ibm.com/redbooks

Red Hat OpenShift and IBM Cloud Paks on IBM Power Systems Volume 1

Dino Quintero
Ricardo Dobelin Barros
Daniel Casali
Luis Ferreira
Alain Fisher
Federico Fros
Luis Daniel Gonzalez
Miguel Gomez Gonzalez
Mahesh Gurugunti
Rogelio Rivera Gutierrez
Nicolas Joly
Boris Litichevsky
Ismael Solis Moreno
Gabriel Padilla
Sudipto Pal
Bogdan Savu
Richard Wale

Redbooks

IBM Redbooks

Red Hat OpenShift and IBM Cloud Paks on IBM Power Systems: Volume 1

March 2020

SG24-8459-00

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

First Edition (March 2020)

This edition applies to:

- Red Hat OpenShift Container Platform for Power Enterprise V3.11
- Red Hat Enterprise Linux Server release V7.6 (Maipo) for ppc64le
- IBM Virtual I/O Server V3.1.1.0
- IBM Cloud PowerVC Manager V1.4.3.1
- Terraform V0.12.9
- provider.null V2.1.2
- provider.openstack V1.22.0

Contents

Notices

Trademarks

Preface

This IBM® Redbooks® publication educates and prepares the readers to understand and enter the multicloud era.

This book describes a journey to the following aspects of multicloud and associated context of application modernization:

- Introduction to the rationale and methodology of this publication
- Concepts and terminology
- Why move to the cloud?
- Introduction to containers and orchestration with Kubernetes
- Introduction to OpenShift on Power Systems
- Why IBM? Why IBM Power Systems?
- Reference architecture for Red Hat OpenShift on Power Systems
- Installation planning, considerations and guidelines to help provide a system configuration and implementation
- Implementation details
- Use case studies

The goal of this publication is to describe the journey to implement an IBM Cloud™ Solution that uses Red Hat OpenShift and IBM Cloud Paks on IBM Power Systems by using theoretical knowledge to learn the concepts, hands-on exercises to practice the theory, and documenting these findings by way of sample scenarios.

The publication addresses topics for developers, IT architects, IT specialists, sellers, and anyone who wants to implement a Red Hat OpenShift and IBM Cloud Paks on IBM Power Systems. This book also provides technical content to transfer how-to skills to the support teams, and solution guidance to the sales team.

This book compliments the documentation that is available at IBM Knowledge Center, and also aligns with the educational offerings that are provided by the IBM Systems Software Education (SSE).

Authors

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

Dino Quintero is an IT Management Consultant and an IBM Level 3 Senior Certified IT Specialist with IBM Redbooks in Poughkeepsie, New York. Dino shares his technical computing passion and expertise by leading teams developing technical content in the areas of enterprise continuous availability, enterprise systems management, high-performance computing, cloud computing, artificial intelligence including machine and deep learning, and cognitive solutions. He also is a Certified Open Group Distinguished IT Specialist. Dino holds a Master of Computing Information Systems degree and a Bachelor of Science degree in Computer Science from Marist College.

Ricardo Dobelin Barros is an IBM Certified IT Specialist and IT Architect in Brazil. Ricardo joined IBM in 2002 and has a total of 20 years of experience in the IT industry. He works in the Sales Solutions group for IBM Services™ as pre-sales Technical Solution Architect. Ricardo has expertise in availability management, service-levels management, sizing, and performance tuning in operating systems. He also has experience in capacity planning, modelling, measurement, automation, virtualization, multicloud, and infrastructure management for many IBM clients in Brazil. Ricardo holds a bachelor degree in systems analysis, post-graduate in Teaching Degree in Higher Education. He also has Master of Business Administration in Strategic Management and Business Planning.

Daniel Casali is a Thought Leader Information Technology Specialist working for 15 years at IBM with Power Systems, high-performance computing, big data, and storage. His role at IBM is to bring to reality solutions that address client's needs by exploring new technologies for different workloads. He is also fascinated by real multicloud implementations, always trying to abstract and simplify the new challenges of the heterogeneous architectures that are intrinsic to this new consumption model, be that on-premises or in the public cloud.

Luis Ferreira is a Senior Software Engineer at IBM Austin, working on cloud containers, Kubernetes-related products, and cloud computing design and architecture. He holds a Master of Science degree from Universidade Federal do Rio de Janeiro in Brazil. Before joining IBM Austin, Luis worked at Tivoli® Systems as a Certified Tivoli Consultant, at IBM Brasil as a Certified IT Specialist, and at Cobra Computadores as a SOX Kernel developer and operating systems designer.

Alain Fisher is an IT specialist and DevOps Developer and supports many IBM development teams at the IBM Hursley Lab, UK. He holds a B.Sc. (Hons) degree in Computer Science from The Open University, England. He joined IBM in 2001 supporting DB2® middleware products, such as DB2, before moving to the Hursley Lab. His areas of expertise include the OpenStack cloud platform, cloud deployments, automation, and virtualization. He contributed to two other IBM Redbooks publications since 2005.

Federico Fros is an IT Specialist who works for IBM Global Technologies Services, leading the UNIX and Storage team for the IBM Innovation center in Uruguay. He has worked at IBM for more than 15 years, including 12 years in IBM Power Systems and IBM Storage. He is an IBM Certified Systems Expert for UNIX and high availability. His areas of expertise include IBM AIX®, Linux, PowerHA® SystemMirror®, IBM PowerVM®, SAN Networks, Cloud Computing, and IBM Storage Family, including IBM Spectrum® Scale.

Luis Daniel Gonzalez is a System Administrator and an Automation Engineer on Power Systems in IBM Guadalajara, Mexico. He holds a B.Sc. degree in Cyber Security and Networking Engineering. He joined IBM in 2018 as an automation engineer, helping to automate Power Systems installation and configurations. He has experience in configuring Linux systems and services for 5 years.

Miguel Gomez Gonzalez is an IBM Power Systems Integration engineer who is based in Mexico with over 11 years experience in Linux and IBM Power Systems technologies. He has extensive experience in implementing several IBM Linux and AIX clustering solutions. His areas of expertise are Power Systems performance boost, virtualization, high availability, system administration, and test design. Miguel holds a Master in Information Technology Management from ITESM.

Mahesh Gurugunti is a Senior Solutions Architect at Red Hat in New York City. Mahesh is a seasoned technologist with experience in engineering, design, and architecture of IaaS, PaaS, and Cloud infrastructures. He is certified in OpenStack and high availability (design and implementation) that uses Veritas Cluster Services.

Rogelio Rivera Gutierrez joined IBM in 2011 as part of the Storage Development group. As a Test Performance Lead, he has become a SME in tape technologies and has trained customers and IBMers on IBM Spectrum Archive. Rogelio has published several technical white papers about tape products. As a Product Field Engineer, he provides support for customer escalations from beginning to resolution for the IBM Security Appliance Product Line. He also participated and provided support as a Hardware Development Engineer in the release of IBM POWER9™. Rogelio holds a Master degree in Computer Science and is currently pursuing a Ph.D. in IT, focusing on topics, such as Cloud, Blockchain, and Smart Cities. He has presented at IEEE technical conferences with topics related to Cloud and Blockchain technologies.

Nicolas Joly is a pre-sales architect with IBM Systems in New York City, New York. His areas of knowledge include software-defined infrastructure, analytics solutions, storage, technical computing, and clustering solutions. He is working with major customers in the finance and telecommunication industry. Before joining IBM US, Nicolas was working for IBM France, where he was a technical sales specialist for analytics and technical computing solutions. Nicolas holds a Master's degree in Computer Science with a major in parallel and distributed computing from Institut Polytechnique de Bordeaux (ENSEIRB-MATMECA), France.

Boris Litichevsky is a Infrastructure Analyst IV Horizon Blue Cross. He works with RHEL and lately with Red Hat on Power Systems. Boris participated in the IBM Cloud Private PoC. He worked as a Senior Systems Engineer Consultant working with Tivoli Storage Manager (TSM), Commvault Backup solution, VMware and created and configured Linux (RHEL and CentOS) and Wintel servers 2008 and 2012. Boris has 20 years in the financial industry, 16 of which in computer operations and 4 years as a TSM administrator.

Ismael Solis Moreno is a data scientist and performance analyst for IBM at Mexico Software Lab. He received his Masters from the National Center of Research and Technological Development in Mexico and his PhD from the University of Leeds in the UK. Ismael has participated as technical leader in different projects that were related to big data analytics and machine learning within the IBM and other companies, including the University of Leeds, The UK Datacenter Alliance, and Apollo MIS researching predictive algorithms for Google, Alibaba, and the British Premier League. Ismael has approximately 15 international publications in prestigious computing science journals. He has participated as a speaker in over 25 international conferences related to data science and machine learning, and co-authored patents to improve datacenter energy efficiency by using big data. Ismael has collaborated as researcher in the field of data science at the University of Leeds in the UK, The National University of Defense and Technology, and The University of Aeronautics and Astronautics in China. Ismael's work at IBM is focused on developing machine learning mechanisms to improve the performance of large distributed storage systems by analyzing big data.

Gabriel Padilla is a Test Architect for Hardware on Power System. He is involved in early faces of most of the hardware general availability working closely with development to achieve best quality products. Gabriel previously worked on manufacturing lines at IBM Mexico as Test Lead for XIV® Storage and Power Systems. Gabriel has a Masters degree in Information Technology and a Bachelors of Science in Electronic Engineering.

Sudipto Pal is Solution Architect for IBM Cognos® Analytics in GBS. He successfully delivered several critical deliverable with IBM clients from USA and Europe. He led Cognos administration competency and monitored several candidates. He co-authored IBM Redbooks publications about Cognos implementation with PowerVM platform. He has experience in IBM Power system for Virtualized environment setup and provisioning. He also has hands-on experience in data lake implementation by using DIP over a big data platform. He is based in IBM India, Kolkata. He holds Master of Computer Application and has experience in product development that uses C, C++ and Python,

Bogdan Savu is a Cloud Infrastructure Architect at IBM Cloud Managed Application Services and works for IBM Global Technologies Services in Romania. He has over 13 years of experience in designing, developing, and implementing Cloud Computing, Virtualization, Automatization, and Infrastructure solutions. Bogdan holds a Bachelor's degree in Computer Science from the Polytechnic University of Bucharest. He is an IBM Certified Advanced Technical Expert for Power Systems, TOGAF 9 Certified, VMware Certified Professional, and Red Hat Certified Specialist in Containerized Application Development. His areas of expertise include Cloud Computing, Virtualization, DevOps, and Scripting.

Richard Wale is a Senior IT Specialist, supporting many IBM development teams at the IBM Hursley Lab, UK. He holds a B.Sc. (Hons) degree in Computer Science from Portsmouth University, England. He joined IBM in 1996 and has been supporting production AIX systems since 1998. His areas of expertise include IBM Power Systems, PowerVM, AIX, and IBM i. He has participated in co-writing many IBM Redbooks publications since 2002.

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

Wade Wallace
IBM Redbooks, Austin Center

Manoj Kumar, Joe Cropper, Chuck Bryan, Keshav Ranganathan, Bruce Anthony, Bruce Semple, Reza Ghasemi, Mike Easlon
IBM USA

Miguel Angel de la Mora, Cesar Dominguez Moreno, Guillermo Hernandez Gonzalez, Arianne Navarro
IBM Guadalajara, Mexico

Yenugu Madhavi
IBM India

Alfonso Jara
IBM Spain

Now you can become a published author, too!

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

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks
- Send your comments in an email to:

  redbooks@us.ibm.com
- Mail your comments to:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:

  http://twitter.com/ibmredbooks
- 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

Part 1

Introduction

This part provides an overview of this first publication of a planned multi-volume series, and introduces fundamental cloud concepts, topics and components.

The following chapters are included in this part:

1

Introduction to the Journey to the Cloud: Volume 1

This chapter focuses on the rationale behind the multiple publication volumes. This chapter describes why delivering by way of volume methodology helps the reader throughout the journey to multicloud.

This chapter includes the following topics:

1.1 Introduction

Most companies started or are contemplating their journey to cloud. Although in recent years the adoption of cloud became much more common place, the scope of what a cloud is or can be also increased. This broadening of possibilities unfortunately added confusion and can result in companies being unsure of how their existing application estate can change to integrate with the cloud model.

As such, doubts still exist around how to start and progress on this journey. It is also true that although people understand traditional enterprise applications and more modern cloud-hosted applications, the integration or co-existence of both can prove equally confusing and contradicting.

Recent industry trends, combined with the new partnership between Red Hat and IBM, seek to bring some clarity to the landscape while providing new modernization opportunities for existing enterprise applications and familiar environments.

The main focus of this IBM Redbooks publication relates to IBM Cloud Paks and Red Hat OpenShift, which is hosted on IBM Power Systems. Although individually much can be written about either topic, the relationship this publication highlights is between Red Hat OpenShift and IBM Power Systems.

We show what Red Hat OpenShift brings to the IBM Power Systems platform specifically discuss how it can be deployed and added into existing familiar Power System environments, and the benefits that integration and co-existence can provide from an existing enterprise application viewpoint.

This publication is a first volume in a planned multi-volume publication over the next 12 - 18 months. Within this initial volume, we explain the fundamental perspective (which is accurate as of the time of this writing) while providing pointers to future direction that will be discussed in future volumes.

Note: This initial publication relates to Red Hat OpenShift 3.11, because this release was the current OpenShift Container Platform (OCP) release for IBM Power Systems at the time of this writing. IBM and Red Hat intend to deliver Red Hat OpenShift 4 for IBM POWER® to accelerate agility for enterprise clients through integrated tooling and a feature-rich Kubernetes container platform for cloud-native development on POWER9 and IBM POWER8® processor-based servers.

1.2 Red Hat and IBM

On July 9th, 2019, IBM closed its acquisition of Red Hat, a leader in enterprise Linux and open source technology.

This acquisition puts Red Hat and IBM in a unique position to unlock the true value of hybrid cloud for your business. By combining the power and flexibility of Red Hat’s open hybrid cloud technologies with the scale and depth of IBM innovation and industry expertise, you now have the tools to accelerate your cloud journey.

IBM and Red Hat worked together for more than 20 years in making open source a competitive advantage for businesses on x86, IBM Power Systems, and IBM z Systems®. Together, we are both on a mission to improve open source technology and help your companies capture the business value of the cloud.

This publication describes how Red Hat and IBM can advance your cloud journey and speed growth and innovation for your business by using Red Hat OpenShift on IBM Power Systems.

Note: Red Hat joins IBM as a distinct unit, preserving the independence and neutrality of Red Hat's open source development heritage and unique development culture. Red Hat's unwavering commitment to open source remains unchanged and it continues to offer customers choice and flexibility.

Introduction to containers and orchestration with Kubernetes

This chapter presents the conceptual foundations of containers and the open source container orchestration Kubernetes. It also introduces the Red Hat Enterprise Kubernetes product that is called Red Hat OpenShift.

This chapter includes the following topics:

2.1 A new computing paradigm in cloud transformation

Cloud computing transformed the way that IT is managed.

In the traditional method of using services or resources, the owner of the infrastructure is responsible for managing every piece of hardware and software they use. Normally, it takes some time for a user to access a new resource, but it can be configured exactly as needed.

Traditional infrastructure is often related to aging core applications (typically integrated with aging infrastructure and technologies) that cannot be easily migrated to cloud paradigms. Elasticity, standardization, and other clear cloud advantages are not sufficient reasons to migrate. In other cases, rigid security and country regulations sometimes force users to locate data nearby and under total management control.

2.1.1 Cloud service model

This section describes the different cloud service models.

Infrastructure as a service (IaaS)

The management responsibility for the company starts with the operating system layer and the provider ensures the availability and reliability of the infrastructure provided.

Several use cases can benefit from this pattern. Companies that lack an owned data center look to IaaS as a quick, cheap infrastructure for their business initiatives that can be expanded or ended as needed. Traditional companies that need compute power to run variable workloads with less capital expenditure are perfect examples of IaaS adoption. In both cases, companies pay only for the services that they use.

Platform as a service (PaaS)

Development companies and factories that want to implement agile methodologies are the most suited for PaaS. PaaS providers publish many services that can be used inside applications. Those services are always available and up-to-date. PaaS provides a simple way to test and prototype new applications. It can save money when developing new services and applications. Applications can be released more quickly than usual to get user feedback.

The API economy is the new paradigm in development. The cloud provides the perfect platform for its implementation.

Software as a service (SaaS)

At the time of this writing, SaaS patterns are accepted by many companies that want to benefit from application usage without the need to maintain and update infrastructure and components. Mail, ERP, collaboration, and office apps are the most accepted SaaS solutions. The flexibility and elasticity of the SaaS model are great benefits.

No “one-size-fits-all” solution exists for cloud adoption. Companies must consider their own cost and benefit equation and then decide on the best model. Each application and process that is needed is a workload. A deep workload assessment is normally performed by companies that decided to move to the cloud.

2.1.2 Cloud adoption

The initial forays for most early adopters were focused on the easy things. Moving basic applications to cloud infrastructure to cut costs and building new applications in the cloud to speed innovation. However, to date only approximately 10 - 20% of applications that are moved to the cloud. The next 80% of the cloud opportunity focuses on shifting business applications to the cloud and optimizing everything from supply chains to sales transactions. To address that opportunity, companies must move and manage data, services, and workflows across multiple clouds (companies are running 5 - 16 cloud vendors). This issue is challenging for them because it is largely a manual process, with major security implications and inconsistent cloud management and automation services.

2.1.3 Why is hybrid cloud so important

A one-cloud-fits-all approach does not work, Companies are choosing multiple cloud providers and clouds (public, private, software-as-a-service, and so on) to best meet their needs. They also are integrating those clouds with existing IT to get more value.

The result is a hybrid approach to a multicloud environment. It is a mix of public clouds to quickly develop and deploy applications, private clouds to maintain the highest levels of security and availability for business-critical data and processes, and in many cases traditional on-premises IT.

2.1.4 Application modernization journey for a cloud-centric business transformation

The journey to the cloud starts with application modernization. On average, only 20% of applications in an environment were moved to the cloud. Application modernization is a strategy for helping companies to move aging application estates to the cloud.

Business pressures demand faster time to market, robust workload mobility to allow quick migrations to the cloud, which improves operational efficiency, elasticity, and cost reductions. Applications must be adapted and ported to the cloud. Cloud-native technologies, such as containers, Kubernetes, and microservices, simplify this journey. How much time do you need to rewrite the entire portfolio of applications? Of course, it is a lot; therefore, a gradual application modernization is inevitable.

Modernizing your applications results in the following immediate benefits:

- Accelerate digital transformation. New applications quickly match the business requirements.
- Improve developer productivity. Application containerization enables self-service for developers.
- Improve operational efficiency and standardization. CI/CD practices of continuous integration and continuous delivery to take advantage of the DevOps enablement culture.

An application modernization aligns with the hybrid nature of cloud adoption by large enterprises. It fully supports multicloud deployments that can span private, on-premises clouds and various public clouds by using standard Kubernetes-based container orchestration platforms, such as Red Hat OpenShift. For more information, see 2.5, “Enterprise Kubernetes: Red Hat OpenShift” on page 31.

To start the modernization journey, first you must analyze your current application estate and prioritize your modernization goals. The outcome of the analysis can help you select the most suitable modernization patterns:

- Move the monolith to cloud

  Containerize the applications, also called the VM-based *lift and shift* method. When you containerize an application, you reduce future costs, simplify operations, and provide flexible cloud migration to a common Kubernetes platform. You also focus on a single operational model that can be used by new cloud-native and traditional applications.

- Expose on-premises assets by using APIs

  The use of APIs allows new applications to use the exposed capability in an accessible way. Typically, this connection is made by way of the OpenAPI Specification (OAS) REST standard that allows the interface to be discovered and managed.

- Refactor the monolith into macroservices

  Transform your monolith application to become cloud-native. For example, when you deal with large Java or JavaScript files and break monoliths into smaller deployable components.

- Add capabilities as microservices

  Enrich your application with new business functions that are implemented as microservices. Your application can use various services that are available in the cloud, such as AI, industry, and domain services.

- Strangle the monolith

  Incrementally withdraw the monolith from service by replacing functions with new cloud-native implementations.

To help you on your cloud journey, IBM introduced new and better experiences on modernization approaches, such as the incremental lift-and-shift to cloud-native microservices. You do not need to rewrite your entire application assets to move to the cloud. With a cloud-native microservices approach, you can capitalize on the scalability and flexibility that is inherent in any robust cloud infrastructure.

DevOps approach

DevOps is about tools and techniques that are needed to bring the worlds of development and operations closer together. It is about the practices and automation that is required to reliably and rapidly deploy new code through the build and test cycles to production environments.

DevOps is an approach where companies and their development, operations, and quality assurance teams collaborate to continuously deliver software. By following this approach, which is based on lean and agile principles, enterprises can seize market opportunities and reduce the time to include customer feedback in their products. DevOps applies to aging, traditional, cloud-ready, and cloud-native applications.

Failing fast and iterating quickly are DevOps requirements for competitive app delivery. They imply application architectures that decouple services from each other in a continuous development and delivery cycle, although ensuring well-performing interactions with users.

In a cloud-native environment, applications are viewed as collections of microservices with many deployable components that deploy at various speeds and independently of each other. Manually deploying anything becomes an unsustainable effort that is prone to error, inconsistency, and delays.

Automating continuous delivery pipelines is a way to help operations from deploying an application they have little knowledge of and rescue developers from complex configurations and deployments. Efficient delivery means more time for innovation, which enables the business to disrupt the market and competitors.

Whether they are accessing a website or a mobile app, people have high expectations of the apps that they regularly use. As a result, companies must continuously deliver new features and fixes. In the past, this process was painful because an application was developed, built, and made available as a single, often monolithic, application.

Containerization or the paradigm shift towards microservices

Microservices is an application architectural style in which an application is composed of many discrete, network-connected components called microservices. Consider the following points:

- Large monolithic applications are broken into small services.
- A single network-accessible service is the smallest deployable unit for a microservices application.
- A microservice is a small application that usually houses one function. The function is exposed through APIs and messaging. This rule, sometimes referred to as *one service per container*, might be a container or any other lightweight deployment mechanism,
- Each microservice can feature its own DevOps pipeline, scale individually, and have its own database where it owns a data model.

Figure 2-1 shows how a monolithic application architecture evolves into a microservices-based application architecture.

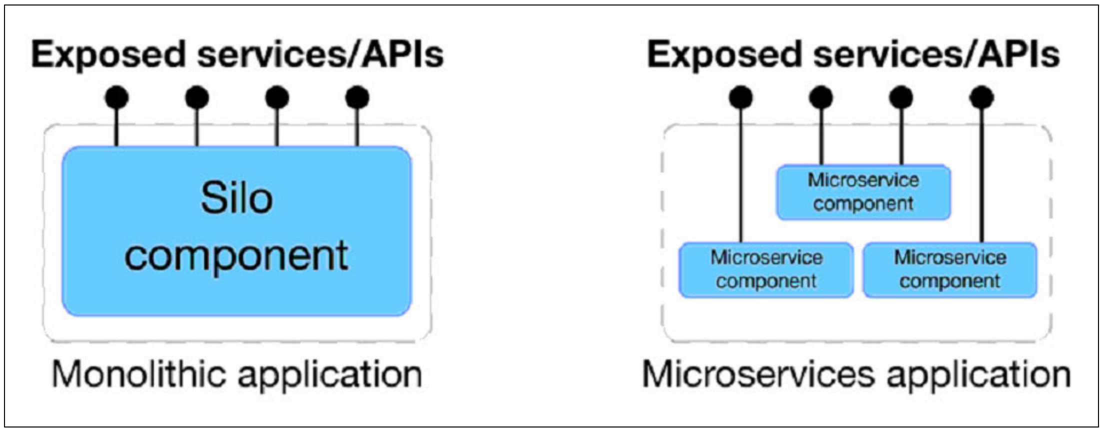


Figure 2-1 Monolithic application versus Microservices application

2.2 Virtual machines meet containers

The use of a hybrid container architecture by combining virtualization by both virtual machines and containers can help to address the requirement for a robust cloud computing platform.

2.2.1 Perfect recipe for application modernization

For years, Infrastructure as a Service (IaaS) cloud was the main focus of cloud implementations. The goal of IaaS is to provide virtual machines as a service. IaaS standards initiatives, such as OpenStack and others, were developed to establish an easy way to provision virtual machines.

Considering the IBM Power Systems environment, the IBM Power Virtualization Center (PowerVC) product became the key element for cloud implementation. Software developers and vendors created VM images as the way to deploy workloads. Putting in perspective, VM clouds were an evolution of the old grid computing architectures, where the elasticity and power were determined by the number of nodes that were performing tasks remotely. With IaaS, cloud infrastructure became solid and easy to implement.

PowerVC is a virtualization management offering that simplifies the tasks of creating and managing virtual machines on IBM Power Systems servers. At the lower level, PowerVC works with virtualization resources from PowerVM or KVM on Power hypervisors.

At a higher level, as shown in Figure 2-2 on page 13, PowerVC interacts with cloud management services through OpenStack software infrastructure. It is designed to build a private cloud on the Power Systems servers and improve administrator productivity. It can further integrate with multicloud environments through higher-level cloud orchestrators.

For more information about PowerVC, see this website.

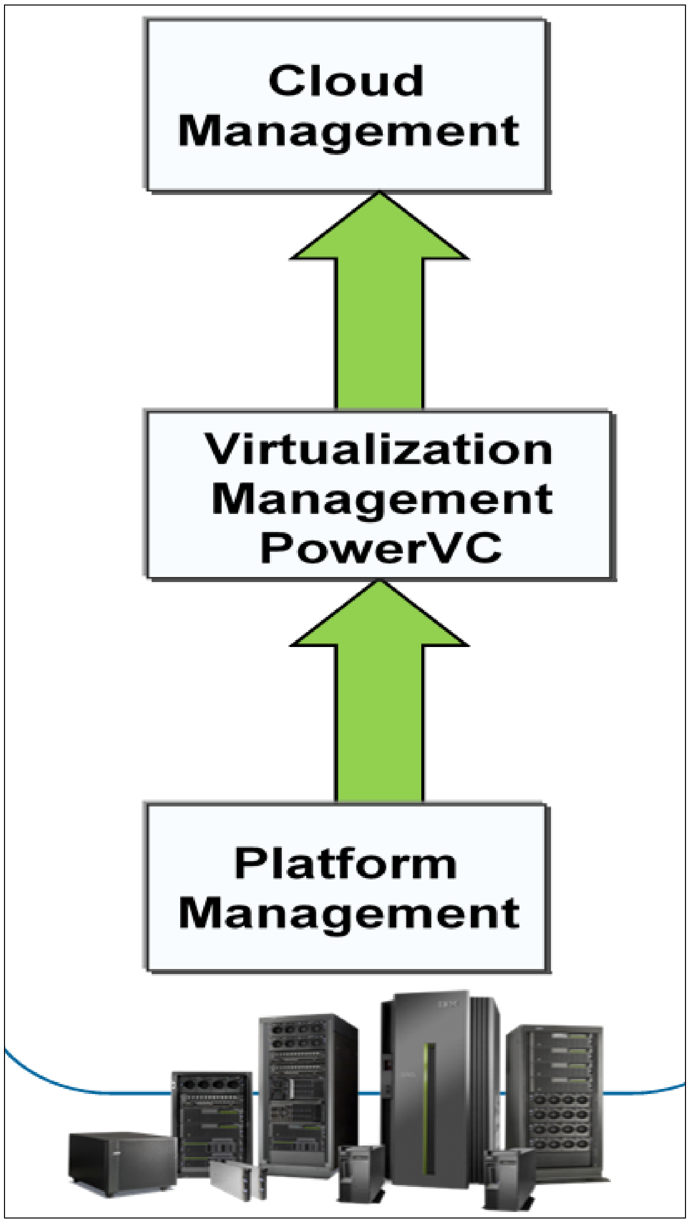


Figure 2-2 IBM PowerVC

Around 2011, Container technology started to be a strong player in the cloud arena, which is a method to package an application in a box so it can be run with its dependencies, isolated from other applications. For more information, see 2.3, “Containers” on page 19.

