svc-ix/svc\\_ifx\\_configure\\_task.html](https://www.ibm.com/support/knowledgecenter/en/SSQNUZ_3.0.1/svc-ix/svc_ifx_configure_task.html)
  - Watson AIOps (RWO example):  
[https://www.ibm.com/support/knowledgecenter/en/SSQNUZ\\_3.0.1/svc-aiops/aiops-prereqs.html#aiops-prereqs\\_\\_storage](https://www.ibm.com/support/knowledgecenter/en/SSQNUZ_3.0.1/svc-aiops/aiops-prereqs.html#aiops-prereqs__storage)

- ▶ Cloud Pak for Integration storage requirements (RWX and RWO):  
[https://www.ibm.com/support/knowledgecenter/en/SSGT7J\\_20.3/install/sysreqs.html#icip\\_sysreqs\\_\\_file](https://www.ibm.com/support/knowledgecenter/en/SSGT7J_20.3/install/sysreqs.html#icip_sysreqs__file)
- ▶ Cloud Pak for Automation (RWX and RWO):  
[https://www.ibm.com/support/knowledgecenter/SSYHZ8\\_20.0.x/com.ibm.dba.install/op\\_topics/tsk\\_plan\\_storage.html](https://www.ibm.com/support/knowledgecenter/SSYHZ8_20.0.x/com.ibm.dba.install/op_topics/tsk_plan_storage.html)  
Business Automation Workflow and Workstream Service (RWX Example):  
[https://www.ibm.com/support/knowledgecenter/SSYHZ8\\_20.0.x/com.ibm.dba.install/op\\_topics/tsk\\_bawprep\\_storage.html](https://www.ibm.com/support/knowledgecenter/SSYHZ8_20.0.x/com.ibm.dba.install/op_topics/tsk_bawprep_storage.html)
- ▶ Cloud Pak for Multicloud Management (RWX and RWO):  
[https://www.ibm.com/support/knowledgecenter/SSFC4F\\_2.1.0/install/prep.html#storage](https://www.ibm.com/support/knowledgecenter/SSFC4F_2.1.0/install/prep.html#storage)
- ▶ Ansible Tower (RWO):  
[https://www.ibm.com/support/knowledgecenter/SSFC4F\\_2.1.0/install/ansible\\_tower.html](https://www.ibm.com/support/knowledgecenter/SSFC4F_2.1.0/install/ansible_tower.html)
- ▶ Cloud Automation Manager Managed Services (RWX):  
[https://www.ibm.com/support/knowledgecenter/SSFC4F\\_2.1.0/install/infra\\_mgmt\\_config.html](https://www.ibm.com/support/knowledgecenter/SSFC4F_2.1.0/install/infra_mgmt_config.html)
- ▶ Cloud Pak for Security (RWO):  
[https://www.ibm.com/support/knowledgecenter/SSTDPP\\_1.4.0/platform/docs/security-pak/persistent\\_storage.html](https://www.ibm.com/support/knowledgecenter/SSTDPP_1.4.0/platform/docs/security-pak/persistent_storage.html)

## Help from IBM

IBM Support and downloads:

[ibm.com/support](https://www.ibm.com/support)

IBM Global Services:

[ibm.com/services](https://www.ibm.com/services)







REDP-5613-00

ISBN 0738459674

Printed in U.S.A.

Get connected