A year later, Docker Containers exploded in popularity, but one thing was missing: the thorough view and management of the entire environment.

IT Service Management and orchestration

An IT Service Management (ITSM) perspective can provide automation and a global management view, and incorporate the necessary software disciplines that are required to build a solid infrastructure for an enterprise, commercial or not.

The missing point was the orchestration and the orchestration of all containers and resources around them. Many people think that orchestration and automation are the same thing, but the orchestration is more complex. Automation often is discussed in the context of specific tasks, whereas orchestration refers to the automation of processes and workflows.

Orchestration deals with the end-to-end process simplify the automation and the administration across specific machines and diverse dependencies (see Figure 2-3). Automation attempts to move people out of the equation whereas orchestration is not about rigid planning, but arranging and coordination of automated tasks, which ultimately results in a consolidated process or workflow. Parts can be automated, but the decision is still human-centric; for example, the definition of which tasks must run, the order of the tasks, role assignments, permission, post-deployment, failure recovery, and scaling.

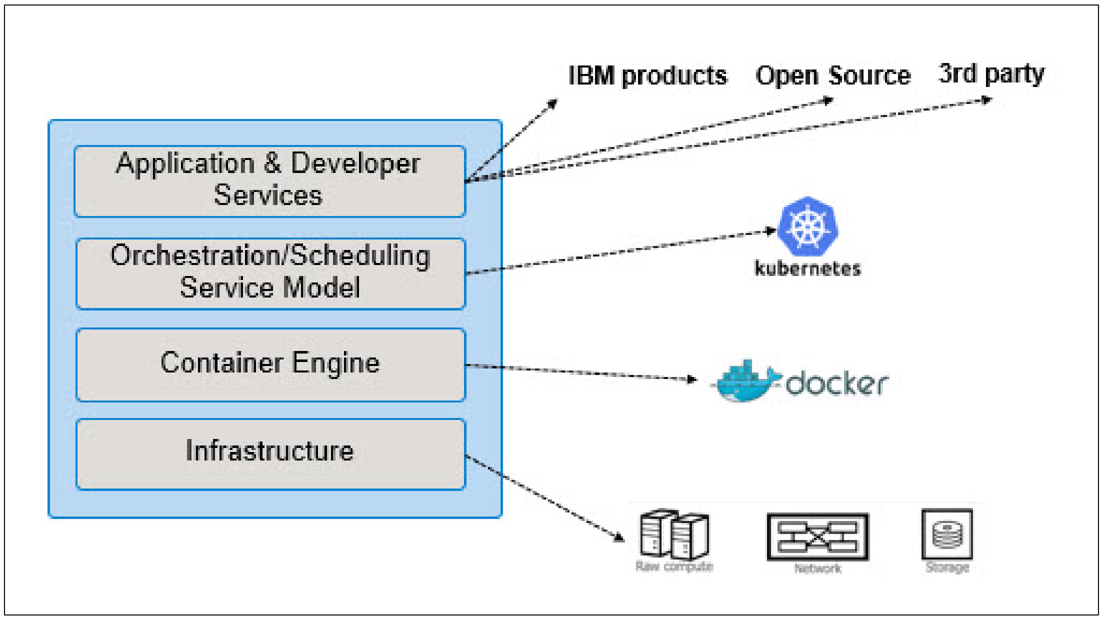


Figure 2-3 Where orchestration fits

For more information about automation, see 2.4, “Kubernetes: An open source container orchestration” on page 24.

Cloud engineering

In the same line of ITSM, the application of an engineering approach on cloud infrastructures helped the clients and system administrators to integrate better and manage their day-to-day business.

Cloud engineering focuses on cloud services, such as SaaS, PaaS, and IaaS. It is a multidisciplinary method that includes the foundation of cloud, implementation, cloud development-delivery lifecycle, and management.

An orchestrator normally includes a range of technologies, products, and components, as shown in Figure 2-4.

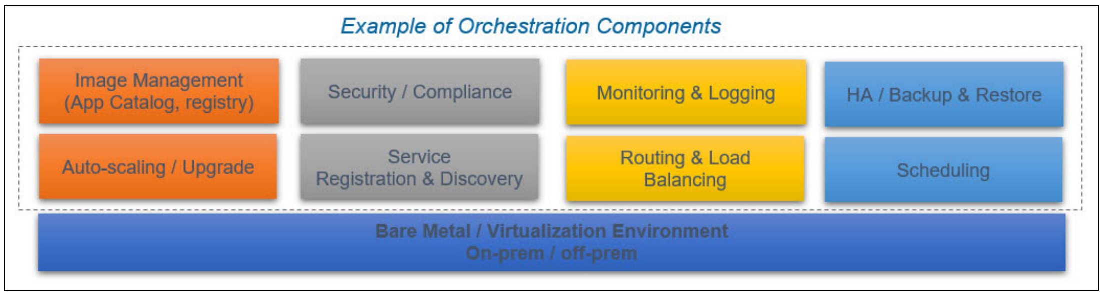

Example of Orchestration Components
Image Management (App Catalog, registry)
Security / Compliance
Monitoring & Logging
HA / Backup & Restore
Auto-scaling / Upgrade
Service Registration & Discovery
Routing & Load Balancing
Scheduling
Bare Metal / Virtualization Environment
On-prem / off-prem

Figure 2-4 Example of Orchestration Components

The following cloud engineering disciplines are addressed by an orchestrator:

- Platform management
- Virtualization services
- Authentication and authorization services
- Resources management
- Disaster recovery
- Workload resilience
- Monitoring, usage, and accounting
- Configuration services
- Application lifecycle
- Service automation
- Service catalog

2.2.2 Coexistence of virtual machines and containers to modernize workloads

The component that is responsible for running containers is the container engine (Docker). In a hybrid cloud or not environment, you can have a container engine running on a stand-alone server or in a virtual machine-based server, as shown in Figure 2-5. For the functional standpoint, there is no difference where the container engine runs. Some management advantages exist for having containers on virtual machines, mainly in terms of flexibility and control. Depending on what you need to do, a virtual machine might be the best place to host those containers.

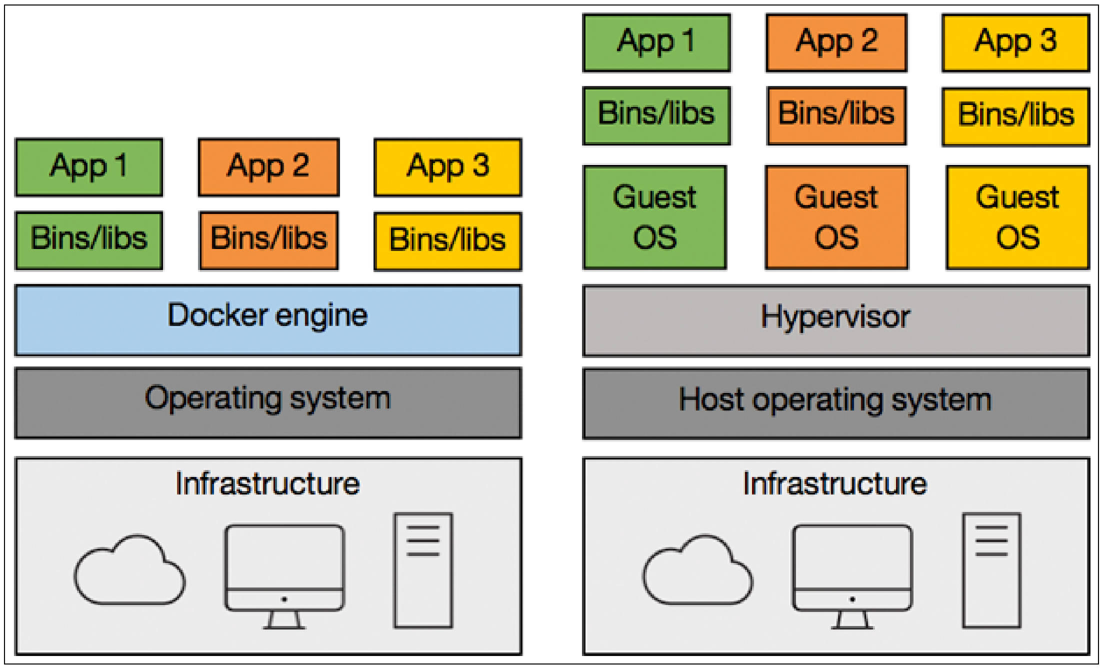


Figure 2-5 Docker engine versus virtual machine

The synergy of combining virtual machines and containers empowers the system administrator to better tune the resources (processors, memory, storage, and network). They also can optimize the overall usage of the available assets to maximize and simplify the use of the physical hardware.

For management and infrastructure nodes of a cloud cluster, the use of virtual machines might have more advantages. The system administrator has robust control of the resources that are assigned to these nodes, which makes it easy to maintain the nodes as active and well-tuned.

The decision to run it on a virtual machine or in a bare-metal machine must follow largely the non-functional requirements of the architecture decisions. You can also take advantage of the virtual resources that are provided by a virtual machine. Containers can interact with virtual machine-provided services. For example, your application containers must interact with a database that uses storage that is backed by volumes that are managed through IBM PowerVC.

You can dynamically increase or decrease the amount of storage available to the node as the deployed containers change. Figure 2-6 shows an example of how several containers across multiple virtual machines can share one set of volumes.

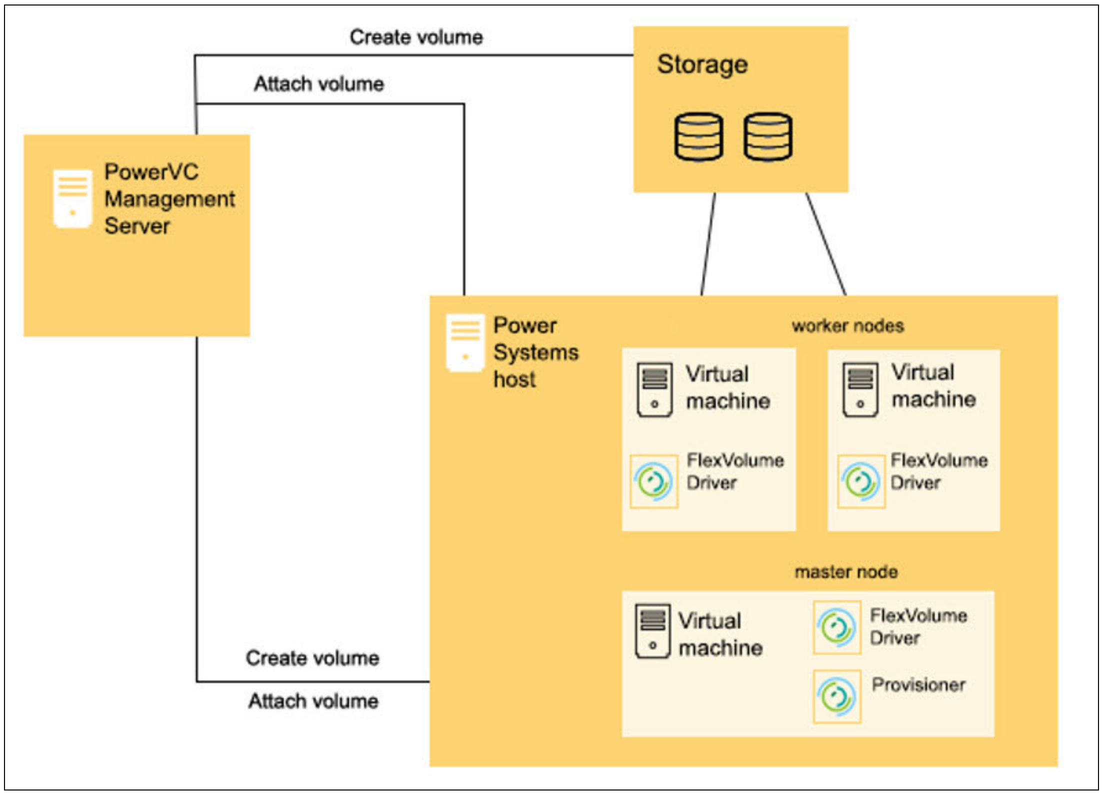


Figure 2-6 Storage Backed By Volumes Managed Through PowerVC

Note: For more information about the use of IBM PowerVC storage with containers, see IBM Knowledge Center.

2.2.3 Virtual machines and containers in a hybrid multicloud architecture

Hybrid multicloud is a cloud environment that combines a private cloud and public clouds that allows applications and data to be shared between them. A *multicloud* refers to an environment that is made of up of more than one cloud provider (or vendor). Hybrid is an environment that combines a private cloud and a public cloud that allows applications to take advantage of the resources on either cloud. Therefore, hybrid multicloud combines a private cloud, a public cloud, and more than one cloud service from more than one cloud vendor.

IBM Power Systems customers use a simplified deployment of enterprise resources by using various virtual machines (LPARs). Enterprises worldwide are exploring container technology and developing plans for how to integrate them into their enterprise. They want to start to deploy, manage, and operate containerized applications smoothly by integrating them into virtual machines.

IT administrators, developers, and line-of-business users want to continue a simplified access to the infrastructure and applications, which is possible by adopting the IBM Power Systems cloud technologies within the data center.

From a Power Systems view, IBM PowerVC delivers the IaaS layer on-premises, which simplifies virtualization management and cloud deployments virtual machines. This integration with multicloud environments is by way of cloud orchestrators, such as the Kubernetes-based Red Hat OpenShift platform.

OpenShift provides a universal PaaS solution that covers the entire hybrid multicloud landscape. Therefore, users can run their software of choice, which is built as containers, including IBM enterprise software delivered by way of IBM Cloud Paks (see Chapter 3, "IBM Cloud Paks: Middleware anywhere" on page 37), ISV applications, and Open-source software (OSS). Finally, technologies, such as IBM Multicloud Manager or VMware vRealize, can integrate the historically separate cloud infrastructures into a single interconnected cloud fabric.

Figure 2-7 shows an example of a hybrid multicloud scenario that is inclusive of the major hardware platforms, including IBM Power Systems, IBM Z®, and x86.

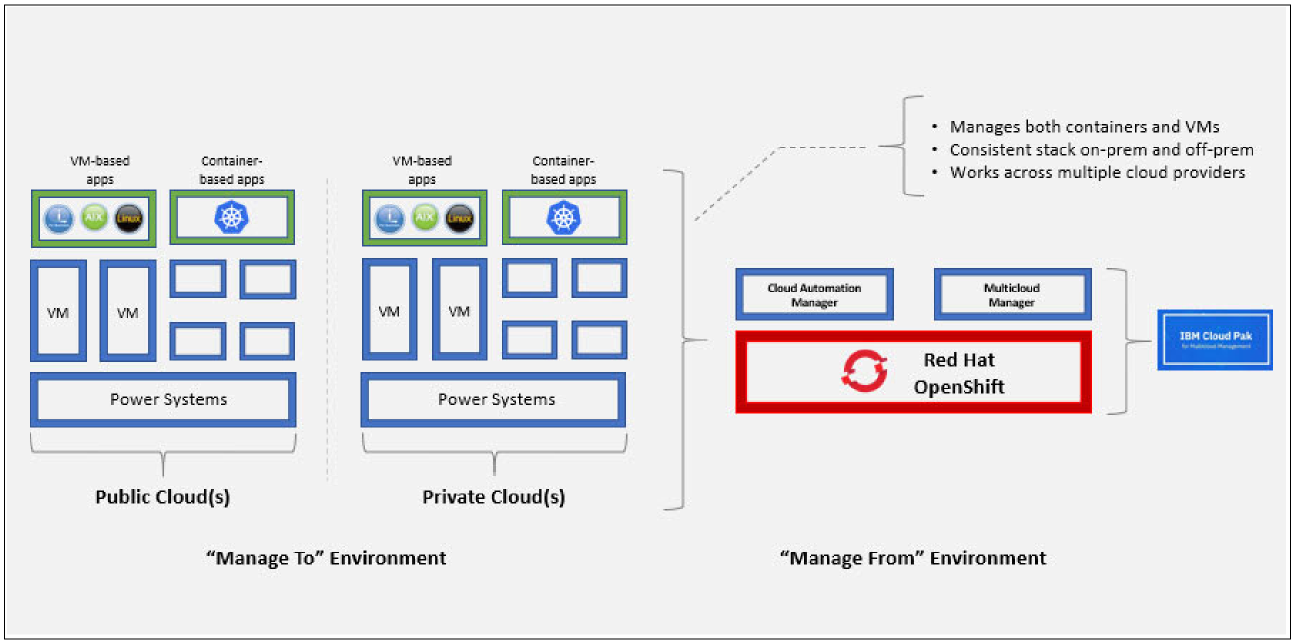


Figure 2-7 Hybrid multicloud scenario

Manage-To

In the Manage-To side of Figure 2-7, you see the hybrid IT that includes the foundation, platforms, apps, and tools from different vendors. Because proprietary tools do communicate with each other, you need advancements in infrastructure, management, and development that bring your clouds together.

Manage-From

In the Manage-From side of Figure 2-7, you see Red Hat OpenShift, Multicloud Manager, and Cloud Automation Manager (CAM) management components.

Multicloud Manager is the enterprise-grade multicloud management solution for Kubernetes. It provides consistent visibility, governance, and automation across the hybrid multicloud infrastructures. It also provides a single source to provision applications and enforces access and security policies across heterogeneous cloud environments.

Whether your enterprise application is a modernized traditional application or a cloud-native, 12-factor application, Multicloud Manager provides a consistent way to deploy your application across clusters. Placement policy provides control of deployment that is based on several factors. Multicloud Manager uses CAM services to provision, configure, and deliver individual Kubernetes clusters as a service in any cloud that CAM supports.

Multicloud Manager and CAM are offered as part of the IBM Cloud Pak™ for Multicloud Management. For more information, see Chapter 3, "IBM Cloud Paks: Middleware anywhere" on page 37.

2.3 Containers

Interconnected applications and services are the primary focus in 21st century. Services availability is a key user expectation. Availability, scalability, resilience, and security are some of the primary requirements for all service providers. All of these features demand platform availability where operational overhead is a major challenge.

Operating system virtualization (see Figure 2-8) is a significant step to achieve optimal resource utilization with reduced operation management effort. Container services are a step further to ensure higher availability of services with minimal operation management overhead. Containers are facilitating rapid and agile software development, testing, and deployment.

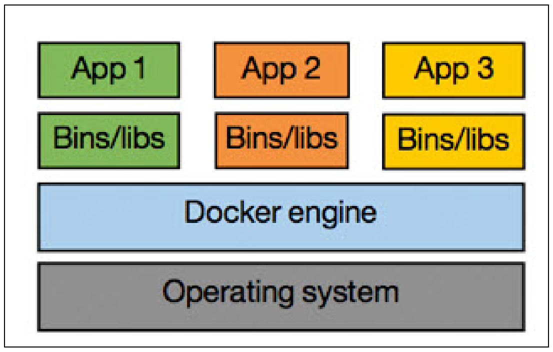


Figure 2-8 Containerized applications

2.3.1 What are containers?

A container is the object for encapsulating or packaging up software code, along with all required dependencies, so that it can run uniformly and consistently on any infrastructure. This abstraction away from the operating system makes the container highly portable to run across any platform or cloud and clear of any issues. A container is also termed as *lightweight*, considering that the same operating system kernel is shared across multiple instances of containers.

A portability feature is also known as *Write once and run anywhere*, which is important in context to development process and compatibility with other dependent software stack. Containerization improved security and efficiently of code development with the adoption of Docker and DevOps.

Containers include the following key features and benefits:

- Portability:
  - Single executable package with all code, configuration files, dependencies, and required libraries
  - Bundle must not include operating system-related files
  - Open runtime engine is a prerequisite
  - Common bins and libraries can be shared across multiple containers
- Agility:
  - Container system is managed by Open Container Initiative
  - DevOps tools and process are used for rapid code deployment by using continuous integration and continuous deployment (CI/CD)
  - Open Source Docker engine works for Linux and Windows platforms
- Performance:
  - Multiple containers share operating system kernel for lightweight execution mode
  - Improves service usage, which results in reduced software license costs
  - Container start time is much faster than VM start time
- Fault isolation:
  - During concurrent execution, each container runs independently. A fault in one container does not affect other container's execution.
  - Container engine takes advantage of operating system security isolation technique.
- Ease of management:
  - Container orchestration manages installation, scalability, availability as defined
  - Application version upgrade, monitoring, and debugging managed centrally through container orchestration system
- Security:
  - Encapsulation and isolation is the first level of security for any containerized application. A rogue application does not affect other applications of the hosting environment
  - Container engine inherits default security features from hosting platform
  - Namespace provides an isolated view; for example, file system, mount point, network, process ID, and User ID

2.3.2 History of containers

Container seems a latest buzzword with cloud technology. However, a similar concept was used for the first time as early as 1970 where application code was decoupled from UNIX native system calls.

Overtime, a few more enhancements were made, from stand-alone computer systems to integrated environments. Then, virtualization features evolved with LPAR and workload partition (WPAR) concepts. Although these virtualization features added flexibility, application portability always was a challenge. With containerization, new age software developers received all of the flexibility, portability, and security that they needed.

The following timeline highlights the major shifts in the development of container to date (see Figure 2-9):

- 2000 FreeBSD Jails: FreeBSD Jails enabled Computer systems to be partitioned into multiple servers that were independent subsystems named Jail with unique IP address.
- 2001 Linux Vserver: Similar to FreeBSD Jails, Linux also developed a feature for operating system virtualization where a file system, memory, and network can be shared among independent systems.
- 2004 Solaris Containers: Solaris Containers combined system resource controls and boundary separation that was provided by zones to take advantage of features, such as snapshots and cloning from ZFS.
- 2006 Google process containers: Process Containers was designed for limiting, accounting, and isolating resource usage (CPU, memory, disk I/O, and network) of a collection of processes. Later, this was renamed as Control Groups (cgroups) and merged to Linux kernel 2.6.24.
- 2008 LXC evolved (Linux Container Group): Linux Containers (LXC) was the first, most complete implementation of Linux container manager. It was implemented in 2008 by using cgroups and Linux namespaces.
- 2013 Let Me Contain That For You (LMCTFY): Let Me Contain That For You (LMCTFY) started in 2013 as an open source version of Google's container stack. Applications can be made container aware, which creates and manages their own subcontainers.
- 2013 Docker: Docker emerged, which made container service even more popular. Docker and container grew together.
- 2016 Security and DevOps: Container security enhanced and DevOps method evolved as most preferred Container Application process.
- 2017 Container becomes more matured with CNCF and Kubernetes.

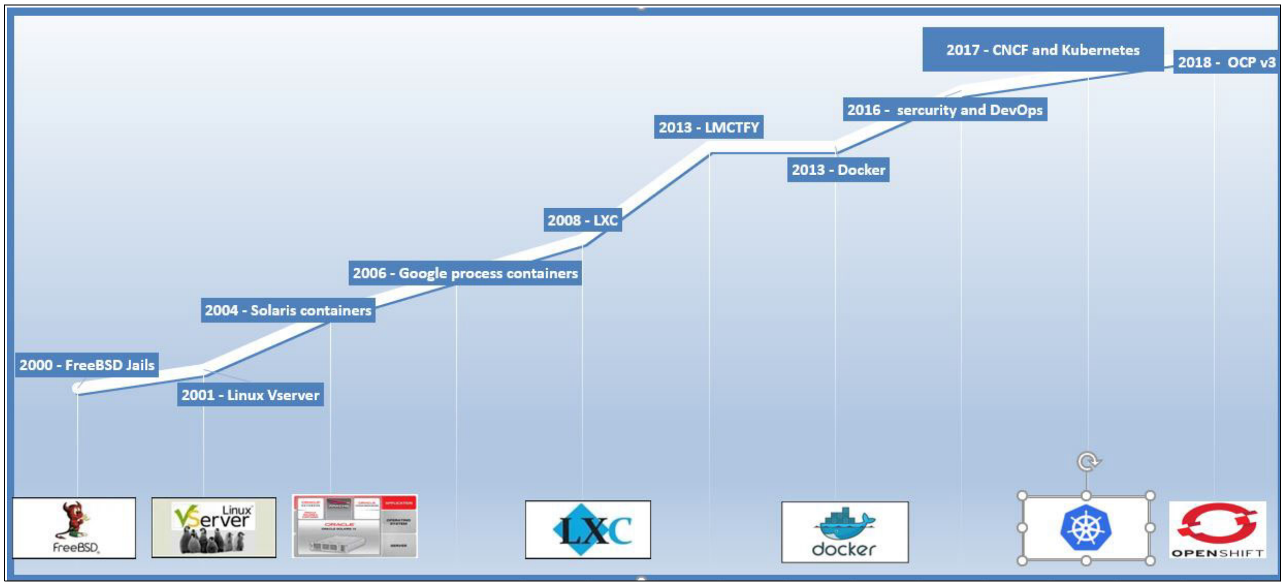


Figure 2-9 Containers timeline

2.3.3 Docker as a container manager

Docker is the virtualized platform to host containerized applications, starting from software development until execution. Docker engine is a client/server application that features the following components:

- Daemon: These processes run in the background and are not attached to any terminal session. Mostly, these processes are designed to receive instructions from other applications to perform specific tasks. Daemons plays a key role to manage Docker objects, such as images, containers, networks, and volumes.
- REST API: These user interfaces are used in code to send instructions to Daemon jobs.
- CLI: These commands are run in terminal sessions to send instructions to Daemon. Script-based batch jobs are created by using CLI.

Docker Containers optimize software development process with following objectives:

- Streamline the development process by using standardized environments of local containers, which are highly compatible for CI/CD workflow. Automation plays a significant role to build and run test scenarios.
- Docker containers are lightweight in nature, which makes it highly portable to almost every possible environment from workstation to virtualized environment to cloud. This feature makes it highly scalable based on user requirements.
- Docker requires minimal resources so that multiple Docker containers can be hosted on the same hardware for optimal performance and resource usage.

2.3.4 Docker architecture

This section describes the Docker architecture, as shown in Figure 2-10.

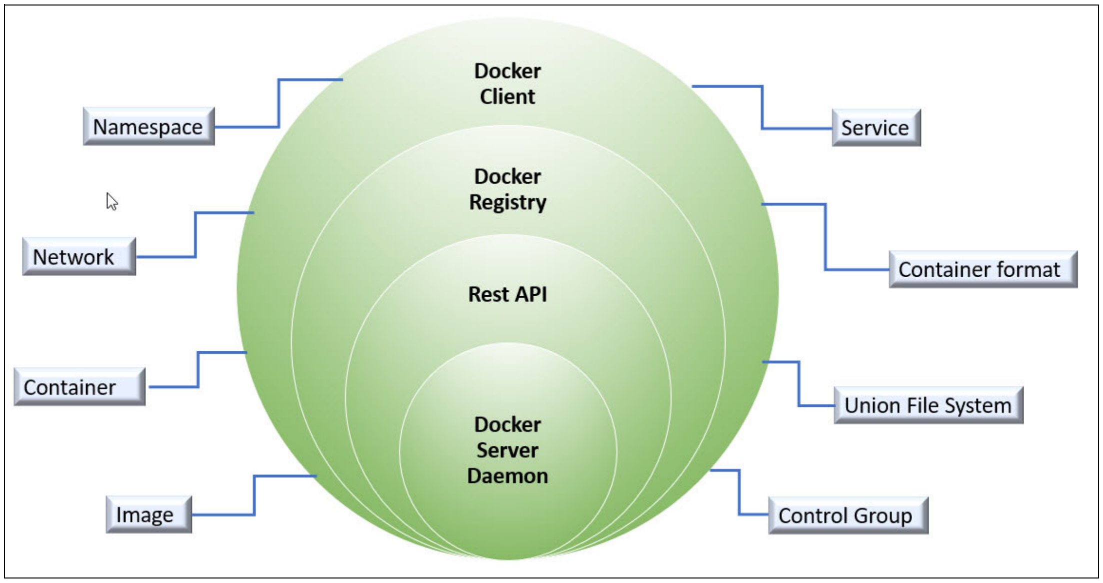


Figure 2-10 Docker architecture

The Docker architecture includes the following components:

- Docker Server Daemon

  Daemon is the Docker process that runs as background process and listen for API requests. It also manages Dockers objects, such as images, containers, networks, and volumes.

- Docker Registry

  A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use. Docker is configured to look for images on Docker Hub by default.

- Docker Objects:

  – Images: This template is read-only with instruction to build a Docker container. An image can be layer on another image with specific changes. An images library is available from the Docker registry.

    A Dockerfile contains the configuration information that is needed to build and run an image. Based on instructions that are defined in the Dockerfile, layers are created for an image. During the build process, only the changed layer is rebuilt; therefore, Docker remains lightweight, which makes it small and fast.

  – Container: A container is an executable instance that is built from the image, which can be started, stopped, moved, or deleted by using Docker API or CLI. Containers can be connected to one or many networks. Storage can be added and a new container can be built by using a container.

  – Services: Services allow you to scale containers across multiple Docker daemons, which all work together as a swarm with multiple managers and workers. Each member of a swarm is a Docker daemon, and the daemons all communicate by using the Docker API.

  – NameSpace: In context of Docker, NameSpace is the technology that provides isolated workspaces for a container (see Table 2-1). Each container encapsulates all its features within the namespace that is associated with that specific container.

Table 2-1 NameSpace

Namespace	**Description**
PID	Process isolation (PID: Process ID)
NET	Managing network interfaces (NET: Networking)
IPC	Managing access to IPC resources (IPC: InterProcess Communication)
MNT	Managing file system mount points (MNT: Mount)
UTS	Isolating kernel and version identifiers. (UTS: UNIX Timesharing System)

  – Control groups: A control group (cgroup) is the technology that limits an application to a specific set of resources. This feature allows Docker Engine to share available hardware resources to containers and optionally enforce limits and constraints.

  – Union file system: Union file systems (UnionFS) are file systems that operate by creating layers, which makes them lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers.

  – Container format: Container format is the wrapper around NameSpaces, control groups, and UnionFS. The default container format is libcontainer.

- Docker Client

  A Docker client is primary interface with which the user can use Docker features, as shown in Figure 2-11. Docker client starts APIs in this process. Developers build applications by using Docker APIs.

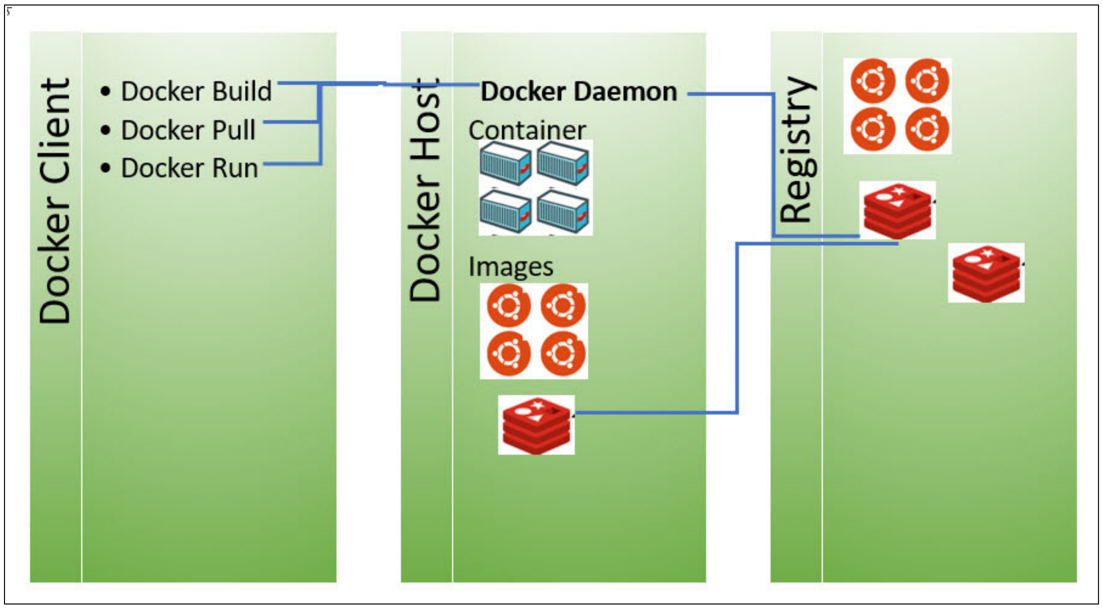


Figure 2-11 Docker orchestration

2.4 Kubernetes: An open source container orchestration

This section describes Kubernetes open source container orchestration.

2.4.1 What is container orchestration?

Container orchestration is the process of organizing properly to achieve the wanted performance. Cloud-based applications are intended to be hosted on several commodity hardware on several hardware environments. These loosely coupled containerized objects must be organized and coordinated to meet functional requirement, such as starting and stopping an application, and grouping and coordinating applications in a cluster. Some of the most popular orchestration services are Apache Mesos, Google Kubernetes, and Docker Swarm.

A container platform that is lead by Docker is used to package applications that were divided into micro services. Such discrete services can be hosted on separate containers that are helpful during the continues integration and continues delivery process. Container orchestration is primarily focused on managing the lifecycle of containers for automated deployment, management of nodes, scalability and availability of services based on work load, and networking among distributed containers in large systems.

Container orchestration configuration is created in YAML or JSON files. Based on the declarative configuration, container tools perform one or more of the following tasks:

- Fetch required configuration image from repository by way of Docker Hub
- Establish networks across container
- Allocate storage and space

- EnsurerRedundancy of container
- Manage load balancing
- Log and monitor services

2.4.2 Kubernetes architecture, system, and components

Kubernetes is highly portable with open source development and design that is heavily influenced by Google's Borg system. Kubernetes and DevOps grew simultaneously and appeared as the leading self-serviced Platform as a Service (PaaS). The following components the Kubernetes architecture are shown in Figure 2-12:

- Node: A logical collection of system resources to support containers. Node contains all required services to run Pods.
- Cluster: A collection of nodes with at least one master node and several worker nodes.
- Kubernetes master: Through the API server, the Kubernetes master communicates with nodes in which application and containers are deployed. Several services are hosted from containers. The scheduler assigns nodes to Pods based on policies and constraints that are defined in a control plan.
- Kubelet: This agent process runs on each node. This process tracks the status of the node and other configuration parameters from the Control Plan. This information is fetched from the API server.
- Pods: These individual objects can be run or scheduled as a stand-alone service. Each pod can have one or more containers and features its unique IP within the cluster. Therefore, each pod simulates an independent hosting environment where load balancing is achieved by using the same service that can be hosted by using the same port. Pod-specific configurations are placed in PodSpec by using YAML or JSON. PodSpec communicates with Kubelet by using the API server.
- Deployments: A deployment is a YAML object that defines the Pods and the number of container instances (called *replicas*) for each Pod.
- Replicas and ReplicaSets: The number of replicas that you want to have running in the cluster are defined by using a ReplicaSet, which is part of the deployment object.

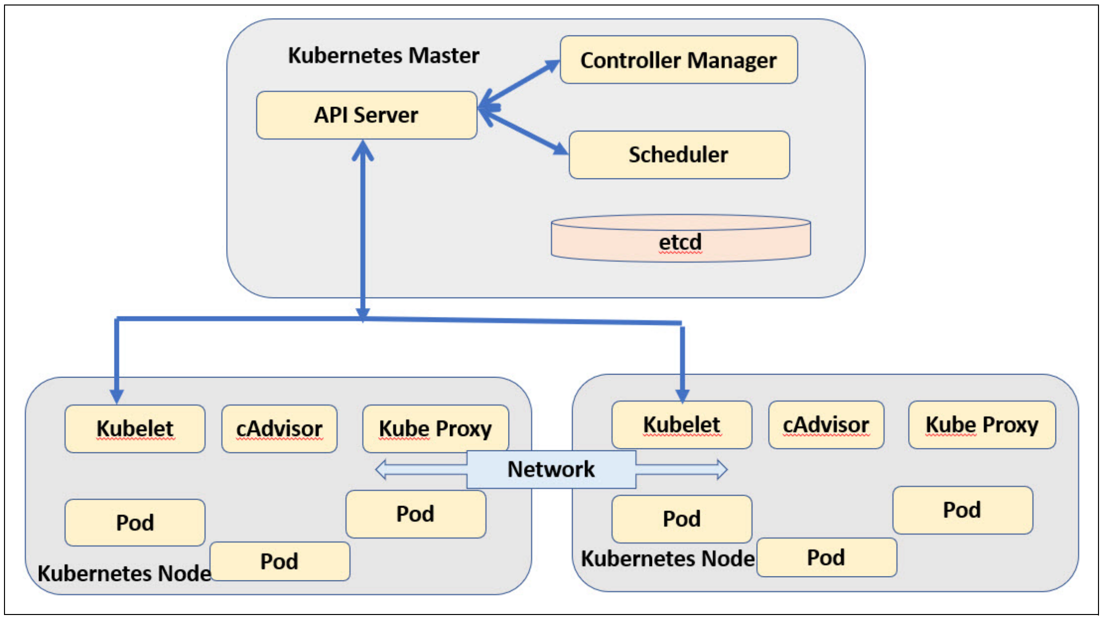


Figure 2-12 Kubernetes architecture

2.4.3 Kubernetes operating environment, objects, and basic operations

This section describes the Kubernetes operating environment, including its objects and basic operations.

Master node

This node runs multiple controllers that are responsible for the health of the cluster, replication, scheduling, endpoints (linking Services and Pods), Kubernetes API. It interacts with the underlying cloud providers and others. Generally, it ensures that everything is running and monitors worker nodes.

Worker node

This node runs the Kubernetes agent that is responsible for running Pod containers by way of Docker or rkt, requests secrets or configurations, mounts required Pod volumes, performs health checks, and reports the status of Pods and the node to the rest of the system.

Pod

Within a cluster, a pod encapsulates an application that is composed of one or more processes from one and at time multiple containers. Every pod includes dedicated I/O resources, such as storage, a unique IP, and a set of configuration properties for the runtime environment. These features make pod the smallest unit of deployment and basic unit of execution.

Docker is the most popular container run time that is used for Kubernetes Pod[1]. Depending on associated containers, pods are available in the following types:

- Pod with a single container: This configuration is the most common.
- Pod with multiple containers: Must be colocated containers to serve a functional requirement.
- Networking: Each pod shares its namespace, IP, and port. However, for optimal performance, containers in same Pod communicates with the localhost identity.
- Storage: A pod specifies shared storage volume. All containers in a pod can share persistent data through this volume.

After a pod is created and is scheduled to run on a node, it persists until one of the following actions occurs:

- The process is ended.
- The pod objected is deleted.
- The pod is evicted for lack of resources.
- The node fails.

A pod alone is not self-healing, which means that during any failure, a pod does not attempt to restart. A pod is the encapsulation of containers, which primarily are executable entities. Therefore, to “run a pod” means running an application and service through containers.

[1] https://www.sumologic.com/blog/kubernetes-vs-docker/

Services

Kubernetes Service is one more REST object and is similar to a pod. However, this service is designed as an abstraction for logical set of pods and associated policy to access those sets. This abstraction enables cloud-based applications to be decoupled from its endpoints. A Service definition is posted to API server to create a Service instance with or without Selector definition. Cloud native applications take advantage of the API server to query and discover Service endpoints, along with notification for changes in pods that are associated with a service.

Service with selector

Service with selector includes the following features:

- An object is created with cluster IP, which is assigned by Kubernetes to new a Service object
- Automatically creates endpoints
- Controller scans and find this service with the reference of Service selector

Service without selector

Service without selector includes the following features:

- Maps with objects from external clusters
- Maps with objects from different namespace
- During phased migration, section of objects from be run as a backend process
- Endpoint objects are not created automatically (requires manual mapping of service with IP and ports)

Storage volume

Kubernetes volume is an abstraction to address two important aspects, such as data persistency and sharing. Data persistency helps to retain the most updated data set after a container is stopped or crashed. Data sharing can be an essential requirement if multiple containers are hosted in a pod.

A Kubernetes volume is persistent until the enclosing pod is available. A pod can select or move volumes from the following persistent block level storage devices:

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- cinder
- configMap
- csi
- downwardAPI
- emptyDir
- fc (Fibre Channel)
- flexVolume
- flocker
- gcePersistentDisk
- gitRepo (deprecated)
- glusterfs
- hostPath
- iscsi
- local
- nfs

- persistentVolumeClaim
- projected
- portworxVolume
- quobyte
- rbd
- scaleIO
- secret
- storageos
- vsphereVolume

NameSpace

Kubernetes NameSpace is set of virtual clusters that are backed by physical clusters. NameSpace is intended to divide cluster between users to provide an isolated and secure environment. NameSpaces are built with unique names. But the same name can be present in different NameSpaces. Therefore, a Name is supposed to be unique in a NameSpaces, but not so when it encounters NameSpaces. A NameSpace must not be nested.

Controller

Controller is another Kubernetes abstraction that handles the availability of a pod. Therefore, the most preferred practice is to use the Controller to manage pods. A Controller can create and manage multiple pods for you. It also can handle replication and rollout and provide self-healing capabilities at the cluster scope. For example, if a node fails, the Controller might automatically replace the pod by scheduling an identical replacement on a different node.

ReplicaSet

ReplicaSet is the process to maintain the configured availability level with a specified number of identical pods in the Kubernetes Cluster. A ReplicaSet must be created with a selector and the template to include specifications for pod instances to be created or deleted to meet the needed availability level. This task is performed by ReplicationController. A ReplicaSet identifies new pods to acquire by using its selector.

Deployment

Kubernetes nodes are managed over declarative configuration to ensure the availability of the wanted number of pods or ReplicaSets. At times, a deployment also can be rolled back based on business requirements.

StatefulSet

StatefulSets are API objects that are used to manage stateful application for deployment and ensures the ordering and uniqueness of pods. In this scheduling option, pods carry a unique identity, although it starts with identical pods. This scheduling can be helpful for the following aspects of an application:

- Persistent unique network identifier and storage
- Ordered deployment, scaling, and automated rolling updates

DaemonSet

A DaemonSet is another API server object that ensures scalability of a pod with the change of available nodes in the system. After a node is added to the cluster, the corresponding pod is initialized in that node. It also cleans up the pod after the node is deleted.

A DaemonSet often is used for the following tasks:

- Cluster storage daemon on each node
- Logs collection daemon on every node
- Node monitoring daemon on every node

The use of one or multiple DaemonSet depends on the complexity of the cluster and application.

Job

A Job creates one or more pods and ensures that a specified number of those pods successfully end. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (for example, Job) is complete. Deleting a Job cleans up the created pods.

A simple case is to create one Job object to reliably run one pod to completion. The Job object starts a new pod if the first pod fails or is deleted (for example, because of a node hardware failure or a node restart).

Control plan

Pods do not, by themselves, self-heal. If a pod is scheduled to a node that fails, or if the scheduling operation fails, the pod is deleted; likewise, a pod does not survive an eviction because of a lack of resources or node maintenance.

Kubernetes uses a higher-level abstraction, called a Controller, that handles the work of managing the relatively disposable pod instances. Although it is possible to use pods directly, it is far more common in Kubernetes to manage your pods by using a Controller. For more information about how Kubernetes uses Controllers to implement pod scaling and healing, see “Pod” on page 26 and “Controller” on page 28.

2.4.4 Cloud Native Computing Foundation

The Cloud Native Computing Foundation (CNCF) hosts critical components of the global technology infrastructure. For more information, see this website.

One of the projects under incubation to develop a Kubernetes Container Runtime interface is the Container Runtime Interface (CRI) for OCI (CRI-O). This initiative is explained in this section, including its architecture, benefits, and distinguishing features.

CRI-O

CRI-O is a project that was started by Red Hat in 2016 to be an open Container Initiative (OCI) that is based the implementation of Kubernetes CRI. This effort contributed to the Cloud Native Computing Foundation (CNCF). CRI-O and Kubernetes follows the same release cycle and dependencies.

CRI uses the OCI-compatible environment for running Pods. CRI-O's compatibility with OCI enables it to be pulled in from any Container registry. This feature is a lightweight alternative for Docker and similar services. CRI-O is available with Red Hat OpenShift.

The CRI-O architecture is shown in Figure 2-13.

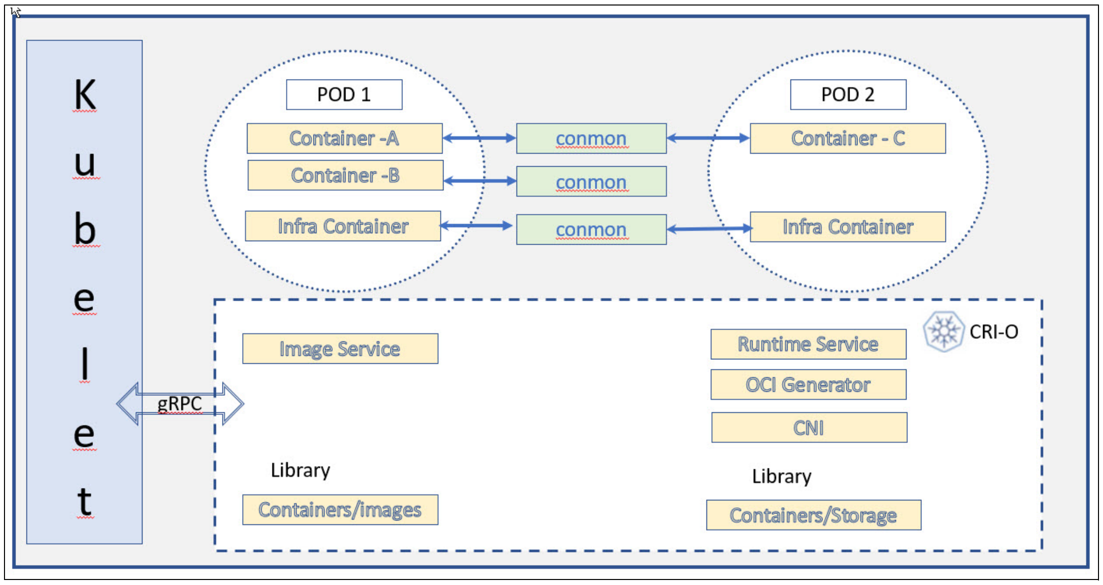


Figure 2-13 CRI-O architecture

The architecture overview that is shown in Figure 2-13 includes the following components:

- Pods are managed by a Kubelet that is based on information that is received from Kubernetes. The Kubelet and CRI-O communicate over CRI-gRPC API. Each pod hosts one or more containers from same cgroup and share common resources that are available in Pod.CRI-O uses CIN plug-in for network setup for the pod.
- CRI-O daemon starts a new pod that is based on Kubelet instructions that are received by way of Kubernetes CRI, which pulls images from the Container registry.
- The Container library unpacks the downloaded image in the Container root file system and stores it in the COW file system.
- After `rootfs` is created for container by using OCI generate tools, CRI-O creates JSON files with a runtime specification for the pod. The OCI-compatible run time is started based on the runtime specification. The default OCI run time is `runc`.
- Containers are monitored by using a common process that handles logging for the container process until the exit code ends the process.
- GitHub repository for several CRI-O components provides the following components:
  - OCI compatible run time
  - Containers and storage
  - Containers and images
  - Networking - CNI
  - Container monitoring (common)

Some key distinguishing features for CRI-O: CRI-O limits its scope to Kubernetes. New features are also added based on the requirement of Kubernetes only. Primary driving factors include stability, security, and performance. CRI-O has limited troubleshooting capability and it uses Kubernetes' CRI-API for Container-building activities.

2.5 Enterprise Kubernetes: Red Hat OpenShift

OpenShift is an open source container application platform by Red Hat that is based on the Kubernetes container orchestrator.

2.5.1 Red Hat OpenShift overview

Based on industry standards, such as Docker and Kubernetes, Red Hat OpenShift is one of the most reliable enterprise-grade containers. It is designed and optimized to easily deploy web applications and services. Categorized as a cloud development Platform as a Service (PasS), OpenShift allows developers to focus on code. It also manages all of the complex IT operations and processes.

One of the main features that OpenShift offers to the industry is the ability to handle applications that are written in different languages, such as Python, Node.js, Java, and Perl.

Red Hat OpenShift is presented to the users in three different products:

- Red Hat OpenShift Container Platform
- Red Hat OpenShift Online
- Red Hat OpenShift Dedicated

Red Hat OpenShift Online and Dedicated versions are clusters that are fully provided as a cloud service. They offer many features, such as high availability, flexible authentication options, and integrated container registries.

The differences between Online and Dedicated options are on the cloud service implementation. Online Service is a multi-tenant operating system that is designed for individual developers to quickly gain access to a hosted OpenShift environment, although the Dedicated version offers clusters in a virtual private cloud, which is exclusive for a single customer (single-tenant).

In contrast, OpenShift Container Platform works on-premises private PaaS. Although the implementation purposes and business modeling can differ from one product to another, the core of the code and functionality from the different versions are the same.

2.5.2 Red Hat OpenShift Container Platform

Formally known as a OpenShift Enterprise, the OpenShift Container Platform (OCP) is built and optimized for easily deployment of web applications and services. It provides developers with a secure and scalable operating system for their applications, which helps them to build and deploy containerized infrastructure.

The following section describes the OpenShift architecture, its components, and high availability considerations.

2.5.3 OpenShift Container Platform architecture

Designed as a layered system, OpenShift Container Platform architecture supports and manages each layer with Docker, which allows for the creation of lightweight containers. All of these containers are handled by Kubernetes on the top of the layers, which provides the cluster management and orchestration of containers on multiple hosts.

OpenShift Container Platform also integrates the following features:

- Source code management, builds, and deployments for developers
- Managing and promoting images at scale as they flow through your system
- Application management at scale
- Team and user tracking for organizing an organization with tons of developers
- Networking infrastructure that supports the cluster

Figure 2-14 shows the general architecture of OpenShift Container Platform.

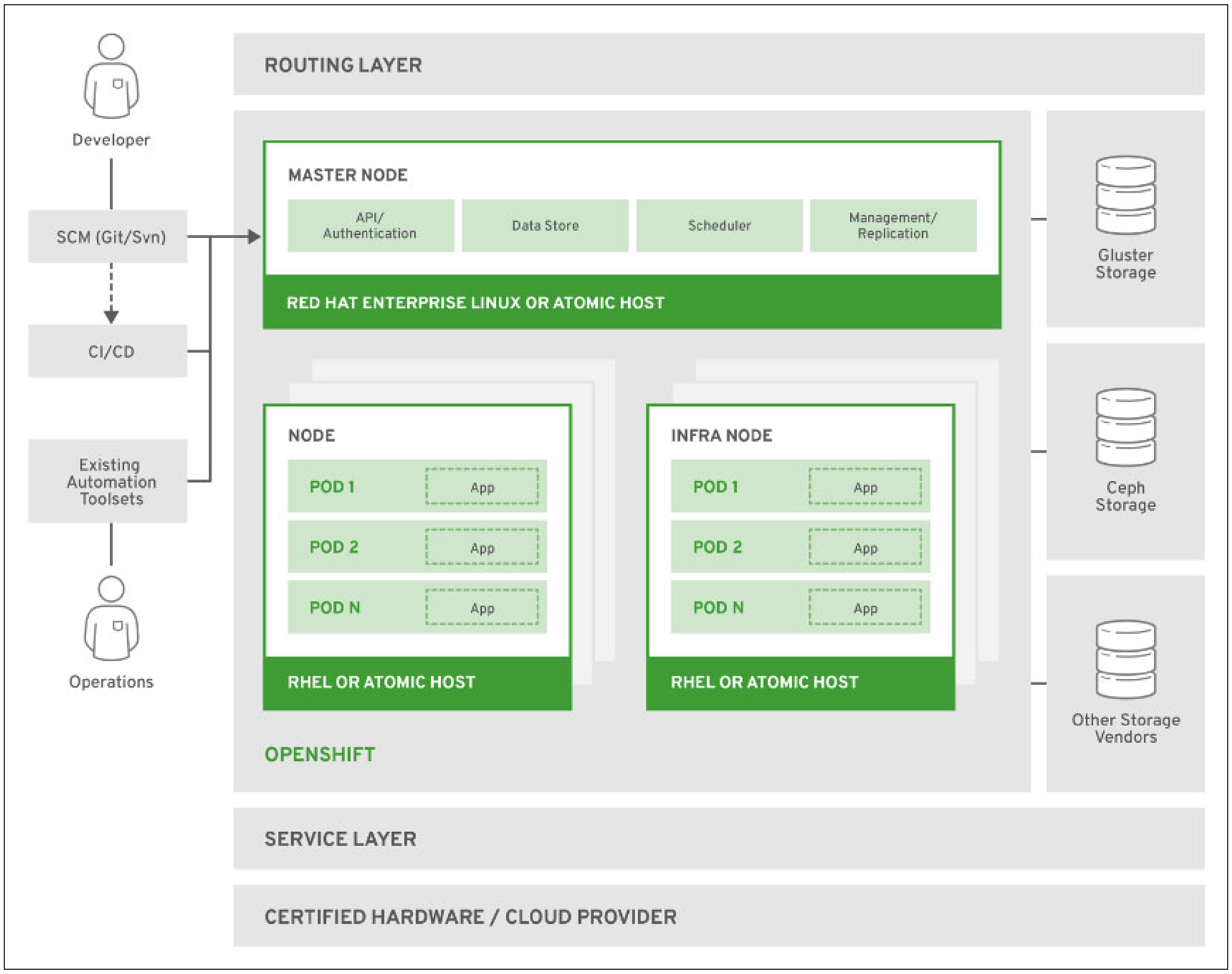


Figure 2-14 OpenShift Container Platform architecture

OpenShift Container Platform is a microservices-based architecture that is composed of decoupled and small units working together. The microservices are categorized by function: REST APIs to change the state of the system, and Controllers, which use the REST API to read the user’s wanted state, and then try to bring the other parts of the system into sync.

OpenShift Kubernetes cluster components

OpenShift is based on Kubernetes cluster. Kubernetes manages containerized applications across a set of hosts and provides mechanisms for deploying, maintaining, and application scaling.

A Kubernetes cluster consists of one or more masters and a set of nodes:

- Master

  The master is the host or hosts that contain the control plane components, including the API server, controller manager server, and `etcd`. The master manages nodes in its Kubernetes cluster and schedules pods to run on those nodes (see Table 2-2).

Table 2-2 OpenShift Master components

Component	Description
api server	The Kubernetes API server validates and configures the data for Pods, services, and replication controllers. It also assigns Pods to nodes and synchronizes Pod information with service configuration.
etcd	The etcd component stores the persistent master state while other components watch etcd for changes to bring themselves into the wanted state. Also, etcd optionally can be configured for high availability, typically deployed with 2n+1 peer services.
Controller Manager Server	The controller manager server watches etcd for changes to replication controller objects and then uses the API to enforce the wanted state. Several such processes create a cluster with one active leader at a time.
HAProxy	Optional, used when configuring highly available masters with the native method to balance load between API master endpoints. The cluster installation process can configure HAProxy for you by using the native method. Alternatively, you can use the native method, but pre-configure your own load balancer of choice.

- Service Proxy

  Each node also runs a simple network proxy that reflects the services that are defined in the API on that node. This feature allows the node to perform simple TCP and UDP stream forwarding across a set of back ends.

OpenShift Container Platform creates nodes from a cloud provider, physical systems, or virtual systems. Kubernetes interacts with node objects that are a representation of those nodes. The master uses the information from node objects to validate nodes with health checks. A node is ignored until it passes the health checks, and the master continues checking nodes until they are valid.

Note: It is recommended to have more than three master nodes to provide high availability and quorum.

2.5.4 Red Hat OpenShift access and control

OpenShift 3 allows access to the system from the command-line interface (CLI), web console, or Eclipse integrated development environment (IDE). For the purpose of this book, we focus on the use of the command-line tool and the web console.

Command Line Interface tool

The OpenShift Container Platform includes a CLI that allows running pre-configured commands for managing your applications. IT also includes lower-level tools to interact with each component of your system. The **`oc`** command-line tool, also known as OpenShift CLI, is used to interact with the OpenShift and Kubernetes HTTP API(s).

For more information about CLI, see this website.

The **oc** command-line tool is verb-focused. The following base verbs are used:

- `get`
- `create`
- `delete`
- `replace`
- `describe`

These verbs can be used to manage Kubernetes and OpenShift resources. Overall, the following command groups are available:

- `basic`
- `build and deploy`
- `application modification`
- `troubleshooting and debugging`
- `advanced`
- `settings`

For more information about OpenShift Command-Line Interface, see this website.

Web console based

For the web console, the developers of OpenShift Container Platform can access their applications by a web browser to visualize and manage the content of their projects. The web console runs as a pod on the master. The static resources that are needed to run the web console are set out by the pod. Administrators can also customize the web console by using extensions, with which you can run scripts and load custom stylesheets when the web console loads.

For more information, see this website.

Figure 2-15 shows the request architecture for the Web Console.

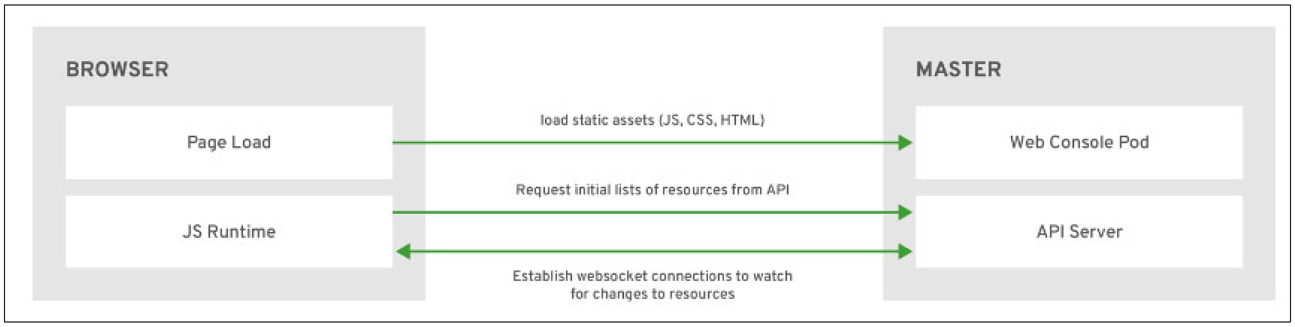

BROWSER
Page Load
JS Runtime
MASTER
Web Console Pod
API Server
load static assets (JS, CSS, HTML)
Request initial lists of resources from API
Establish websocket connections to watch for changes to resources

Figure 2-15 Web Console Request Architecture

For some operational tasks, such as the initial deployment of an application or project, the web console provides a more form-based interface, which can help you start working with the system faster.

The way to access web console is from a browser at `https://<master_public_addr>:8443`; then, the system automatically redirects to a login page, as shown in Figure 2-16 on page 35. After providing the login credentials on the login page, the user receives a token to make API calls and the system shows the projects through the web console.

For more information, see this website.

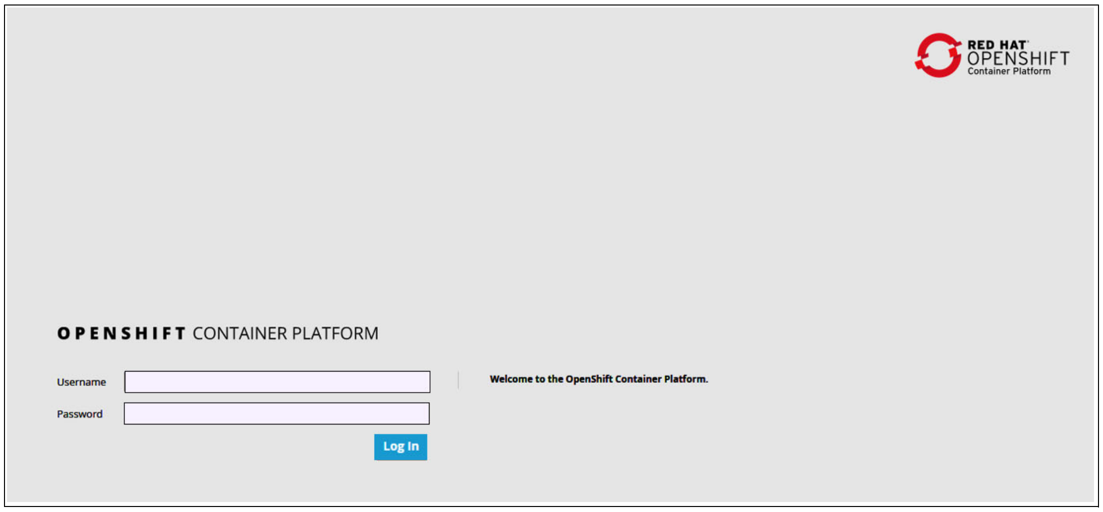

RED HAT OPENSHIFT Container Platform
OPENSHIFT CONTAINER PLATFORM
Username
Password
Log In
Welcome to the OpenShift Container Platform.

Figure 2-16   OpenShift Web Console Authentication Interface

OpenShift Container Platform includes a service catalog (see Figure 2-17), based on the Open Service Broker API (OSB API) for Kubernetes, which offers the developers an easy user experience. Also, this API allows users to connect any of their applications that are deployed in OpenShift Container Platform to various service brokers.

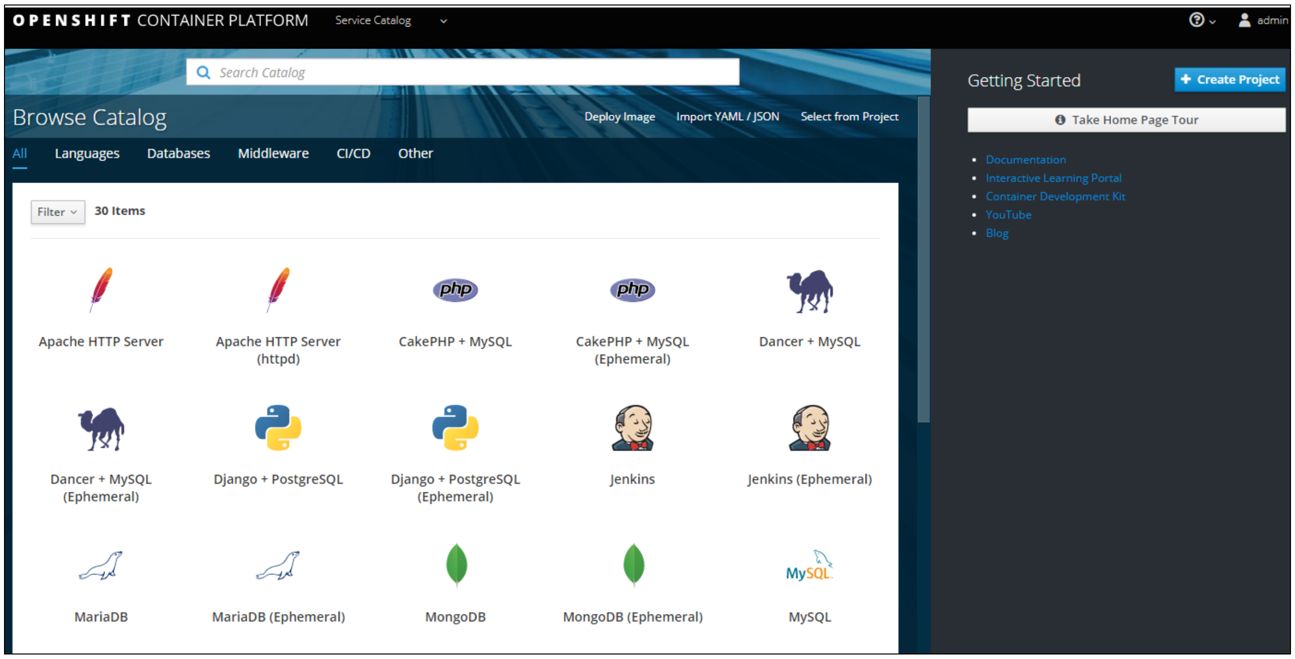


Figure 2-17   General view to the OpenShift 3.11 Service Catalog

The service catalog allows cluster administrators to integrate multiple platforms by using a single API specification. The OpenShift Container Platform web console displays the cluster service classes that are offered by service brokers in the service catalog. This display allowed users to discover and instantiate those services for use with their applications[2]

[2] Overview to the OpenShift Service Catalog:
https://docs.openshift.com/container-platform/3.11/architecture/service_catalog/index.html

3

IBM Cloud Paks: Middleware anywhere

This chapter describes IBM Cloud Paks and provides information about each Cloud Pak, including pointers to more resources.

This chapter includes the following topics:

3.1 Overview

This section introduces and describes IBM Cloud Paks.

3.1.1 What are IBM Cloud Paks?

IBM Cloud Paks are pre-integrated containers, pre-packaged solutions, that are ready for deployment in production environments. These IBM Cloud Paks can be easily deployed to Kubernetes-based container orchestration platforms. They are software solutions that give clients an open, faster, and more secure way than individual solutions to move core business applications to any cloud.

Designed to reduce development time, IBM Cloud Paks include IBM middleware and common software services for development and management, on top of a standard integration layer. They are portable and can run on-premises, on public clouds, or in an integrated system.

At the time of this writing, IBM offers six independent Cloud Paks for Power Systems that provides a Kubernetes environment to help to build cloud-native applications or modernize the existing applications. Also, IBM Cloud Paks enable you to quickly bring workloads to an integrated container platform to support production-level qualities of service and end-to-end lifecycle management.

Figure 3-1 on page 39 shows the following evolution of containers:

- Ad hoc Containers: Clients obtain IBM software binaries then create their container.
- IBM Containers: Clients obtain an individual IBM container software product.
- IBM Cloud Paks: Clients obtain a ready-to-use solution.

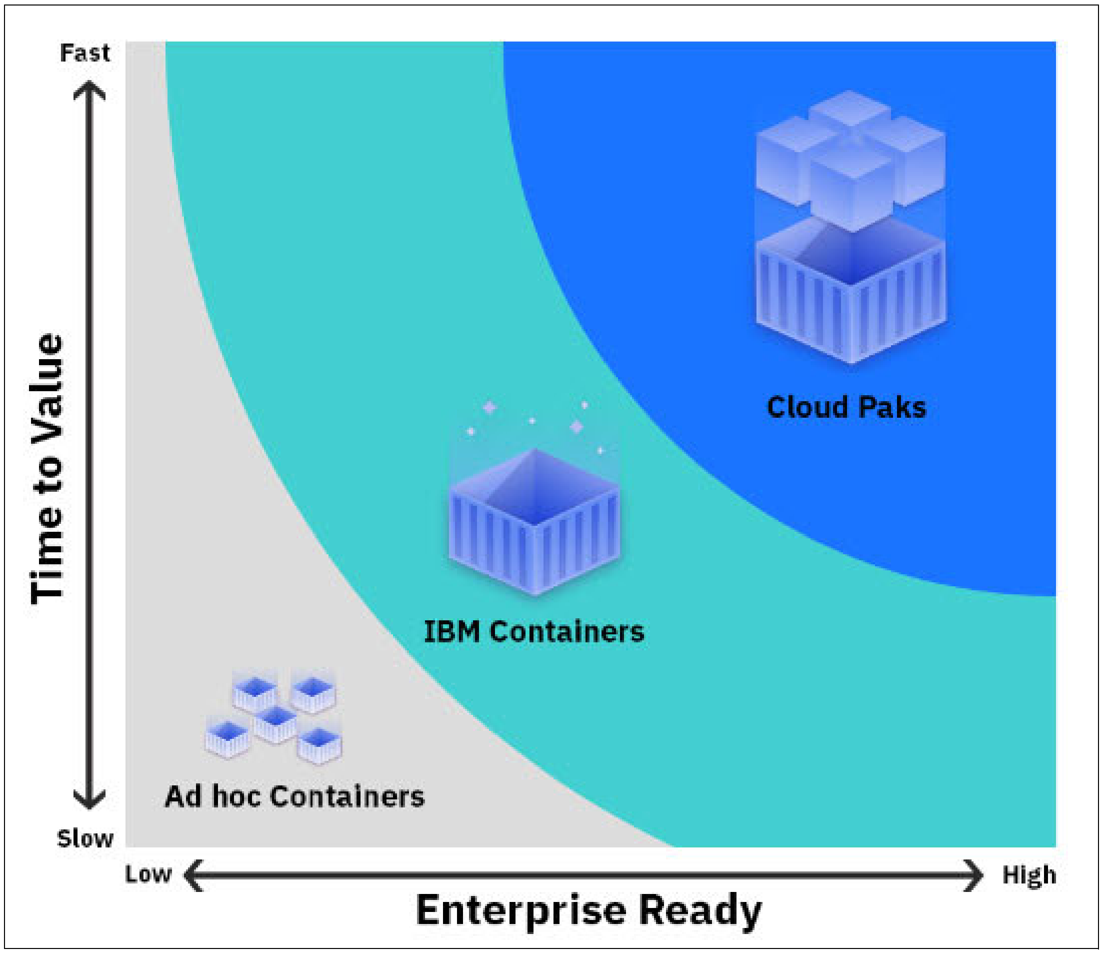


Figure 3-1 From simple containers to IBM Cloud Paks

3.1.2 First IBM Cloud Paks

IBM Cloud Paks are portable and can run on-premises, on public clouds, or in an integrated system, as shown in Figure 3-2. In addition, they are certified by IBM and Red Hat for the OpenShift Container Platform.

For more information, see this website.

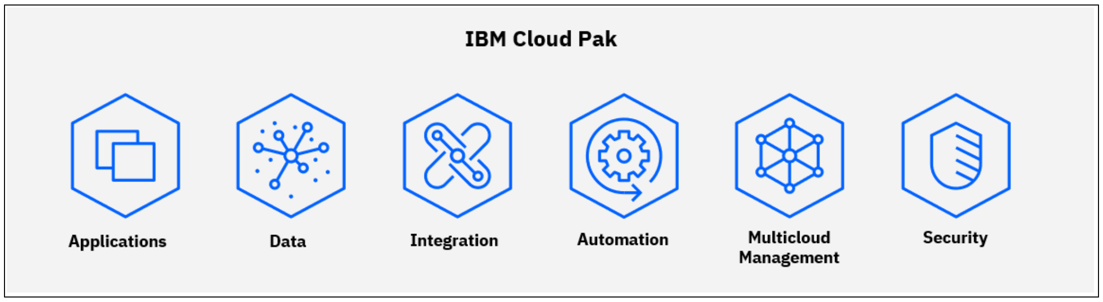


Figure 3-2 First IBM Cloud Paks

IBM Cloud Pak for Applications

IBM Cloud Pak for Applications accelerates the build of cloud-native apps by using built-in developer tools and processes, including support for microservices functions and serverless computing. Customers can quickly build apps on any cloud and at the same time, existing IBM middleware clients gain the most straightforward path to modernization.

For more information, see this website.

IBM Cloud Pak for Data

IBM Cloud Pak for Data unifies and simplifies the collection, organization, and analysis of data. Enterprises can turn data into insights through an integrated cloud-native architecture. IBM Cloud Pak for Data is extensible and easily customized to unique client data and AI landscapes through an integrated catalog of IBM, open source and third-party microservices add-ons.

For more information, see this website.

IBM Cloud Pak for Integration

This Cloud Pak supports the speed, flexibility, security, and scale that is required for all of your integration and digital transformation initiatives and features a set of preinstalled capabilities, including API lifecycle, application, data integration, messaging, events, high-speed transfer, and integration security.

For more information, see this website.

IBM Cloud Pak for Multicloud Management

This Cloud Pak provides consistent visibility, automation, and governance across a range of hybrid, multicloud management capabilities, such as event management, infrastructure management, application management, multicluster management, edge management, and integration with existing tools and processes.

For more information, see this website.

IBM Cloud Pak for Automation

IBM Cloud Pak for Automation orchestrates the deployment on your choice of clouds, with low-code tools for business users and real-time performance visibility for business managers. Customers can migrate their automation run times without application changes or data migration, and automate at scale without vendor lock-in.

For more information, see this website.

IBM Cloud Pak for Security

This platform helps you uncover hidden threats, make more informed risk-based decisions, and prioritize your team's time. You can connect it to your existing data sources to generate deeper insights, and securely access IBM and third-party tools to search for threats across any cloud or on-premises location. You also can quickly orchestrate actions and responses to those threats, while leaving your data where it is.

For more information, see this website.

3.1.3 Core Services

IBM Cloud Paks use a common set of operational services by default. These services are called Core Services (see Figure 3-3) and are a layer on top of Red Hat OpenShift, which is responsible for security and identity services, logging, monitoring, and auditing. Core Services can easily monitor workload performance and general logs by using a consistent dashboard view, regardless of the IBM Cloud Pak.

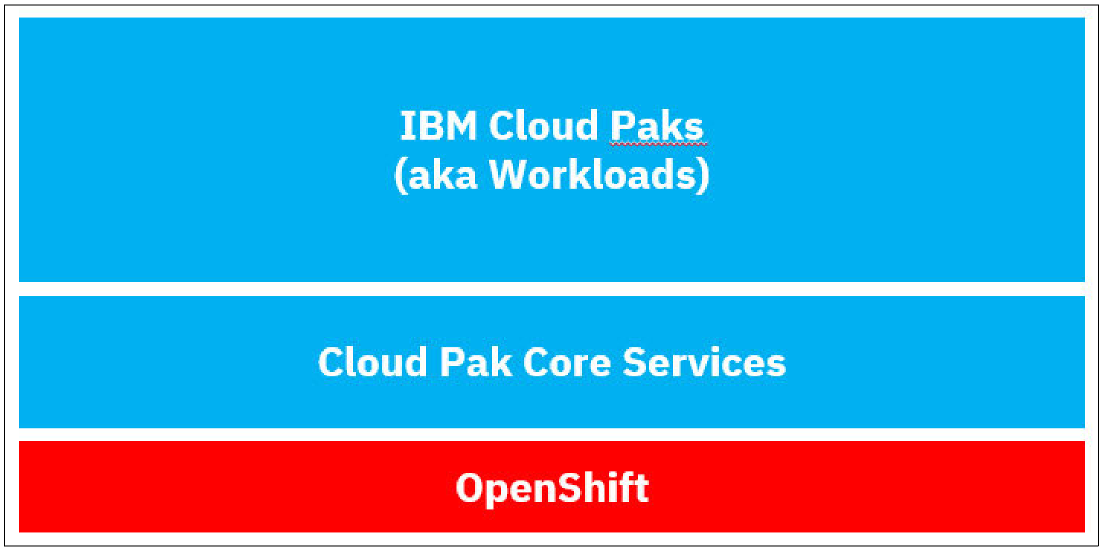


Figure 3-3 Core Services

Core Services contain a collection of services that provides essential capabilities that are needed by most enterprise applications. Red Hat tests and certifies each Core Service component to provide the necessary updates and security fixes as needed. For more information, see the following web pages:

- Red Hat Middleware Core Services Collection datasheet
- IBM Cloud computing news: What are IBM Cloud Paks?

3.1.4 Production-ready Containers Images

Containers are the key for modular cloud solutions, which allows integrating multiple vendors by isolating pieces of software so they can run independently. All IBM container images that are provided in IBM Cloud Paks follow a set of well-defined best practices and guidelines, which ensures support for production use cases and consistency across the IBM software portfolio.

IBM Cloud Paks employs Kubernetes resources to deploy, manage, and monitor the workloads. Configurations are pre-built but easily customized by using the Kubernetes operators during deployment. Upgrades can be easily rolled out or rolled back.

IBM Cloud Paks are certified by IBM and Red Hat for the OpenShift Container Platform; the container images that are included in IBM Cloud Paks are required to complete Red Hat container certification, which is complementary to IBM certification process.

3.2 IBM Cloud Pak for Applications

Enterprises must consistently update their software applications to meet the demands of their customers and users.

3.2.1 Features

IBM Cloud Pak for Applications supports the acceleration of modernizing existing and developing new cloud native applications, as shown in Figure 3-4. Running on OpenShift, IBM Cloud Pak for Applications enables quick building, testing, and deployment microservice-based applications, providing an end-to-end experience to speed developing applications by using predefined built-in developer tools and processes.

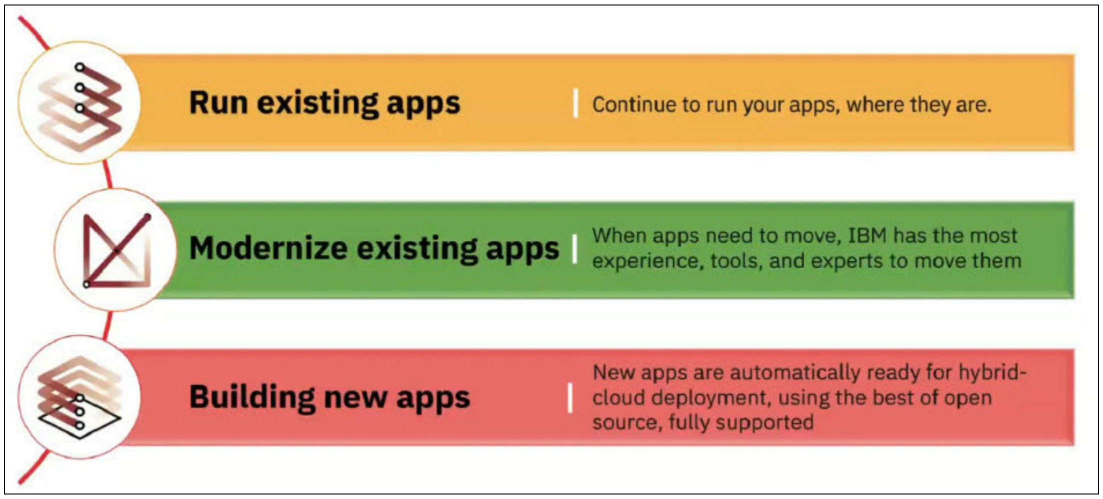

Run existing apps
Continue to run your apps, where they are.
Modernize existing apps
When apps need to move, IBM has the most experience, tools, and experts to move them
Building new apps
New apps are automatically ready for hybrid-cloud deployment, using the best of open source, fully supported

Figure 3-4 IBM Cloud Pak for Applications goals

IBM Cloud Pak for Applications is based on IBM WebSphere® offerings and OpenShift with IBM Kabanero Enterprise and provides a hybrid, multicloud foundation to enable workloads and data to run anywhere. IBM Cloud Pak for Applications also provides a self-service environment that combines open source tools with your existing middleware for continuous compliance and quick development integration (see Figure 3-5).

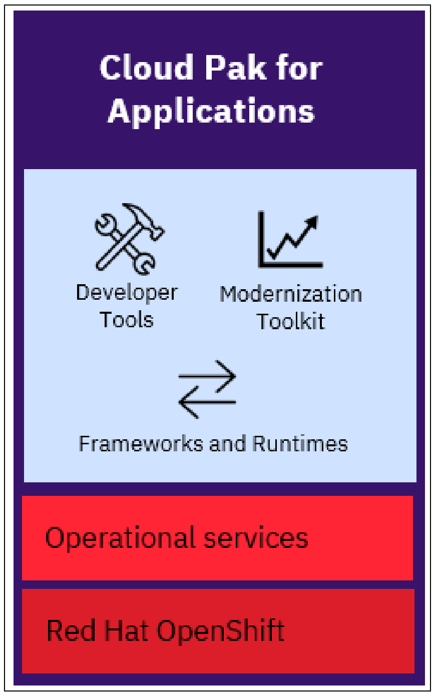


Figure 3-5 IBM Cloud Pak for Applications

The self-service portal enables the developers to download the tools and runtime executable they need for the project. Allowing them to build their application on a proven software stack that is pre-configured and tested to work with the integrated logging, monitoring, scalability, and high-availability mechanisms. Also, including DevOps practices for continuous integration and continuous delivery (CI/CD).

3.2.2 Programming Language support

IBM Cloud Pak for Applications offers support for programming languages that are designed for cloud-native, including Knative, Java programming models, Microprofile, Java Enterprise Edition, Jakarta EE and Spring, Node.js, Swift, Reactive, and CodeWind (see Figure 3-6).

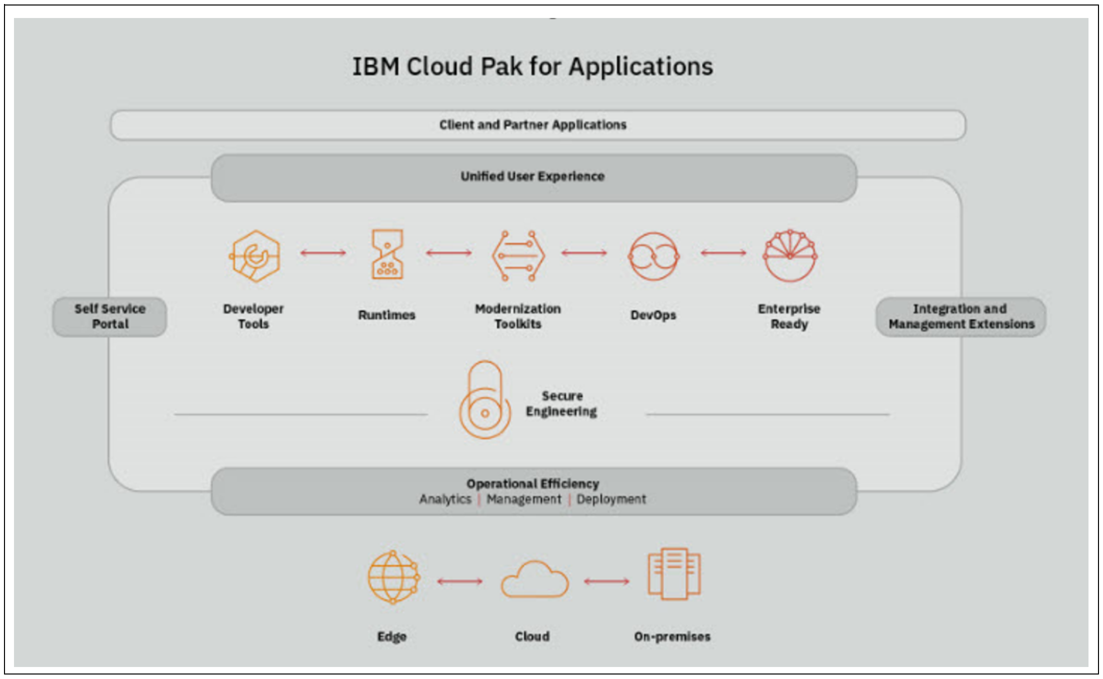


Figure 3-6 IBM Cloud Pak for Applications end-to-end

After developers start using IBM Cloud Pak for Applications, they gain the platform solutions from IBM in one bundle. You can move between WebSphere editions or to the new OpenShift with the Kabanero Enterprise container platform. You no longer must choose between private cloud-based platforms, traditional server deployments, or public cloud WebSphere deployments. IBM Cloud Pak for Applications delivers all of those platform options, the ability to mix and match as you need to today, and the flexibility to change that mix over time as you transition and create more applications for the cloud.

3.2.3 Tools and runtime packages

IBM Cloud Pak for Applications provides to developers access to the following industry-leading areas:

- Continuous delivery for hybrid clouds
- Runtime support
- IBM Mobile Foundation
- IBM WebSphere Application Server family of products

IBM WebSphere Application Server

IBM WebSphere Application Server, with its traditional and Liberty runtimes, offers production-ready, standards-based Java EE-compliant architectures. WebSphere run times in DevOps workflows make it easy to integrate WebSphere into modern toolchains and DevOps implementations. It enables secure, flexible, and efficient access to internal or external software components and services.

IBM Cloud Pak for Applications includes the WebSphere Application Server Network Deployment, WebSphere Application Server, and WebSphere Liberty Core editions.

IBM Kabanero Enterprise

Kabanero Enterprise is the commercial enterprise-ready and fully supported implementation of the Kabanero open source community project. Kabanero Enterprise integrates with, extends, and adds value to Red Hat OpenShift.

Red Hat Runtimes

Kabanero offers open source technologies in a microservices-based framework that simplifies development, build, and deployment of applications for Kubernetes and Knative (serverless).

Red Hat Runtimes provides a set of open runtimes, tools, and components for developing and maintaining cloud-native applications. It offers lightweight runtimes and frameworks for highly distributed cloud architectures, such as microservices.

IBM Application Navigator

IBM Application Navigator is a tool that helps you visualize, inspect, and monitor the deployed resources in applications, with a single view across hybrid deployments. Application Navigator extends the open source Kubernetes Application Navigator (kAppNav) with integrated support for WebSphere Application Server Network Deployment and Liberty.

IBM Cloud Transformation Advisor

IBM Transformation Advisor helps you plan, prioritize, and package your on-premises workloads for modernization on IBM Cloud Pak for Applications. Transformation Advisor gathers preferences about your on-premises and wanted cloud environments and then analyzes existing middleware deployments by using a data collector.

After you upload the results of the data collector, you can review recommendations for migrating your applications to different cloud platforms and the estimated effort to migrate and modernize. Transformation Advisor also creates necessary deployment artifacts to accelerate your migration into IBM Cloud Pak for Applications.

IBM Mobile Foundation

IBM Mobile Foundation offers a secured platform for developers to rapidly build and deploy the next generation of digital apps, including mobile, wearables, conversation, web, and Progressive Web Apps (PWAs). With Mobile Foundation, developers get containerized mobile back-end services covering comprehensive security, application lifecycle management, push notifications, feature toggle, offline sync, and back-end integration. The platform also includes a low-code studio, private Appstore, and rich SDKs for widely used mobile frameworks for native and hybrid developers.

3.3 IBM Cloud Pak for Automation

IBM Cloud Pak for Automation is a pre-integrated set of essential software that enables clients to easily design, build, and run intelligent automation applications at scale. With IBM Cloud Pak for Automation, clients can deploy on their choice of clouds anywhere Kubernetes is supported by low-code tools for business users and real-time performance visibility for business managers. It is one flexible package with simple, consistent licensing, and ensures no vendor lock-in. Also, existing customers can migrate their automation run times without application changes or data migration.

3.3.1 Features

IBM Cloud Pak for Automation is a containerized automation software platform with pre-integrated automation capabilities, such as workflow and decision automation, content management, document processing, and operational intelligence (see Figure 3-7). This feature enables organizations to digitize all styles of work with AI-infused business-oriented tools and built-in operational analytics for visibility and governance.

This offering empowers business users to rapidly deliver applications and services at enterprise scale for greater cost savings and operational efficiencies. The IBM Cloud Pak for Automation can run anywhere (on-premises, on private and public clouds, and in pre-integrated systems) and is Red Hat OpenShift certified.

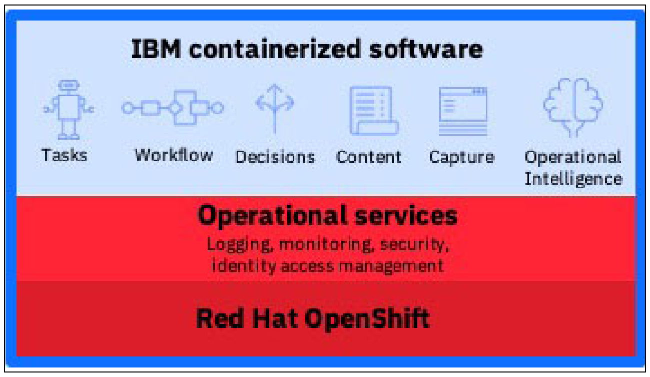


Figure 3-7 IBM Cloud Pak for Automation

Successful enterprises look for modernizing their business operations with intelligent automation lowering costs and increasing revenue, as shown in Figure 3-8.

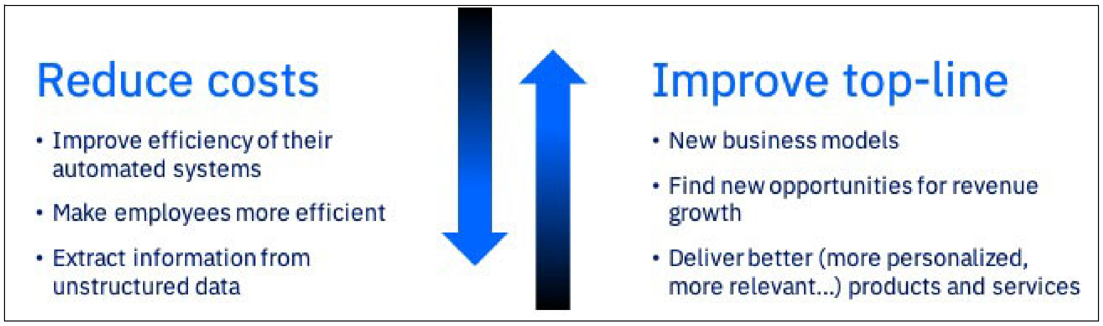

Reduce costs
• Improve efficiency of their automated systems
• Make employees more efficient
• Extract information from unstructured data
Improve top-line
• New business models
• Find new opportunities for revenue growth
• Deliver better (more personalized, more relevant...) products and services

Figure 3-8 IBM Cloud Pak for Automation benefits

IBM Cloud Pak for Automation includes the following features:

- Drive scale, speed, and assist with complex knowledge work.
- Automate complex and less-structured business processes and optimize lighter customer experiences for task efficiency.
- Drive growth with enhanced customer experiences and new business models.
- Enable a few expert employees to create great customer experiences at scale.
- Integrated automation platform.
- Measure, in real time, the value of human and automated work across your business.

3.3.2 Core capabilities

Integral to the IBM Cloud Pak for Automation is a set of containerized software, both IBM and open source, which includes the following components:

- IBM Business Automation Workflow

  Manual workflows can easily disrupt or slow operations. Lack of transparency and dependencies on employees leave businesses vulnerable to various bottlenecks that create inefficiencies. Automating workflows safeguards against potential barriers and empowers business professionals to directly participate in designing business solutions. Workflow automation orchestrates multiple business processes straight-through, human-assisted or case management within operations and provides visibility into each step.

- IBM Operational Decision Manager

  A business rules management system (BRMS) enables businesses to create and manage business logic independently from applications and processes. Through business rules, your team can specify decision logic in simple terms, close to natural language. Because rules are easily integrated with other IT systems, your applications can scale and run automated decisions across multiple channels.

  When changes to business rules are required, business users can quickly update them, which provides the agility and speed that is needed to meet changing business demands. Decision automation uses business rules to remove manual work from a decision process, which improves business agility and reducing IT reliance.

- IBM Business Automation Content Analyzer

  To reinvent under-performing, high-friction business processes, enterprises are investing in digital transformation. This investment requires processes and applications to access and control a wide range of content, including documents, images, and audio files.

  Content services are accessible in multiple ways, including mobile devices and desktops, and as discrete capabilities embedded in workflows or applications, such as Enterprise Resource Planning (ERP) systems. This content analyzer enables efficient, consistent, and accurate content collaboration and decision-making across the organization. Content services are capabilities for collecting, governing, managing, and enriching enterprise content to be deployed efficiently across any cloud and within any application.

- Process Mapping

  Inefficient processes cost you time and money. Bottlenecks, complexities, and a lack of understanding mask opportunities for process improvement. Process modeling helps you to gain better visibility into business operations, which helps you create efficiencies at scale. Process mapping is any automation strategy’s first step. It enables non-technical people to work across departments to see a process landscape.

- Data Capture

  Enterprises produce and receive massive volumes of new information every day to make decisions, manage operations, and create value. Most that information is inaccessible and invisible to the business applications that need it most, which undermines the ability of decision makers to truly understand the opportunities and constraints that are affecting their organization.

By standardizing and automating data extraction processes, enterprises can more productively and accurately extract knowledge and intelligence from unstructured content to create insights that accurately reflect operational reality, which enables more effective digital transformation initiatives and better business outcomes. Intelligent data extraction is an innovative evolution in standard data capture that extends optical character recognition (OCR), AI, and other techniques to identify and extract information from unstructured content.

- Tasks

  Automating repetitive tasks saves time and money. Robotic process automation bots expand the value of an automation platform by completing tasks faster, which allows employees to perform higher-value work. Robotic process automation (RPA) is the use of software bots to automate highly repetitive, routine tasks that are normally performed by knowledge workers.

3.4 IBM Cloud Pak for Data

IBM Cloud Pak for Data is a fully integrated data and AI platform that modernizes data engineering for organizations. Built on Red Hat OpenShift, IBM Cloud Pak for data provides containerized solution for data processing and analytics capability.

3.4.1 Features

IBM Cloud Pak for Data is composed of pre-configured microservices. The microservices enable you to connect to your data sources so that you can catalog and govern, explore and profile, transform, and analyze your data from a single web application.

IBM Cloud Pak for Data is a native cloud solution that enables data to work quickly and efficiently. Data is used to generate meaningful insights that can help to avoid problems.

IBM Cloud Pak for Data helps you do both by enabling you to connect to your data, govern it, find it, and use it for analysis. IBM Cloud Pak for Data also enables all of your data users to collaborate from a single, unified interface so that your IT department does not need to deploy and connect multiple applications.

3.4.2 Layers

This section explains the following layers that are available in IBM Cloud Pak for Data:

- Data accumulation

  IBM Cloud Pak for Data System offers, accelerated time to value, which allows you to stand up an entire cloud system for user data and AI architecture in under four hours. IBM Cloud Pak for Data System simplifies expansion. You can buy small increments of computing or storage capacity as needed, rather than outlaying vast amounts of capital. Based on application demand and delivery, budgeting can be completed in quarterly expansions on an ad hoc basis.

- Data organization

  Understanding the quality, content, and structure of your data is an important first step when making critical business decisions. It uses a reusable rules library and supports multi-level evaluations by rule record and pattern. It also facilitates the management of exceptions to established rules.

- Data analysis

  Data Analyzer helps identify data inconsistencies, redundancies, and anomalies and makes inferences about the best choices for structure. Analyzing data is the primary purpose of Information Analyzer.

3.5 IBM Cloud Pak for Integration

With time, every organization expands their IT estate with new infrastructure, software, and applications. This investment in a distributed and diverse environment is a major concern during any cloud migration.

3.5.1 Features

IBM Cloud Pak for Integration includes the following features:

- Maximize utilization: Support traditional integration with a containerized approach across delivery models, domains, endpoints, and personas. Migrate, innovate, and scale at your own pace.
- Improvement efficiency: Use built-in features, including templates, prebuilt connectors, and an asset repository to drive speed in integration development and reduce cost.
- Increased flexibility: Use various integration styles that include APIs, message queues, and emerging capabilities, including event-driven architecture and high-speed data transfer.

3.5.2 Layers

The following layers are available in IBM Cloud Pak for Integration:

- API lifecycle

  This layer is a set of new API that was created for Cloud integration and to meet some basic parameters, such as security, manageability, and continuous availability for the changing needs of business users.
- Application and data integration

  Integrate all of your business data and applications more quickly and easily across any cloud, from the simplest SaaS application to the most complex systems without worrying about mismatched sources, formats, or standards.
- Enterprise messaging

  This layer provides a reliable exchange over a flexible and secured messaging process to ensure that on demand availability of data is the primary objective from this system.
- Event streaming

  Use Apache Kafka to deliver messages more easily and reliably and to react to events in real time. Provide more personalized customer experiences by responding to events before the moment passes.
- High-speed data transfer

  Send large files and data sets almost anywhere, reliably and at maximum speed. Accelerate collaboration and meet the demands of complex global teams, without compromising performance or security.

- Secure gateway

  Create persistent, security-rich connections between on-premises and cloud environments. Quickly set up and manage gateways, control access on a per-resource basis, configure TLS encryption and mutual authentication, and monitor all of your traffic.

3.6 IBM Cloud Pak for Multicloud Management

One of the best features of containers is their ability to run anywhere considering that all of the libraries and binaries that are needed for the run time are packaged and included in the container. This feature makes it easier for a multicloud approach. With the correct multi-architecture platform, containers can spawn seamlessly on any cloud by using any architecture as the underlying infrastructure.

To better use this feature, the IBM Cloud Pak for Multicloud Management (running on Red Hat OpenShift) provides consistent visibility, governance, and automation from on-premises to the edge. Enterprises gain capabilities, such as multicluster management, event management, application management, and infrastructure management. Enterprises can use this IBM Cloud Pak to help increase operational efficiency that is driven by intelligent data, analysis, and predictive golden signals. They also gain built-in support for their Compliance Management.

The IBM Cloud Pak for Multicloud Management includes IBM Multicloud Manager, IBM Cloud App Management, IBM Cloud Automation Manager, and IBM Cloud Event Management. With IBM Cloud Pak for Multicloud Management, you get more application and cluster visibility across the enterprise to any public or private cloud. You can improve automation by simplifying your IT and application operations management with increased flexibility and cost savings, and intelligent data analysis that is driven by predictive signals.

3.6.1 Features

The main goal of this IBM Cloud Pak is to offer the advantage of a centralized governance because you can manage your multicloud environments with a consistent set of configuration and security policies across all applications and clusters.

Operations with IBM Multicloud Manager

IBM Multicloud Manager provides user visibility, application-centric management (governance, deployments, health, and operations), and policy-based compliance across clouds and clusters. With IBM Multicloud Manager, you control your Kubernetes clusters. You also ensure that your clusters are secure, operating efficiently, and delivering the service levels that applications expect.

Monitoring with IBM Cloud App Management

Monitor cloud and on-premises application environments with IBM Cloud App Management. Bridge your existing infrastructure into the cloud.

Visibility with Cloud Event Management

You can visualize and manage multiple clusters when you install Event Management. By using Event Management, you can consolidate information from your monitoring systems and address problems. Events indicate that something occurred on an application, service, or another monitored object.

All events that are related to a single application or particular cluster are correlated with an incident. Event Management can receive events from various monitoring sources (on-premises or in the cloud). Event Management is installed along with IBM Cloud App Management.

Provisioning with IBM Cloud Automation Manager

IBM Cloud Automation Manager is a cloud management solution in IBM Cloud Private that automates provisioning of infrastructure and virtual machine applications across multiple cloud environments with optional workflow orchestration.

3.6.2 Layers explained

IBM Multicloud Manager consists of several components, which are used to access and manage your clusters. A high-level architecture of the components is shown in Figure 3-9.

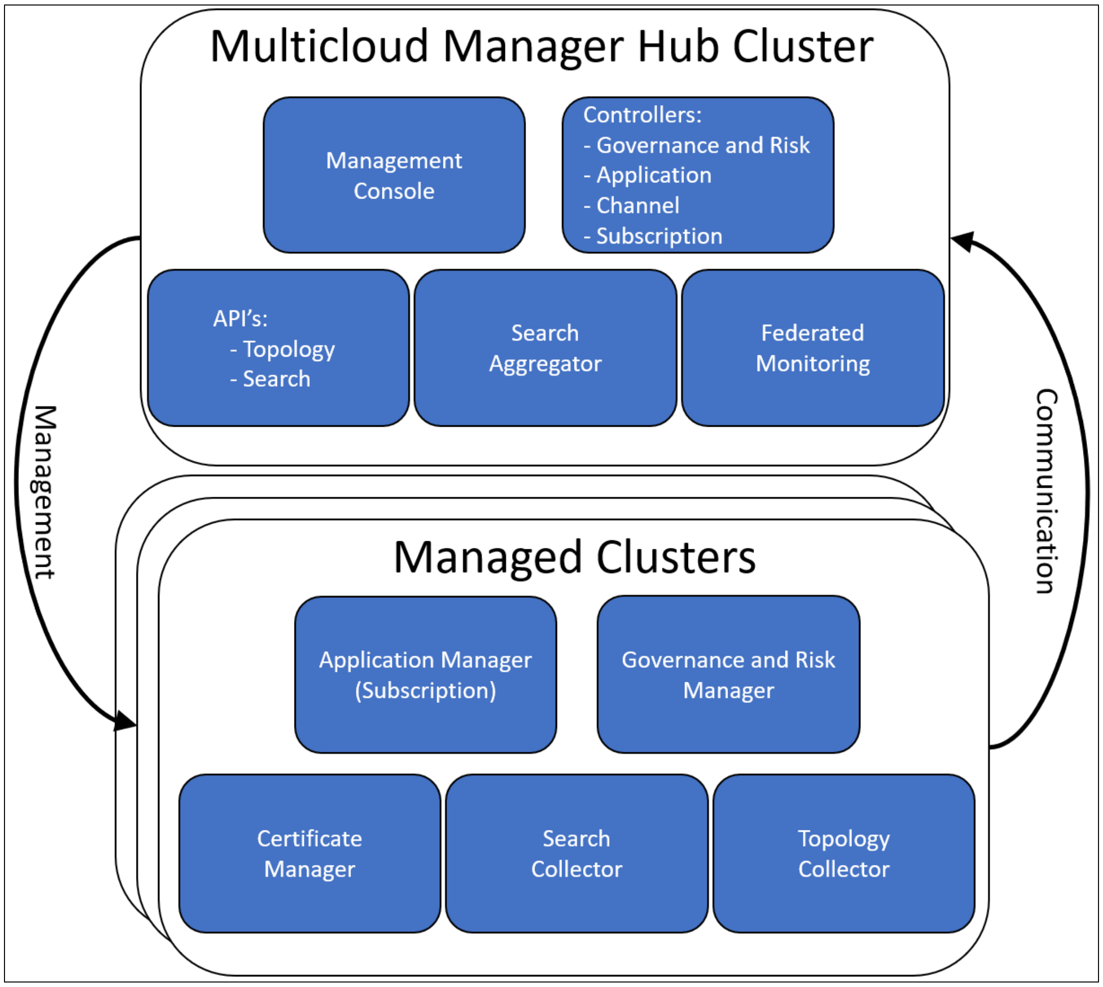


Figure 3-9 Multicloud Manager high-level architecture

Multicloud Manager hub cluster

Hub cluster is used to define Multicloud Manager controller. Hub cluster aggregates information from multiple clusters by using asynchronous work request. With a graph database, the hub cluster maintains the state of clusters and applications that run on it.

Hub cluster also uses `etcd`, which is a distributed key value store to store the state of work requests and results from multiple clusters. It provides a set of REST APIs for various functions that it supports.

Multicloud Manager managed cluster

Managed cluster defines Multicloud manager Klusterlet. Klusterlet is an agent that is responsible for a single Kubernetes cluster. The managed cluster starts a connection to the hub cluster, receives and applies work requests, and returns results. The managed cluster connects to different services in the cluster for operations, such as Kubernetes API service.

IBM Multicloud Manager Application resources

After you configure an IBM Multicloud Manager hub cluster and a managed cluster, you can view and deploy applications with application resources. Your application is used to view only your resource, although other application resource examples are for deployment. A multi-cluster application uses a Kubernetes specification, but with more automation of the deployment and lifecycle management of resources to individual clusters.

IBM Multicloud Manager Governance and risk

After you configure an IBM Multicloud Manager hub cluster and a managed cluster, you can define IBM Multicloud Manager security risk and create policies with templates from the Governance and risk page. For more information see, IBM Knowledge Center.

For more information about IBM Multicloud Manager, see IBM Knowledge Center.

3.7 IBM Cloud Pak for Security

Every enterprise in its journey to the cloud creates challenges for the security team. What becomes clear is that cybersecurity does not need news tools; instead, it needs new rules. Security teams are overwhelmed with hundreds of thousands of events that originate from various sources. What they see today is that the security mechanisms are fragmented and spread across several tools.

3.7.1 Features

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.

Key capabilities and benefits

IBM Cloud Pak for Security features the following key capabilities and benefits:

- Run anywhere - Connect security openly

  IBM Cloud Pak for Security is built on an open cloud-native framework that is integrated with Red Hat OpenShift. As such, the solution can be deployed and run on any hybrid multicloud environment, on-premises, private cloud, or public cloud.

  The unified interface that is provided by IBM Cloud Pak for Security assists the security teams to integrate data with analytic tools, and to combine that data across their cloud environments to spot advanced threats.

- Gain security insights without moving data

  IBM Cloud Pak for Security enables enterprises to connect and coordinate with their security tools and data sources. Transferring data to analyze creates more complexity. Therefore, you can combine all data sources to uncover hidden threats and make better risk-based decisions while leaving the data where it is stored.

  By connecting and coordinating various security tools, IBM Cloud Pak for Security assists the security teams in integrating data with analytic tools, which allows them to orchestrate and automate their security response so they can better prioritize their team's time. These tools often are connected with other IBM and third-party tools, such as Security Information and Event Management (SIEM), endpoint detection systems, threat smart services, Identity Manager, and software repositories.

- Respond faster to security incidents with automation

  IBM Cloud Pak for Security provides a unified interface to orchestrate responses and automates actions to security incidents. The use of automation playbooks assists the security teams to respond faster to security incidents.

  By formalizing security processes and activities across the enterprise, companies can react faster and more efficiently, while supplying themselves with the information that is needed for potential future regulatory scrutiny.

3.7.2 Layers

Figure 3-10 shows the application layer at the top layer of the architecture. Those applications rely on the Data Security Integration Services layer, which gives the capability to connect with the security data sources. The environment is based on the Application Framework layer that allows the integration of several components, including composable (BYOA) applications to create applications and integrate them in the framework.

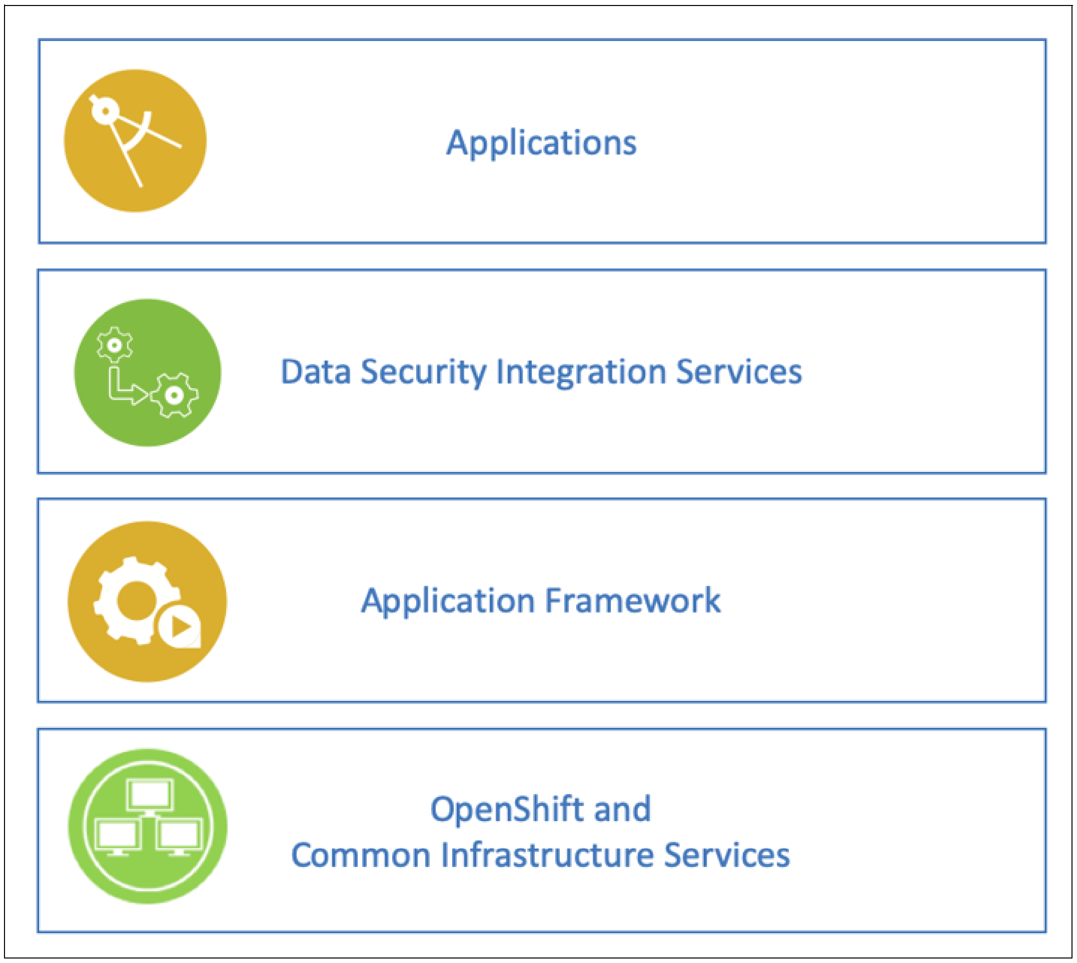

Applications
Data Security Integration Services
Application Framework
OpenShift and
Common Infrastructure Services

Figure 3-10 IBM Cloud Pak for Security Architecture

Figure 3-11 shows an overview of the major blocks of the IBM Cloud Pak for Security architecture.

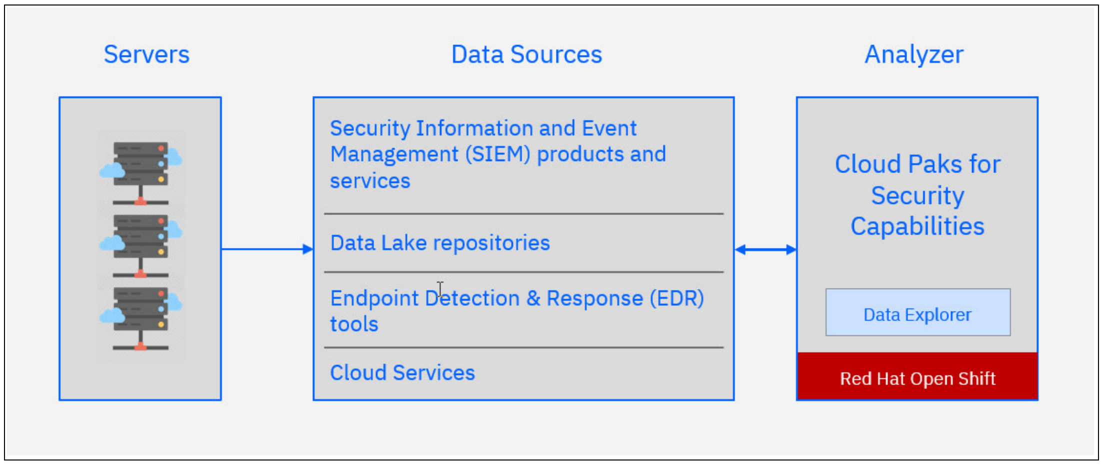

Servers
Data Sources
Analyzer
Security Information and Event Management (SIEM) products and services
Data Lake repositories
Endpoint Detection & Response (EDR) tools
Cloud Services
Cloud Paks for Security Capabilities
Data Explorer
Red Hat Open Shift

Figure 3-11 Major components of the IBM Cloud Pak for Security

IBM Cloud Pak for Security data sources consist of several components. The threats investigating capabilities are bound to the IBM Cloud Pak for Security through special components called *connectors*. The data explores orchestration and automation capability can be alerted by various connectors, as listed in Table 3-1.

Table 3-1 Examples of data sources and connectors

Data source	Connectors
SIEM	IBM QRadar®, Splunk
Data Lake	Elasticsearch
EDR	Carbon Black Response
IBM Guardium®	Guardium Data Protection
Big Fix	Big Fix Compliance
AWS	Cloud Watch, Guard Duty
Azure	Azure Monitor
IBM Cloud	IBM Cloud Advisor

Although IBM Cloud Pak for Security includes a library of connectors, connectors can be created by using an Open Toolkit.

Part 2

Red Hat OpenShift

This part introduces Red Hat OpenShift, provides an overview of the modular components and services, planning (for installation) considerations, preparing an environment, and deploying an OCP instance.

The following chapters are included in this part:

4

Red Hat OpenShift components and architecture

OpenShift is a set of modular components and services that are based on Kubernetes cluster and Docker, which runs on top of Red Hat Linux. OpenShift provides access to container images with a focus on easy composition of applications by developers.

This chapter describes the OpenShift cluster platform components and architecture and includes the following topics:

4.1 OpenShift cluster components

Figure 4-1 shows an overview of the OpenShift container platform components.

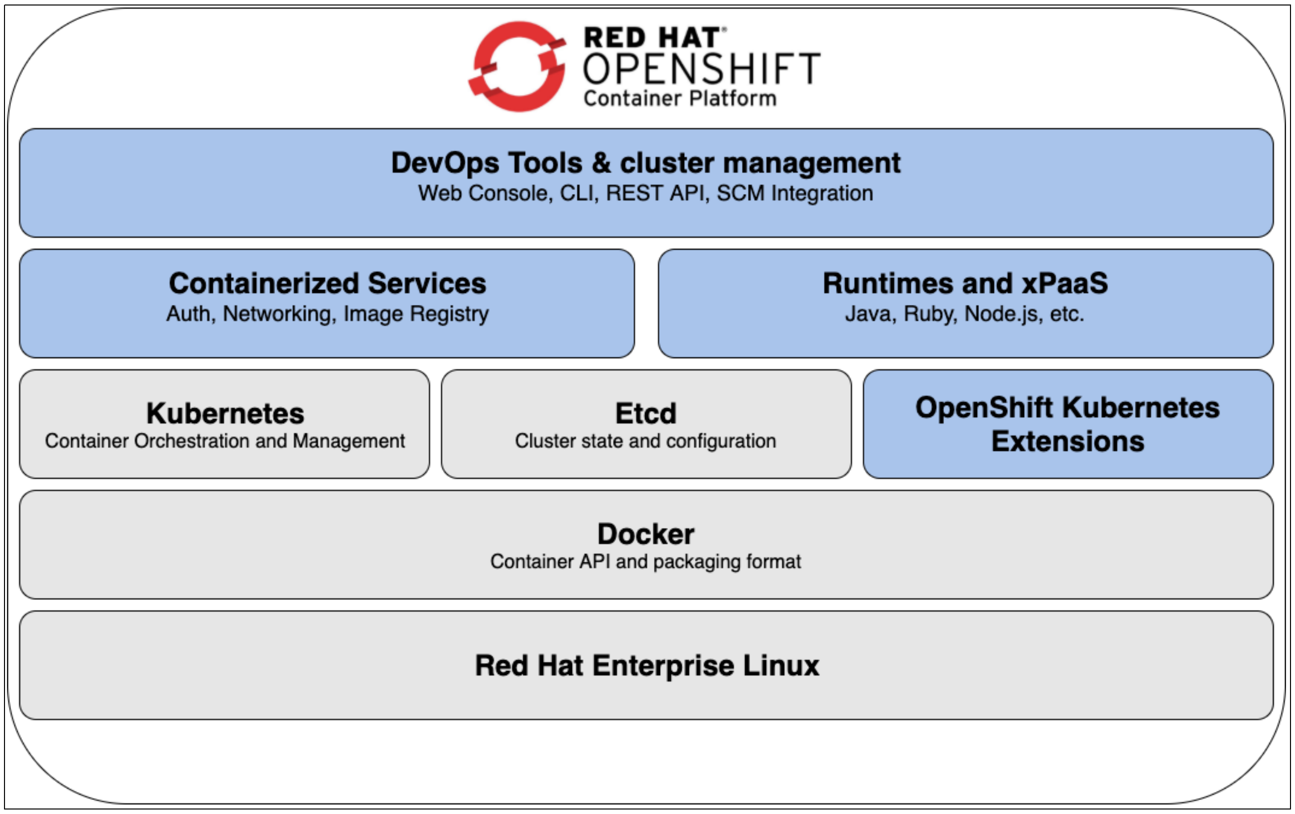

RED HAT
OPENSHIFT
Container Platform
DevOps Tools & cluster management
Web Console, CLI, REST API, SCM Integration
Containerized Services
Auth, Networking, Image Registry
Runtimes and xPaaS
Java, Ruby, Node.js, etc.
Kubernetes
Container Orchestration and Management
Etcd
Cluster state and configuration
OpenShift Kubernetes Extensions
Docker
Container API and packaging format
Red Hat Enterprise Linux

Figure 4-1 Red Hat OpenShift cluster platform components

4.1.1 Docker service and Kubernetes

The Docker service that runs in every OpenShift cluster node provides the container image administration. The Kubernetes cluster provides cluster management and orchestrates containers on multiple nodes in the OpenShift cluster.

4.1.2 etcd store

The `etcd` store is a distributed key value store that is used by Kubernetes to store configuration and state information about the containers and resources that are inside the OpenShift cluster.

4.1.3 OpenShift-Kubernetes extensions

OpenShift-Kubernetes extensions are more resources that are used to save the OpenShift configuration and cluster internal state. These resource extensions are stored in `etcd` with application resources that are managed by Kubernetes.

4.1.4 Containerized services

Containerized services are infrastructure service functions, such as networking components and authorization. Most OpenShift internal services run as Docker containers that are managed by Kubernetes.

4.1.5 Run times and xPaaS

Run times and xPaas are base container images that are ready for use by developers, each preconfigured with a particular runtime language or database. They can be used as-is or extended to add frameworks, libraries, and even other middleware products. The xPaaS offering is a set of base images for JBoss middleware products, such as JBoss EAP and ActiveMQ.

4.1.6 DevOps tools and user experience

OpenShift provides Web UI and CLI management tools for developers and system administrators, which enables the configuration and monitoring of applications and OpenShift services and resources. Web UI and CLI tools are built from the same REST APIs, which can be used by external tools, such as IDEs and CI platforms. OpenShift can also reach external SCM repositories and container image registries and bring their artifacts into the OpenShift cloud.

Note: OpenShift does not hide the core Docker and Kubernetes infrastructure from developers and system administrators. Instead, it uses them for its internal services and allows importing raw containers and Kubernetes resources into the OpenShift cluster so that they can benefit from added capabilities. Also, Raw containers and resources can be exported from the OpenShift cluster and imported into other Docker-based infrastructures.

In addition to Docker and Kubernetes, OpenShift container platform adds source code management for developers, application management, user and group management, and networking infrastructure to give user access to the applications that are running in the cluster.

4.1.7 Master and nodes

An OpenShift cluster is a set of node servers that run containers and are centrally managed by a set of master servers. A master server can act as a node server, but for large OpenShift deployments, the recommendation is that those roles are segregated for increased stability.

For more information about the recommended architecture for IBM Power Systems, see Chapter 5, “Red Hat OpenShift installation planning and considerations” on page 71.

Figure 4-2 shows a graphical view of OpenShift architecture components and how it works.

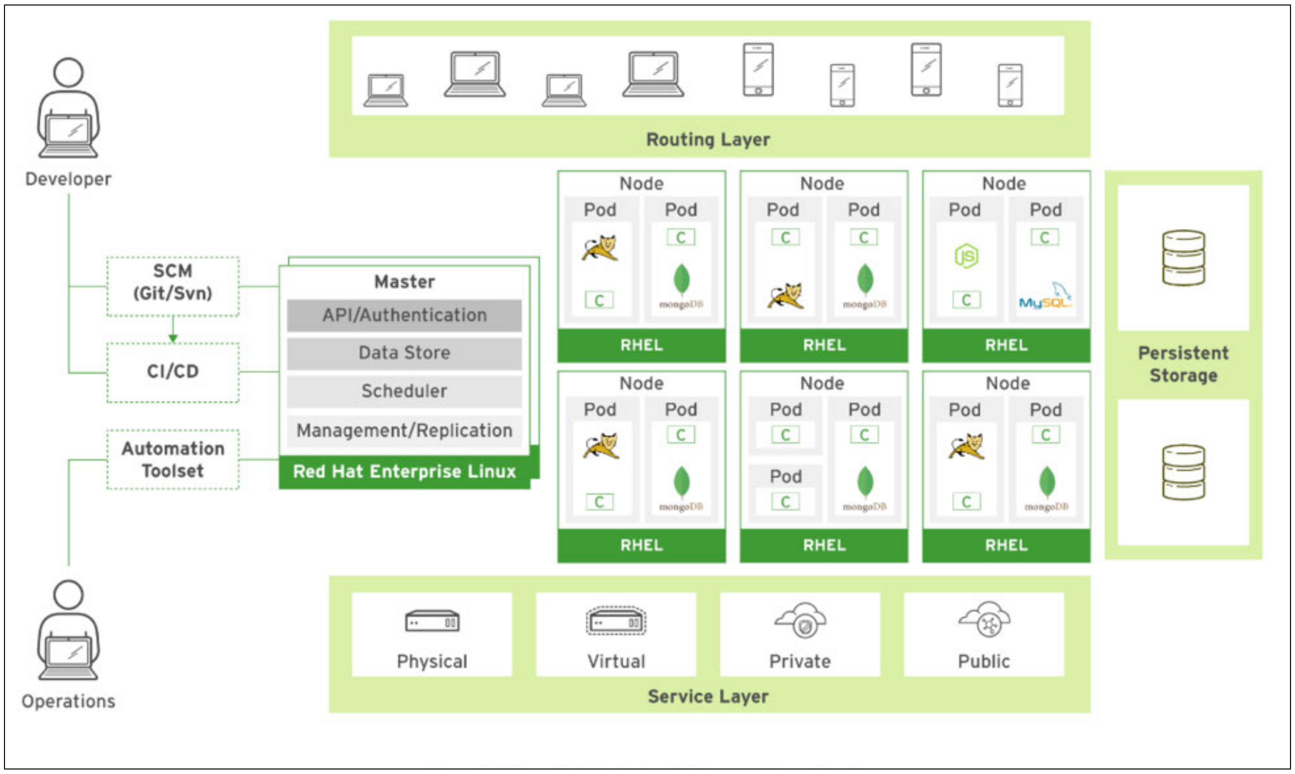


Figure 4-2   OpenShift Container Platform architectural components

The master nodes runs OpenShift core services, such as authentication, and provides the API entry point for administration. The nodes run applications inside containers, which are grouped into pods and then, the Kubernetes cluster divides the load between the cluster nodes.

OpenShift masters run the Kubernetes master services and *etcd* daemons, and at the same time, the nodes run the Kubernetes kubelet and kube-proxy daemons.

The masters are also nodes themselves. Scheduler, Management/Replication, and `etcd` are examples of Kubernetes master services.

A *pod* is one or more containers that share a virtual network device, internal IP address, and storage. The Kubernetes skeduling process ensures that pods are assigned to nodes and that Kubelet service can run them. Skeduling is based on pod policies configuration. Scheduling decisions include individual and collective resource requirements, hardware and software/policy constraints, affinity and anti-affinity specifications, and others.

Kubernetes manages replicas to scale pods. A replica is a set of pods that share a definition. For example, a replica that consists of many Apache and PHP pods that are running the same container image can be used for horizontally scaling a web application.

4.1.8 OpenShift projects and applications

In 2.4, “Kubernetes: An open source container orchestration” on page 24, the Kubernetes resources where described. However, OpenShift also manages projects and users.

A *project* is a group of Kubernetes resources with common user access rights. A project can also be assigned a quota, which limits the number of defined pods, volumes, services, and other cluster resources.

OpenShift client provides a **`new-app`** command that creates resources inside a project; then, OpenShift uses the app label to group these related resources into a new application that is inside that specific project.

4.2 OpenShift container platform networking

While installing all the prerequisites that are needed to deploy the OpenShift Container Platform configuration, all of the nodes must include internet access to register each node and install every dependency that is left.

During the installation process, an internal network is configured. To make this process possible, a virtual switch is installed and configured automatically. This switch ensures that the internal network is forwarded from the public network.

OpenShift Container Platform features a built-in DNS to resolve the hosts that are created internally. This DNS manages the port forwarding for the public IP into the internal network, such as for resolving the internal services IP addresses.

When a service is created, a new IP address can be assigned to it. If this service is deleted and re-created, a new IP address can be assigned by default or the cloud administrator can configure a specific IP address and host name on it.

This section provides an overview of the main OpenShift software-defined networking (SDN) solutions and describes the traffic flow among pods that are inside the cluster. Also discussed is how these pods communicate to destinations that are outside of the cluster.

4.2.1 OpenShift networking overview

OpenShift Networking features two main components: the OpenShift Software Defined Network plug-in to handle the communication within the cluster and the OpenShift Router plug-in to handle the inbound and outbound traffic that is destined to services in the cluster.

The default OpenShift SDN solution is built on Open vSwitch (OVS). With OpenShift, the cluster administrator can choose to deploy with one of the OpenShift native SDN plug-ins or deploy the cluster by using a third-party SDN from the supported system.

If a different SDN is wanted, OpenShift supports Kubernetes CNI-compliant SDN solutions.

4.2.2 OpenShift internal cluster communication

OpenShift container Platform uses a software-defined networking approach to provide a unified cluster network that assigns an internal IP address to each pod in the cluster to ensure that all containers within the pod behave as though they were on the same host. In terms of port allocation, networking, naming, load balancing, and application configuration are the same as though they were physical hosts or virtual machines.

This pod network is established and maintained by the OpenShift SDN, which configures an overlay network by using Open vSwitch (OVS).

Note: For more information about Open vSwitch, see this website.

OpenShift provides the following SDN plug-ins for configuring the network:

- The ovs-subnet plug-in is the original plug-in, which provides a flat pod network in which every pod can communicate with every other pod and service. This configuration is the default configuration for single-tenant clusters.
- The ovs-multitenant plug-in provides project-level isolation for pods and services. Each project receives a unique Virtual Network ID (VNID) that identifies traffic from pods that are assigned to the project.

  Pods from different projects cannot send packets to or receive packets from pods and services of a different project. However, projects that receive VNID 0 are more privileged, so they can communicate with all other pods, and all other pods can communicate with them.

  In OpenShift Container Platform clusters, the default project has VNID 0. This designation facilitates certain services, such as the load balancer, to communicate with all other pods in the cluster and vice versa.
- The ovs-networkpolicy plug-in allows project administrators to configure their own isolation policies by using NetworkPolicy objects.

OpenShift SDN maintains a registry of all nodes in the cluster. This registry is stored in `etcd`. When the system administrator registers a node, OpenShift SDN allocates an unused `/23` subnet from the cluster network and stores this subnet in the registry. When a node is removed or deleted from the cluster, the OpenShift SDN frees the corresponding cluster network subnet. This subnet becomes available for future allocations to new nodes.

This subnet value for nodes is specified in the Ansible inventory host variable `osm_cluster_network_cidr`. The default network if this variable is not set is `10.128.0.0./14` (for example, `10.128.0.0 - 10.131.255.255`).

The nodes are allocated `/23` subnets (for example, `10.128.0.0/23`, `10.128.2.0/23`, `10.128.4.0/23`, and others). Therefore, the cluster network has 512 subnets that are available to assign to nodes, and a specific node is allocated 510 addresses that it can assign to the containers that are running on it.

When the ovs-multitenant plug-in is used, the OpenShift SDN master also watches for the creation and deletion of projects and assigns VXLAN VNIDs to them. These VXLAN VNIDs are used later by the nodes to isolate traffic correctly.

To identify the cluster network subnet that is allocated to each node, run the **`oc get hostsubnet`** command with a user with cluster-admin privilege, as shown in Figure 4-3.

```
# oc get hostsubnet
NAME                             HOST                             HOST IP          SUBNET         EGRESS CIDRS    EGRESS
IPS
rbapp01.domain.example.com    rbapp01.domain.example.com    192.168.11.217    10.1.10.0/23    []              []
rbapp02.domain.example.com    rbapp02.domain.example.com    192.168.11.218    10.1.6.0/23     []              []
rbapp03.domain.example.com    rbapp03.domain.example.com    192.168.11.219    10.1.8.0/23     []              []
rbinf01.domain.example.com    rbinf01.domain.example.com    192.168.11.214    10.1.14.0/23    []              []
rbinf02.domain.example.com    rbinf02.domain.example.com    192.168.11.215    10.1.12.0/23    []              []
rbmst01.domain.example.com    rbmst01.domain.example.com    192.168.11.211    10.1.4.0/23     []              []
rbmst02.domain.example.com    rbmst02.domain.example.com    192.168.11.212    10.1.0.0/23     []              []
rbmst03.domain.example.com    rbmst03.domain.example.com    192.168.11.213    10.1.2.0/23     []              []
```

Figure 4-3 Output from oc get hostsubnet command

When a node is added to the cluster, the OpenShift SDN registers the local host with the registry on master to allocate an open subnet to the node.

Then, OpenShift SDN creates and configures the following network devices:

- br0: Open Virtual Switch (OVS) bridge device to which pod containers are attached. OpenShift SDN also configures a set of non-subnet-specific flow rules on this bridge.
- tun0: OVS internal port (port 2 on br0). This port is assigned the cluster subnet gateway address and is used for external network access. OpenShift SDN configures netfilter and routing rules to enable access from the cluster subnet to the external network by way of NAT.
- vxlan_sys_4789: OVS VXLAN device (port 1 on br0), which provides access to containers on remote nodes. It is referred to as *vxlan0* in the OVS rules.

When a pod is started on the host, OpenShift SDN completes the following process:

1. Assigns the pod an open IP address from the node's cluster subnet.
2. Attaches the host side of the pod's veth interface pair to the OVS bridge br0.
3. Adds OpenFlow rules to the OVS database to route traffic that is addressed to the new pod to the correct OVS port. In the case, of the ovs-multitenant plug-in, it adds OpenFlow rules to tag traffic that is coming from the Pod with the Pod's VNID, and to allow traffic into the Pod if the traffic's VNID matches the Pod's VNID (or is the privileged VNID 0). Non-matching traffic is filtered out by a generic rule.

OpenShift SDN nodes also watch for subnet updates from the SDN master. When a subnet is added, the node adds OpenFlow rules on br0 so that packets with a destination IP address in the remote subnet go to vxlan0 (port 1 on br0) and thus out onto the network. The ovs-subnet plug-in sends all packets across the VXLAN with VNID 0, but the ovs-multitenant plug-in uses the suitable VNID for the source container.

Figure 4-4 shows the components that are involved in pods communication.

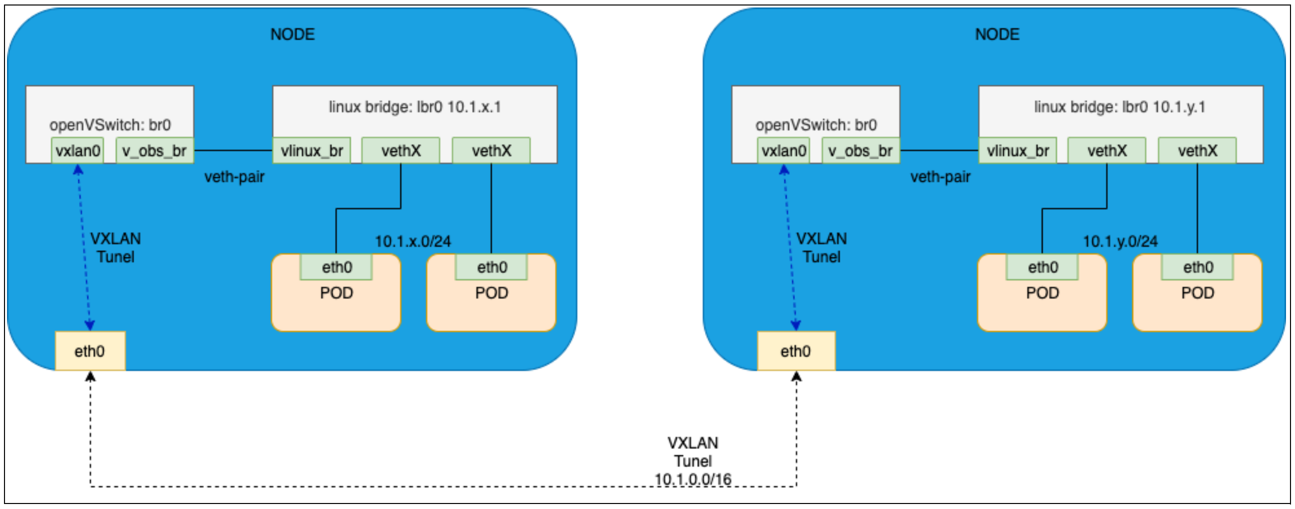


Figure 4-4 Communication example between pods

If both containers are running in the same node, the communication flow from one pod to another uses the vethx interface from the same br0 interface of the node.

If the containers are running on different nodes, the flow of packets from one pod to another use the vethX interface from the br0 ovs interface on different nodes.

Finally, if the Pod connects to an external host, the traffic flow from the eth0 interface in the pod to the vethX in the Linux bridge then to the br0 interface in the OVS uses the tun0 interface through the eth0 to the physical network.

Almost all packet delivery decisions are performed with OpenFlow rules in the OVS bridge br0, which simplifies the plug-in network architecture and provides flexible routing. In the case of the ovs-multitenant plug-in, this configuration also provides enforceable network isolation.

4.2.3 OpenShift external cluster communication

OpenShift Container Platform provides different ways to access the applications or services that are running inside the cluster.

Administrators can make available a service endpoint that external traffic can reach by assigning a unique external IP address to that service from a range of external IP addresses. This IP address range is specified by using a CIDR notation, which allows an application user to make a request against the cluster for an external IP address.

Each IP address must be assigned to only one service to ensure that each service has a unique endpoint.

The recommendation, in order or preference, is:

- Use a router if you use HTTP/HTTPS or TLS-encrypted protocol other than HTTPS.
- Use a Load Balancer, an External IP, or a NodePort.

OpenShift Container Platform Router plug-in

The following OpenShift Router plug-ins are available:

- HAProxy Template Router: The HAProxy template uses the openshift3/ose-haproxy-router image to deploy one or more Router Pods (container) that are running on Infrastructure Nodes on the OpenShift Container Platform.
- F5 BIG-IP Router plug-in: The F5 router integrates with a F5 BIG-IP system in your environment to synchronize routes. F5 BIG-IP version 11.4 or newer is required to have the F5 iControl REST API.

For more information about the use of the router HAProxy plug-in, see this web page.

Assigning a public IP by using a load balancer service

This method allows traffic to nonstandard ports through an IP address that is assigned from a pool.

If you do not need a specific external IP address, you can configure a load balancer service to allow external access to an OpenShift Container Platform cluster.

A load balancer service allocates a unique IP from a configured pool. The load balancer features a single edge router IP (which can be a virtual IP (VIP), but is still a single machine for initial load balancing).

Manually assigning an external IP to a service

A Kubernetes service serves as an internal load balancer. It identifies a set of replicated pods to proxy the connections it receives to them. Backing pods can be added to or removed from a service arbitrarily, although the service remains consistently available, which enables anything that depends on the service to refer to it at a consistent address.

The default service clusterIP addresses are from the OpenShift Container Platform internal network and they are used to permit Pods to access each other.

To permit external access to the service, more externalIP and ingressIP addresses that are external to the cluster can be assigned to the service. These externalIP addresses also can be virtual IP addresses that provide highly available access to the service.

Services are assigned an IP address and port pair that, when accessed, proxy the connections to an appropriate backing pod. A service uses a label selector to find all the containers that are running that provide a specific network service on a specific port, as shown in Example 4-1.

Example 4-1 OpenShift service description

```
# oc   describe service httpd-example02
Name:              httpd-example02
Namespace:         project-test-ff
Labels:            app=httpd-example
template=httpd-example
Annotations:       description=Exposes and load balances the application pods
Selector:          name=httpd-example02
Type:              ClusterIP
IP:                172.30.92.1
Port:              web  8080/TCP
TargetPort:        8080/TCP
Endpoints:         10.1.6.4:8080
Session Affinity:  None
Events:            <none>
```

Configuring a NodePort

NodePort is used to make available the service on a static port on all nodes in the cluster, as shown in Example 4-2.

Example 4-2 Service YAML sample file for nodePort configuration

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    description: Exposes and load balances the application pods
  creationTimestamp: '2019-10-09T16:04:11Z'
  labels:
    app: httpd-example
    template: httpd-example
  name: httpd-test03
  namespace: project-test
  resourceVersion: '72908925'
  selfLink: /api/v1/namespaces/project-test/services/httpd-test03
  uid: 6ec5640d-eaae-11e9-860f-fafa40b70720
spec:
  clusterIP: 172.30.166.205
  externalTrafficPolicy: Cluster
  ports:
    - name: http
      nodePort: 30080
      port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    name: httpd-test03
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

NodePorts are in the 30000-32767 range by default, which is unlikely to match a service's intended port.

For service and application access, the administrator must ensure that the external IPs are routed to the nodes and local firewall rules on all nodes allow access to the open port. The DNS wildcard feature can be used to configure resolution for a subset of names to an IP address in the cluster.

The service is accessed by using the `<NodeIP>:30080` address, as shown in Example 4-2.

4.3 OpenShift persistent storage

Because Plain Docker storage is not shared between nodes, it is not enough for container service availability because pods might be stopped on one node and restarted on another node at any time. If a database pod is stopped and restarted on another node, any stored data is lost.

Kubernetes provides a framework for managing external persistent storage for containers. Kubernetes recognizes a PersistentVolume resource, which can be defined as local or network storage. A pod resource can reference a PersistentVolumeClaim resource to access storage of a certain size from a PersistentVolume.

Kubernetes also specifies whether a PersistentVolume resource can be shared between pods or if each pod needs its own PersistentVolume with exclusive access. When a pod moves to another node, it stays connected to the same PersistentVolumeClaim and PersistentVolume instances. Therefore, a pod's persistent storage data follows it, regardless of the node where it is scheduled to run.

OpenShift adds several VolumeProviders to Kubernetes, which provide access to enterprise storage, such as iSCSI, Fibre Channel, Gluster, or a cloud block volume service, such as OpenStack Cinder.

OpenShift also provides dynamic provisioning of storage for applications by way of the StorageClass resource. By using dynamic storage, you can select different types of back-end storage. The back-end storage is segregated into different tiers, depending on the needs of your application.

For example, a cluster administrator can define a StorageClass with the name of "fast," which uses higher-quality back-end storage, and another StorageClass called "slow," which provides commodity-grade storage.

When requesting storage, a user can specify a PersistentVolumeClaim with an annotation that specifies the value of the StorageClass that they prefer.

4.4 OpenShift registry

OpenShift Container Platform can use any server that implements the container image registry API as a source of images, including the Docker Hub, private registries that are run by third parties, and the integrated OpenShift Container Platform registry.

4.4.1 Integrated OpenShift Container Registry

OpenShift Container Platform provides an integrated container image registry called OpenShift Container Registry (OCR). This registry that adds the ability to automatically provision new image repositories on demand. This feature provides users with a built-in location for their application builds to push the resulting images.

Whenever a new image is pushed to OCR, the registry notifies OpenShift Container Platform about the new image, passing along all the information about it, such as the namespace, name, and image metadata. Different components of OpenShift Container Platform react to new images, creating builds and deployments.

OCR can also be deployed as a stand-alone component that acts solely as a container image registry, without the build and deployment integration.

4.4.2 Third-party registries

OpenShift Container Platform can create containers by using images from third-party registries. However, these registries do *not* offer the same image notification support as the integrated OpenShift Container Platform registry. In this situation, OpenShift Container Platform fetches tags from the remote registry upon imagestream creation. Refreshing the fetched tags is as simple as running the **`oc import-image <stream>`** command. When new images are detected, the build that was described in 4.4.1, "Integrated OpenShift Container Registry" and deployment reactions occur.

4.5 Managing OpenShift resources

All OpenShift resources, images, containers, pods, services, builders, templates, and so on, are stored on `etcd` and can be managed by the OpenShift CLI, web console, or REST API. These resources also are defined in text files in JSON or YAML format and can be changed by editing those files and shared on an SCM system, such as GIT.

OpenShift can even retrieve these resource definitions directly from an external SCM.

5

Red Hat OpenShift installation planning and considerations

This chapter describes the Red Hat OpenShift planning, considerations, and installation guidelines and includes the following topics:

5.1 IBM Power Systems

Over the years, the IBM Power Systems family grew, matured, was innovated, and pushed the boundaries of what clients expect and demand from the harmony of hardware and software.

With the advent of the IBM POWER4 processor in 2001, IBM introduced logical partitions (LPARs) outside of their mainframe family to another audience. What was seen as radical then, grew into the expected today. The term *virtualization* is now common-place across most platforms and operating systems. These days, virtualization is the core foundation for cloud computing.

IBM Power Systems is built for the most demanding, data-intensive, computing on Earth. The servers are cloud-ready and help you unleash insight from your data pipeline: from managing mission-critical data, to managing your operational data stores and data lakes, to delivering the best server for cognitive computing.

IBM POWER9, the foundation for the number 1 and number 2 supercomputers in the world, is the only processor with state-of-the-art I/O subsystem technology, including next generation NVIDIA NVLink, PCIe Gen4, and IBM OpenCAPI™.

IBM POWER9 processor-based servers can be found in three product families: Enterprise servers, Scale-out servers, and Accelerated servers. Each of these three families is positioned for different types of client requirements and expectations.

IBM Power Systems servers that are based on IBM POWER9 processors are built for today's most advanced applications from mission-critical enterprise workloads to big data and AI, as shown in Figure 5-1.

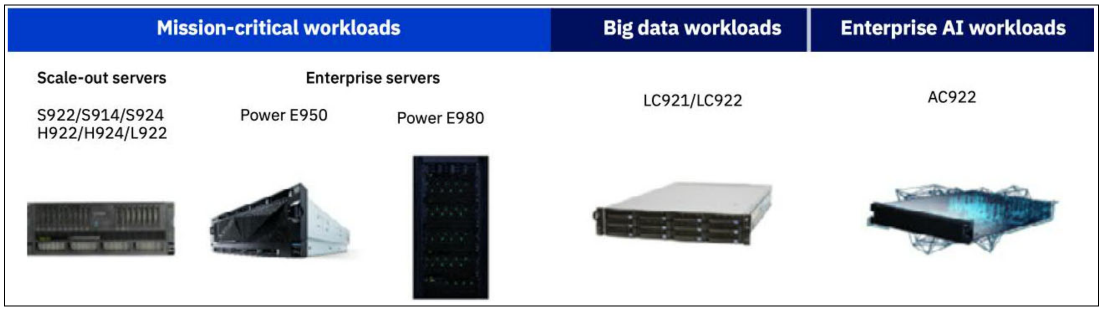


Figure 5-1 IBM Power Systems

- Mission critical

  Robust scale-out and enterprise servers can support a wide range of mission-critical applications that are running on IBM AIX, IBM i, and Linux operating systems. They also can provide building blocks for private and hybrid cloud environments.

- Big data

  Scale-out servers deliver the performance and capacity that is required for big data and analytics workloads.

- Enterprise AI

  Servers provide intelligent, accelerated infrastructure for modern analytics, HPC, and AI workloads. They are advanced enterprise servers that deliver fast machine learning performance.

5.1.1 Mission-critical workloads

To handle the most data-intensive mission-critical workloads, organizations need servers that can deliver outstanding performance and scalability. Whether they are supporting small business groups or building large private and hybrid cloud environments, they need no-compromise infrastructure.

Enterprise (Scale-up) servers

IBM Power Systems Enterprise (Scale-up) servers, as listed in Table 5-1, offer the highest levels of performance and scale for the most data- intensive, mission-critical workloads. They can also serve as building blocks for growing private and hybrid cloud environments. Support for AIX, Linux, and IBM i (for the IBM Power Systems E980 server) gives organizations the flexibility to run a wide range of applications.

The IBM Power Systems E950 server is the correct fit for growing midsize businesses, departments, and large enterprises that are looking for a building-block platform for their data center. The IBM Power Systems E980 server is designed for large enterprises that need flexible, reliable servers for a private or hybrid cloud infrastructure.

Table 5-1 IBM Power Systems - Enterprise servers

	E950	**E980**
Key features	▶ Enterprise-class capabilities in a reliable, space-efficient form factor ▶ Exceptional performance at an affordable price	▶ Ideal foundation for world-class private or hybrid cloud ▶ Can power large-scale, mission-critical applications ▶ Flagship, high-end server
Machine type and model (MTM)	9040-MR9	9080-M9S
Form factors	4U	5U system node and 2U system controller unit
Sockets	2 - 4	4 per node
Processor cores	▶ Up to 48 cores – 12 core processor sockets at 3.15 - 3.80 GHz (max) ▶ Up to 44 cores – 11 core processor sockets at 3.2 - 3.80 GHz (max) ▶ Up to 40 cores – 10 core processor sockets at 3.40 - 3.80 GHz (max) ▶ Up to 32 cores – 8 core processor sockets at 3.60 - 3.80 GHz (max)	One node: 4x POWER9 CPUs; 8, 10, 11 or 12 cores each System Maximum: 16x POWER9 CPUs; 8, 10, 11, or 12 cores each
Memory slots	128	128 per node
Memory max.	16 TB	64 TB per node
PCIe G4 slots	10	8 per node; max. 32
Supported operating systems	AIX and Linux	AIX, IBM i, and Linux

Scale-out servers

IBM Power Systems scale-out servers for mission-critical workloads, as listed in Table 5-2, offer a strong alternative to commodity x86 servers. They provide a robust, reliable platform to help maximize performance, and help ensure availability.

Scale-out AIX, IBM i, and Linux servers are designed to scale out and integrate into an organization's cloud and AI strategy. They deliver exceptional performance and reliability.

Table 5-2 IBM Power Systems: Scale-out servers

	S914	**S922**	**S924**	**L922**
Key features	► Entry-level offering ► Industry-leading integrated security and reliability ► Cloud-enabled	► Strong price performane for mission-critical workloads ► Dense form factor with large memory footprint ► Cloud-enabled with integrated virtualization	► Industry-leading price performane for mission-critical workloads ► Large memory footprint ► Strong security and reliability ► Cloud-enabled with integrated virtualization	► Industry-leading price performane for mission-critical Linux workloads ► Dense form factor with large memory footprint
Machine type and model (MTM)	9009-41A	9009-22A	9009-42A	9008-22L
Form factors	4U and tower	2U	4U	2U
Sockets	1	2	2	1 or 2
Microprocessors	1x POWER9 CPUs; 4, 6 or 8 cores	Up to 2x POWER9 CPUs; 4, 8 or 10 cores	2x POWER9 CPUs; 8, 10 or 12 cores	Up to 2x POWER9 CPUs; 8, 10 or 12 cores
Memory slots	16	32	32	32
Memory maximum	1 TB	4 TB	4 TB	4 TB
PCIe G4 slots	2	4	4	4
Supported operating systems	AIX, IBM i, and Linux	AIX, IBM i, and Linux	AIX, IBM i, and Linux	Linux

Scale-out servers for SAP HANA servers, as listed in Table 5-3 on page 75, deliver outstanding performance and a large memory footprint of up to 4 TB in a dense form factor. These servers help deliver insights fast although maintaining high reliability. They are also scalable: When it is time to grow, organizations can expand database capacity and the size of their SAP HANA environment without having to provision a new server.

Table 5-3 IBM Power Systems: Scale-out servers for SAP HANA

	H922	H924
Key features	► Optimized for SAP HANA ► High performance, tight security ► Dense form factor with large memory footprint ► For Linux-focused customers	► High performance for SAP HANA ► Strong security with large memory footprint ► For Linux-focused customers
Machine type and model (MTM)	9223-22H	9223-42H
Form factors	2U	4U
Sockets	1 upgradeable or 2	2
Cores per socket	4, 8, or 10	8, 10, or 12
Memory slots	32	32
Memory maximum	4 TB	4 TB
PCIe G4 slots	4	4
Supported operating systems	AIX, IBM i, and Linux	AIX, IBM i, and Linux

5.1.2 Big data workloads

Across industries, organizations are poised to capitalize on big data to generate new business insights, improve the customer experience, enhance efficiencies and gain competitive advantage. But to make the most of growing data volumes, they need servers with the performance and capacity for big data and AI workloads.

IBM Power Systems Scale-out servers for big data, as shown in Table 5-4, deliver the outstanding performance and scalable capacity for intensive big data and AI workloads. Purpose-built with a storage-rich server design and industry-leading compute capabilities, these servers explore and analyze a tremendous amount of data, all at a lower cost than equivalent x86 alternatives.

Table 5-4 IBM Power Systems: Scale-out servers for big data

	LC921	LC922
Key features	► High performance in a space-saving design ► Industry-leading compute in a dense form factor	► Highest storage capacity in the IBM Power Systems portfolio ► Up to 44 cores and 2 TB of memory ► High performance at lower cost than comparable x86 systems
Machine type and model (MTM)	9006-12P	9006-22P
Form factors	1U	2U
Sockets	1 upgradeable or 2	2
Microprocessors	1x or 2x POWER9 CPUs; 16 or 20 cores	1x or 2x POWER9 CPUs; 16, 20 or 22 cores

	LC921	LC922
Memory slots	32	16
Memory maximum	2 TB	2 TB
PCIe G4 slots	4	6
Supported operating system	Linux	Linux
Maximum storage	40 TB	120 TB

5.1.3 Enterprise AI workloads

AI holds tremendous promise for facilitating digital transformations, accelerating innovation, enhancing the efficiency of internal processes, identifying new marketplace opportunities, and more. For organizations to take advantage of AI and cognitive technologies, such as machine learning and deep learning, they need powerful, accelerated servers that can handle these data-intensive workloads.

Accelerated servers can also play a vital role in supercomputing. With the correct accelerated servers, researchers and scientists can explore more complex, data- intensive problems and deliver results faster than before.

The IBM Power Systems Accelerated Compute Server, as listed in Table 5-5, helps reduce the time to value for enterprise AI initiatives. The IBM PowerAI Enterprise platform combines this server with popular open source deep learning frameworks and efficient AI development tools to accelerate the processes of building, training, and inferring deep learning neural networks. By using PowerAI Enterprise, organizations can deploy a fully optimized and supported AI platform with blazing performance, proven dependability, and resilience.

Table 5-5 IBM Power Systems - Accelerated Compute servers

	AC922
Key features	▸ Unprecedented performance for modern AI, analytics, and HPC workloads ▸ Proven deployments from small clusters to the world’s largest supercomputers, with near-linear scaling ▸ Simple GPU acceleration
Machine type and model (MTM)	8335-GTH I 8335-GTX
Form factors	2U
Sockets	2
Microprocessors	2x POWER9 with NVLink CPUs: 16 or 20 cores; or 18 or 22 cores with liquid cooling
GPUs	4 or 6 NVIDIA Tesla GPU processors (NVLink 2.0 attached)
Memory slots	16
PCIe G4 slot	1 TB
Supported operating system	Linux

5.2 Red Hat OpenShift Container Platform 3.11 on IBM Power Systems

Red Hat OpenShift Container Platform 3.11 for Power Systems provides a secure, enterprise-grade platform for IBM Power Systems servers. It brings together industry-leading container orchestration from Kubernetes, advanced application build and delivery automation, and Red Hat Enterprise Linux certified containers for IBM Power Systems.

Red Hat OpenShift Container Platform 3.11 for Power Systems brings developers and IT operations together with a common platform. It provides applications, platforms, and services for creating and delivering cloud-native apps and management so IT can ensure that the environment is secure and available. It also enables in-place (for example, on IBM POWER) application modernization of existing enterprise applications by surrounding them with new container technology.

Red Hat OpenShift Container Platform 3.11 for Power Systems provides enterprises the same functionality as the Red Hat OpenShift Container Platform offering on other platforms. Key features include:

- Self-service environment for application and development teams.
- Pluggable architecture that supports a choice of container runtimes, networking, storage, Continuous Integration/Continuous Deployment (CI-CD), and more.
- Ability to automate routine tasks for application teams.

Figure 5-2 shows a high-level view of the Red Hat OpenShift Container Platform components for the various IBM Power Systems hardware platforms.

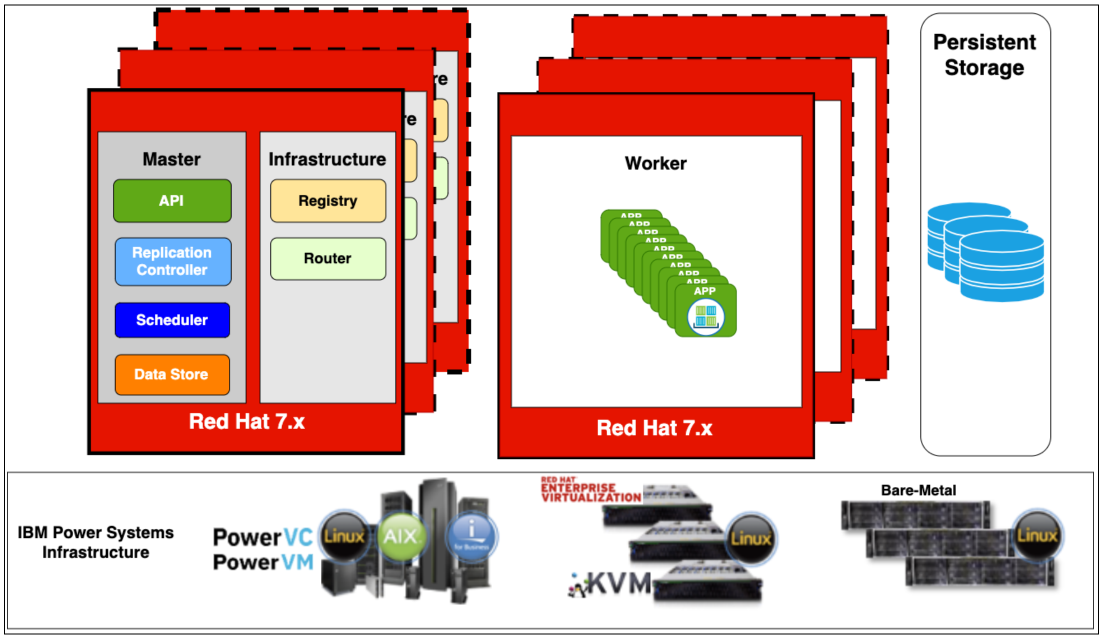


Figure 5-2 High-level view of the OpenShift Container Platform for IBM Power Systems

From bare metal physical machines to virtualized infrastructure, or in private clouds, the OpenShift is supported anywhere that Red Hat Enterprise Linux is running, including all of the supported virtualization platforms (PowerVM or RHEV) and private cloud (PowerVC).

The OpenShift architecture builds on top of Kubernetes and consists of three types of roles for the nodes:

- Master: These nodes are Kubernetes Master Nodes that can provide more functions, such as the web console with the self-service portal, and the developers and operations-focused dashboards.

  Important: Because of the consensus that is required by the RAFT algorithm, the `etcd` service must be deployed in odd numbers to maintain quorum. For this reason, the minimum number of `etcd` instances for production environments is three.

- Infrastructure: These nodes are Kubernetes Worker Nodes that are dedicated to host functions, such as the OpenShift Routes and the OpenShift internal registry.
- Worker: These nodes are the Kubernetes Worker Nodes that are used to run the microservices and containerized applications that are deployed on OpenShift.

Master and Infrastructure roles can run on the same node.

Nodes can run on top of PowerVC, PowerVM, Red Hat Virtualization, KVM, or run bare metal environment. Table 5-6 lists the IBM Power Systems Infrastructure Landscape for OpenShift Container Platform 3.11.

Table 5-6 IBM Power Systems Infrastructure Landscape for OpenShift 3.11

IaaS	PowerVC	N/A	RHV-M	N/A
Hypervisor	PowerVM	PowerVM	KVM/RHV	Bare-Metal
Guest operating system	Red Hat 7.6 or later	Red Hat 7.6 or later	Red Hat 7.6 or later	Red Hat 7.6 or later
Systems	E980, E950, S924, S922, S914, L922	E980, E950, S924, S922, S914, L922	LC922, AC922	LC922, AC922
Storage	SAN, Software Defined Storage	NAS, Software Defined Storage	NAS, Software Defined Storage	NAS, Software Defined Storage
Storage attach	FVD: PowerVC, NFS, GlusterFS, FVD: Spectrum Scale	NFS, GlusterFS, FVD: Spectrum Scale	NFS, GlusterFS, FVD: Spectrum Scale	NFS, GlusterFS, FVD: Spectrum Scale

Note: In this IBM Redbooks publication, we cover only the deployment of the OpenShift Container Platform 3.11 on top of PowerVC 1.4.3.1 environment.

5.3 Red Hat OpenShift Container Platform 3.11 on IBM PowerVC

In this section, we provide guidelines and considerations for deploying and managing Red Hat OpenShift Container Platform on IBM Power Virtualization Center (PowerVC).

Red Hat OpenShift Container Platform is a Platform-as-a-Service (PaaS) that provides developers and IT organizations with a cloud application platform for deploying new applications on secure, scalable resources with minimal configuration and management overhead. It allows developers to create and deploy applications by delivering a consistent environment for both development and during the runtime lifecycle that requires no server management.

IBM PowerVC uses OpenStack technology to provide enterprise virtualization and cloud management for IBM Power Systems, which provides all of the necessary capabilities for a fully featured Infrastructure-as-a-Service (IaaS) private cloud solution.

Note: The minimum version for PowerVC is 1.4.3 Fix Pack 1. This version adds IBM PowerVC FlexVolume Driver support for OpenShift Container Platform 3.11.

5.3.1 Reference architecture summary

The deployment of Red Hat OpenShift Container Platform on PowerVC varies among several factors that affect the installation process. Consider the following questions:

- How many instances do you minimally require in the cluster?
- How to configure the PowerVC Host Groups and colocation rules?
- Is High Availability required?
- What installation tools will you use?
- Which storage and network backends will you use?

For this reference architecture, eight instances (VMs) were defined: six for the OpenShift cluster, one for the Load Balancer, and one for the Deployment Host. We tested on Single or Dual Host Groups (Availability Zones) and soft-anti-affinity rules for OpenShift cluster instances (VMs).

This reference architecture requires High Availability configuration (as described in “High Availability” on page 84) and uses Ansible and IBM Terraform for installation and prerequisite checks.

Note: Terraform is not required to install OpenShift. Multiple approaches are available for deploying infrastructure. In this book, we demonstrate how to use an infrastructure as code with Terraform to simplify the deployment. The Terraform examples in this book are open source and “use-at-your-own-risk” templates.

This reference architecture uses OpenShift ovs-multitenant on top of Shared Ethernet Adapter (SEA) for network and IBM PowerVC FlexVolume on top of Cinder for persistent storage.

Figure 5-3 shows the environment that was used for this reference architecture.

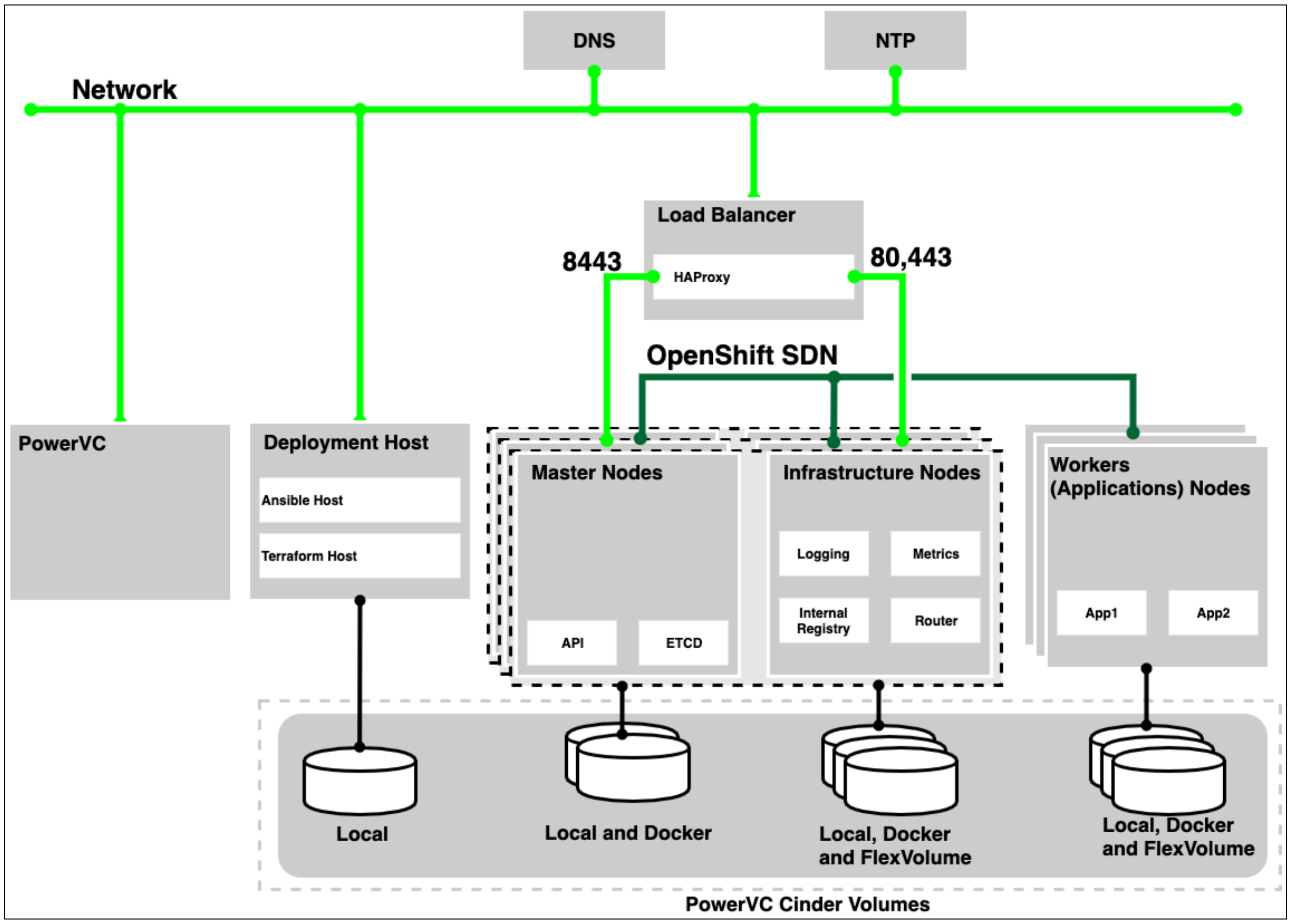


Figure 5-3 Reference architecture summary

The Red Hat OpenShift Container Platform consists of the following instances:

- IBM PowerVC instance
- Deployment Host instance
- Load Balancer instance
- Three Master-Infrastructure instances
- Three Worker (Applications) instances

Note: Instructions for installing and configuring PowerVC is out of the scope of this book. For more information about PowerVC installation and configuration instructions, see IBM Knowledge Center.

5.3.2 Design considerations

This section discusses some design considerations.

PowerVC considerations

IBM PowerVC is an advanced virtualization and cloud management offering for IBM Power Systems. Built on OpenStack, it provides comprehensive virtualization management and cloud deployments for IBM AIX, IBM i, and Linux virtual machines (VMs). PowerVC simplifies the lifecycle management of the virtualization for Power Systems. It includes a deep integration with IBM PowerVM virtualization technologies.

Availability zones (host groups)

Host groups, also known as *host aggregates* in OpenStack's terminology, allow you to create virtual boundaries around a group of hosts. It is a logical group of hosts, regardless of any features that they might or might not have in common. For example, the hosts feature the same architecture, network configuration, or storage, or hosts in the same rack or data center.

When a host group is created by using the user interface, an availability zone with the same name is created and assigned to the host group. PowerVC also supports the standard OpenStack APIs for host groups and availability zones.

Host groups (availability zones) include the following features:

- Every host must be in a host group

  Any hosts that do not belong to a user-defined host group are members of the default host group. The default host group cannot be deleted.

- Virtual machines are kept within the host group

  A virtual machine can be deployed to a specific host or to a host group. After deployment, that virtual machine must always be migrated or remote restarted within the host group.

- Placement policies are associated with host groups

  Every host within a host group is subject to the host group's placement policy. The default placement policy is striping.

- Automated Remote Restart

  If enabled, the PowerVC monitors hosts for failure by using the Platform Resource Scheduler (PRS) HA service. If a host fails, PowerVC automatically remote restarts the VMs from the failed host to another host within a host group.

- Dynamic Resource Optimizer (DRO)

  If enabled, DRO continuously monitors your cloud environment's usage. You can specify that DRO monitors CPU usage or available memory. When a host is found to be overused, the DRO attempts to correct the situation by performing the action that you specified. It can migrate VMs to another host within a host group or, when applicable, work with Capacity on Demand (CoD) to activate mobile cores.

Note: A host can belong only to one host group (availability zone).

Depending on the infrastructure and servers types, you must determine the suitable choice when deciding between single and multiple Host Groups (Availability Zones) configuration. From an administrative perspective, it is easier to have single Host Group and not isolate the systems into multiple Host Group (Availability zones), as shown in Figure 5-4.

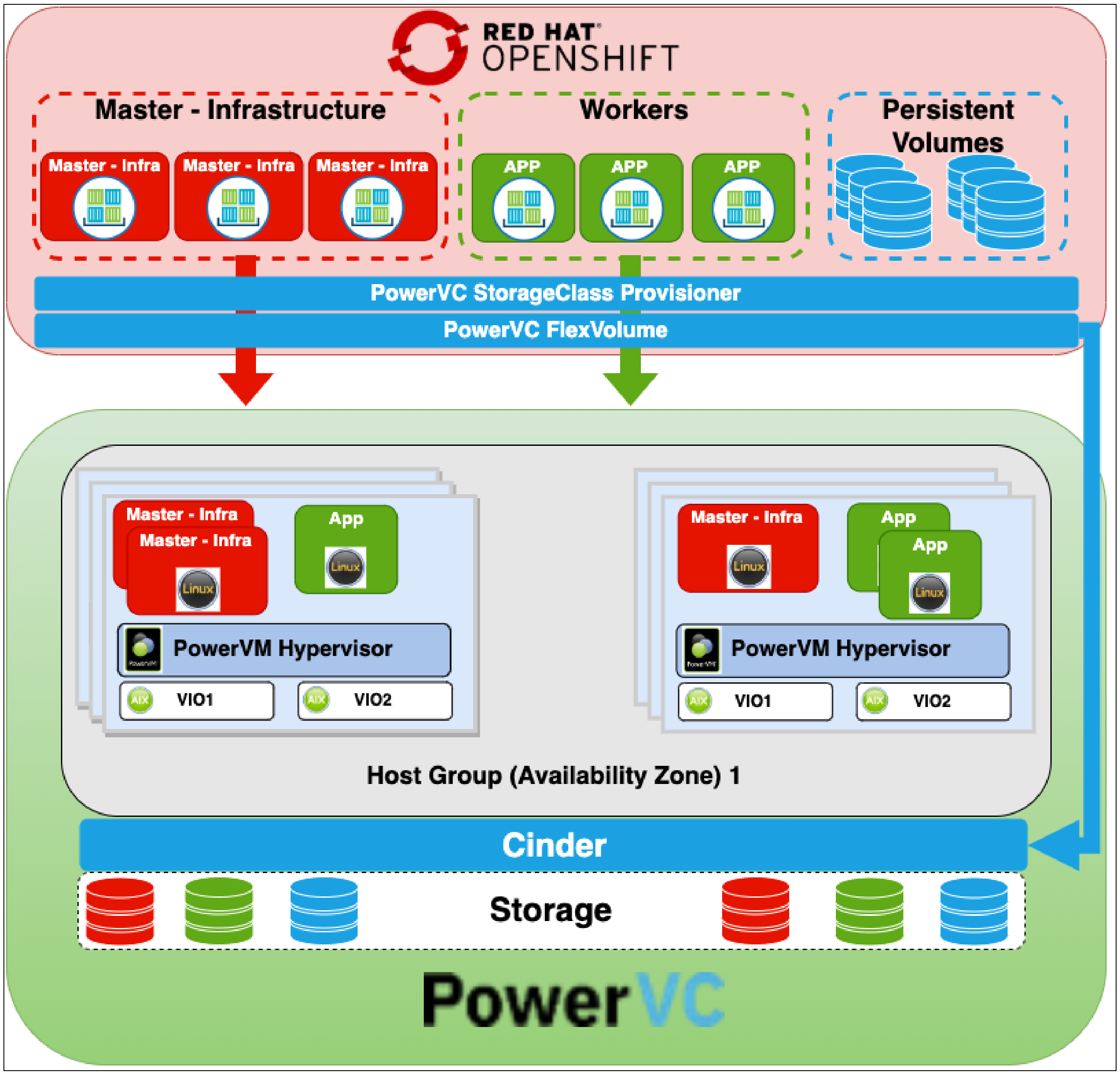


Figure 5-4 PowerVC single Host Group (Availability Zone)

To force the VM placement (in general based on different rack or data center), you can create dual Host Groups, as shown in Figure 5-5.

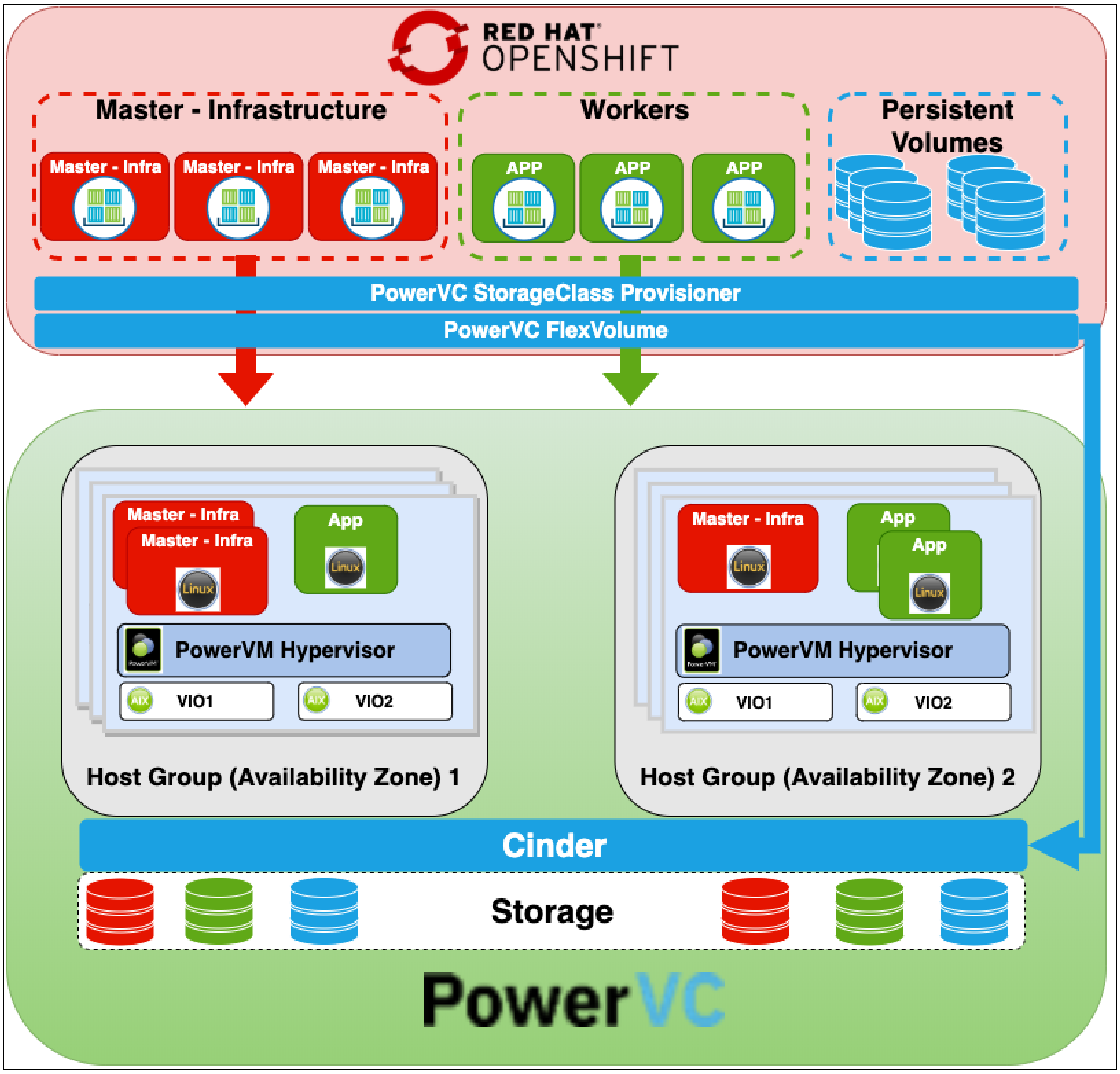


Figure 5-5 PowerVC dualHost Groups (Availability Zones)

Colocation rules

Colocation rules, also known as *server groups* in OpenStack's terminology, are used to specify that selected virtual machines must always be kept on the same host (affinity) or can never be placed on the same host (anti-affinity).

During deployment, migration, and remote restart, PowerVC ensures that these colocation rules are followed.

PowerVC supports the following policies:

- Affinity: All virtual machines from this rule are hosted on the same host.
- Anti-affinity: All virtual machines from this rule are hosted on different host.
- Soft-affinity: All virtual machines from this rule are hosted on the same host if possible; if not possible, they still are scheduled instead of failure and set the status of the rule to violated.
- Soft-anti-affinity: All virtual machines from this rule are hosted on different host, if possible; if not possible, they still are scheduled instead of failure and set the status of the rule to violated.

If the rule status is violated, review the placement of the virtual machines. For any virtual machines that are on an incorrect host, you can migrate it to the correct host by using LPM.

PowerVC image

A custom image must be created in PowerVC before the deployment is started. OpenShift includes the following specific requirements:

- RHEL 7.6 for POWER8 and RHEL-ALT 7.6 for POWER9 Minimum installation.
- SELinux is set to enforcing.
- NetworkManager is enabled and running.
- Firewall is enabled and running.
- The host name from DNS (set_hostname_from_dns) is set in cloud-init.

Note: For more information about image creation in PowerVC, see IBM Knowledge Center.

Project and user

A project, sometimes referred to as a *tenant*, is a unit of ownership. Most resources, such as virtual machines, volumes, and images, belong to a specific project. Only users with a role assignment for a specific project can work with the resources that belong to that project. The `ibm-default` project is created during installation, but PowerVC supports the creation of more projects for resource segregation.

When possible, it is recommended to create a user and project for the OpenShift environment.

High Availability

High Availability (HA) is a requirement for any production deployment. A crucial consideration for HA is the removal of single points of failure (SPOFs). This reference architecture is highly available at all layers (Hardware, PowerVM, PowerVC, and OpenShift).

Hardware HA

You must take care to eliminate single points of failure at the hardware layer. The following hardware fault tolerance recommendations are implemented in this architecture:

- Redundant server power supplies that are connected to different power sources.
- Redundant network adapters that are connected to redundant network switches.
- Redundant Fibre Channel adapters that are connected to redundant SAN fabrics.

PowerVM HA

The VIOS configuration includes the following specifications when LPAR is used with a production OpenShift Container Platform:

- If I/O virtualization is used, a dual-VIOS setup is mandatory. You can have more than two VIOSes in the system that separates different environments, such as production and test or multiple customers
- Each VIOS must be configured with at least two dedicated or dedicated donating cores for any production systems. Size them as needed and monitor CPU usage to adapt to workload changes over the life of the system.
- At least one Fibre Channel card per VIOS is needed. It is recommended to have two to remove the single points of failure (SPOFs).
- At least two Ethernet cards per VIOS are needed. Interfaces with 10 GbE are needed at a minimum for scale-up systems. For scale-out systems, a speed of at least 10 GbE is mandatory.
- Enable VIO servers for Live Partition Mobility (LPM).
- Configure your VIO servers with NPIV (if present) and are supported by the operating system, or with the Shared Ethernet Adapter (SEA) Failover with Load Sharing if they are not supported. The following options are available for setting up virtual networks on your VIO servers:
  - SR_IOV

    Not supported by PowerVC. Shares parts of a dedicated network adapter among several partitions. Works with all current IBM AIX, IBM i, and Linux distributions.

    Restrictions: Prevents Live Partition Mobility, restrictions on Etherchannel. Max 20 VM per network port.
  - vNIC

    SR_IOV enhanced with Live Partition Mobility support. No Etherchannel or bonding support. Maximum 20 VM per network port.
  - vNIC failover

    Provides server-side high availability solution (similar to SEA failover). In the vNIC failover configuration, a vNIC client adapter can be backed by multiple logical ports, preferably allocated from a different SR-IOV adapter and hosted by different VIOSes to avoid a single point failure.

    At any time, only one logical port is connected with the vNIC adapter. If the active (connected) backing device or its hosting VIOS fails, a new backing device is selected to serve the client. The selection of the active backing device is done by the POWER Hypervisor.

    In contrast to the SEA failover, the vNIC failover does not rely on any communication protocol between or among the multiple backing devices. The vNIC failover resorts to the POWER Hypervisor as the decision maker because it (the POWER Hypervisor) has a complete view and receives the real-time status of all the backing devices and is best situated for selecting the correct logical port. Without the implementation of the communication protocol, the vNIC failover is a much simpler, and more robust solution.

– SEA

Network high availability and redundancy with server-side failover solution that is based on the SEA failover to provide virtual network redundancy and failover capability. The main advantage of a server-side failover solution is that it simplifies the client configuration because you need to create only a single virtual Ethernet adapter in the client, which is served by two SEAs that are configured in the high availability mode (failover configuration) in two Virtual I/O Servers.

It is recommended to aggregate more physical connections that operate as one link. It increases network throughput and high availability. It features two basic modes of operation: Etherchannel and LACP 802.3ad, also known as *port trunking*.

Figure 5-6 shows an example with two VMs (LPARs) with dual Virtual I/O Server configuration with SEA on top of LACP (802.3ad) adapter for client LPARs and an Etherchannel adapter for VIO Management interface.

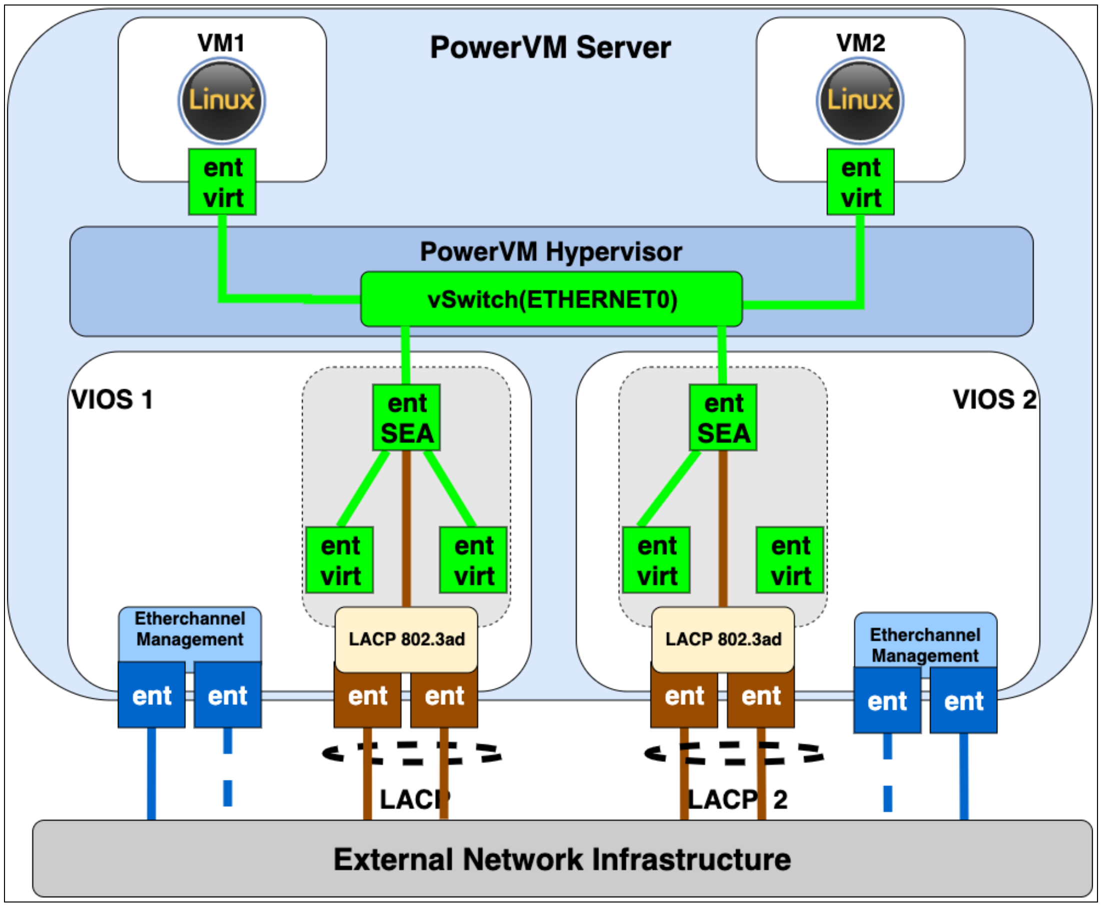


Figure 5-6 Shared Ethernet Adapter (SEA) with Load Sharing and LACP (802.3ad)

Note: At the time of this writing, vNIC on Red Hat was available as a Technology Preview. For more information, see this web page.

- The following options are available for setting up virtual storage on your VIO servers:
  - Shared Storage Pool (SSP)

    These pools allow a single physical volume on any supported storage controller to be shared across a cluster of Virtual I/O Servers. Those Virtual I/O Servers share accesses to aggregated physical volumes and divide that aggregated volume into storage volumes instead of interacting with external storage controllers or fabric switches.

    A cluster consists of up to 16 Virtual I/O Servers with a shared storage pool that provides distributed storage access to the Virtual I/O Servers in the cluster. Each cluster requires one physical volume for the repository physical volume and at least one physical volume for the storage pool physical volume. The shared storage pool can be accessed by all the Virtual I/O Servers in the cluster. All the Virtual I/O Servers within a cluster must have access to all the physical volumes in a shared storage pool.
  - Virtual SCSI (vSCSI) adapters

    These adapters provide one logical partition with the ability to use storage I/O (disk, CD, and tape) that is owned by the Virtual I/O Server partition. Virtual SCSI is based on a client and server relationship; the Virtual I/O Server owns the physical resources and has one Virtual SCSI server adapter that communicates with a logical partition Virtual SCSI client adapter.
  - N_Port ID Virtualization (NPIV)

    The NPIV allows a single physical HBA to register multiple virtual WWPNs. With NPIV, you can configure the managed system so that multiple partitions can access independent physical storage through the same physical Fibre Channel adapter.

    Each client partition is configured with four Virtual Fibre Channel adapters: two are mapped to Virtual Fibre Channel adapters on one VIOS and the other two are mapped to Virtual Fibre Channel adapters on the other VIOS. The client has WWPNs, four active and four inactive (used only during LPM operations).

    For client VMs to see same storage from all these ports, zoning must be done on the SAN. Multi-path software on the client VM takes care of routing IO through passive path if the active path fails. The NPIV is recommended for production environments.

Figure 5-7 shows an example with two NPIV VMs (LPARs) with dual Virtual I/O Server.

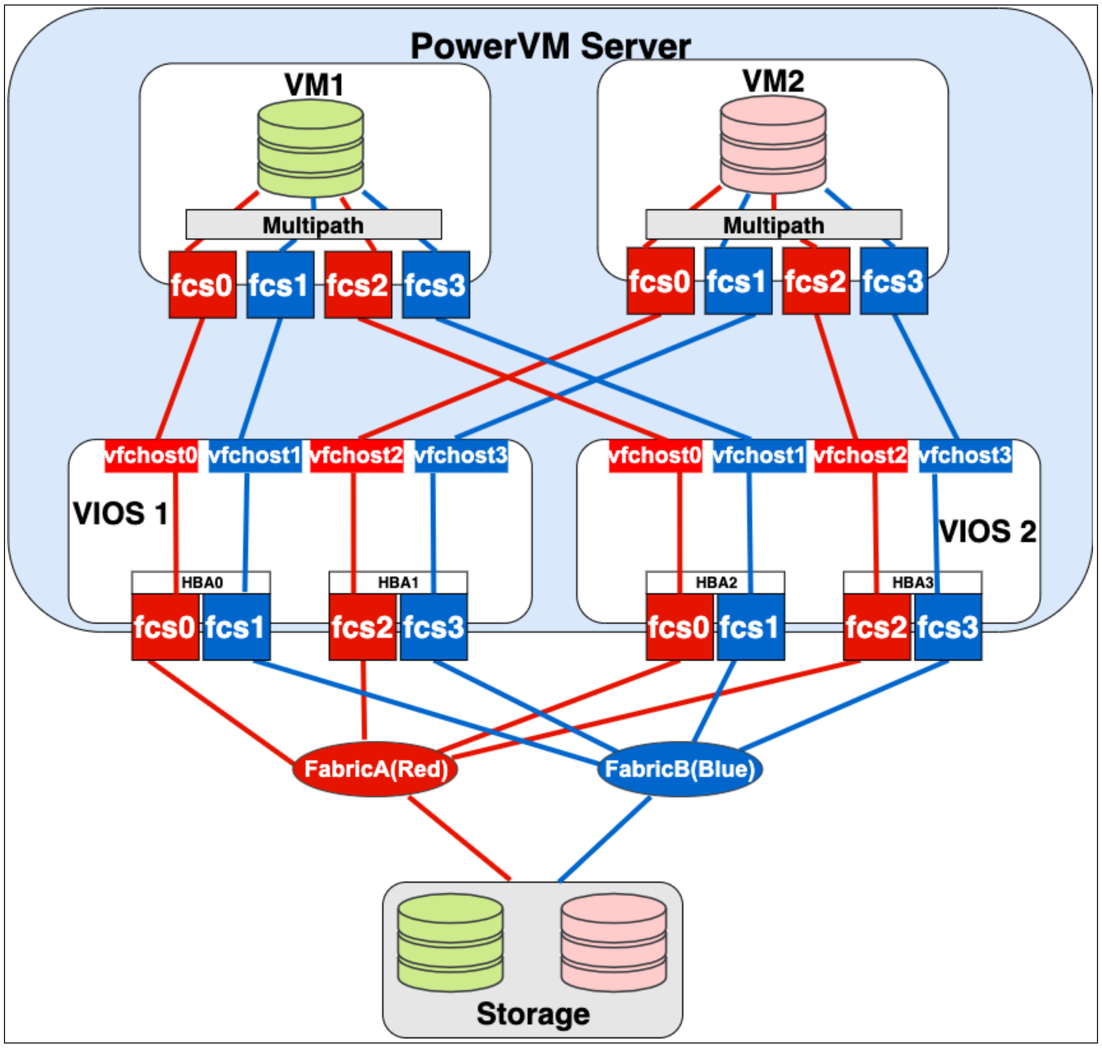


Figure 5-7 Two NPIV VMs with dual Virtual I/O servers

Note: For more information about PowerVM best practices, see *IBM PowerVM Best Practices*, SG24-8062.

PowerVC HA

This architecture includes the following PowerVC recommendations:

- Each Host Group (Availability Zone) must include more that one server to allow Simplify Remote Restart, DRO, and Host evacuation.
- Automated Remote Restart must be enabled at the Host Group level. When an error causes a server outage, a virtual machine that is configured with the remote restart capability can be restarted automatically on a different physical server.

  Figure 5-8 shows how Simplify Remote Restart is working with HMC configuration.

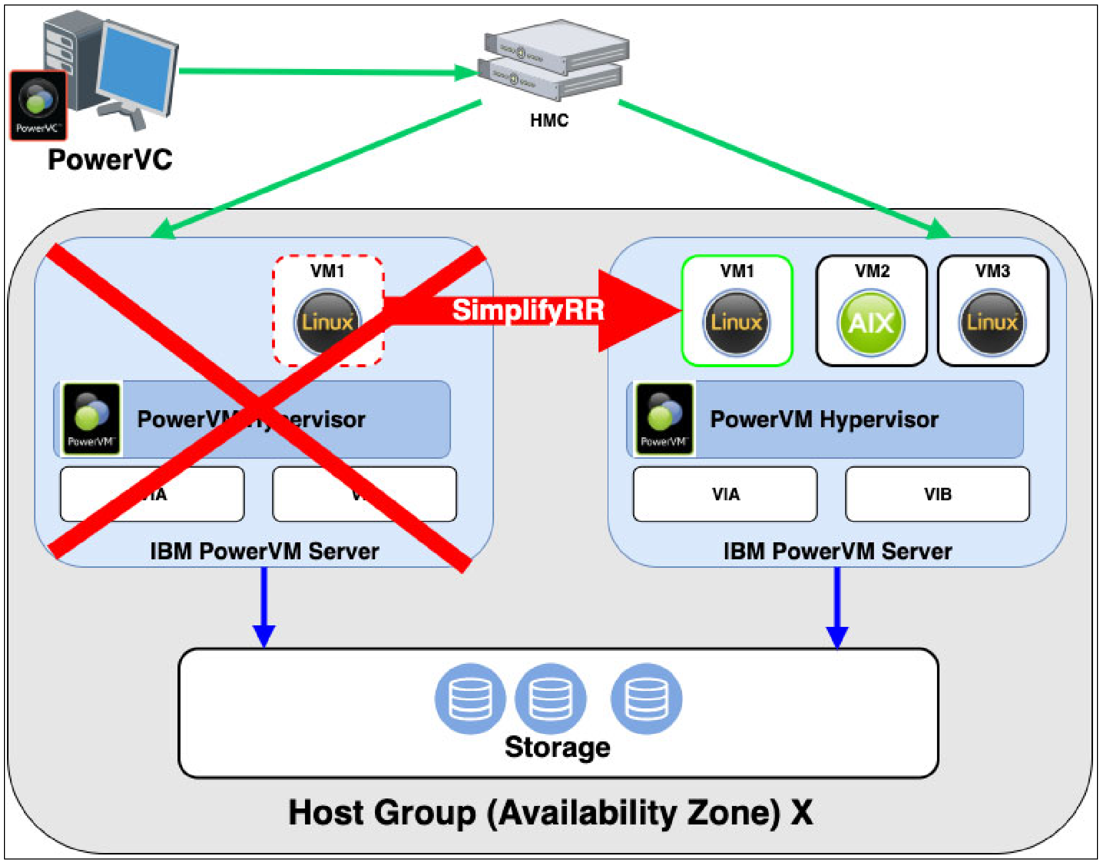


Figure 5-8 Simplify Remote Restart with PowerVC and HMC

- Use PowerVC to manage the LPM feature to migrate VMs from one host to another. The LPM is important for frame evacuation during maintenance operations. It is recommended to include a dedicated Network interface for VIO management.

  Figure 5-9 shows how LPM is working with HMC configuration.

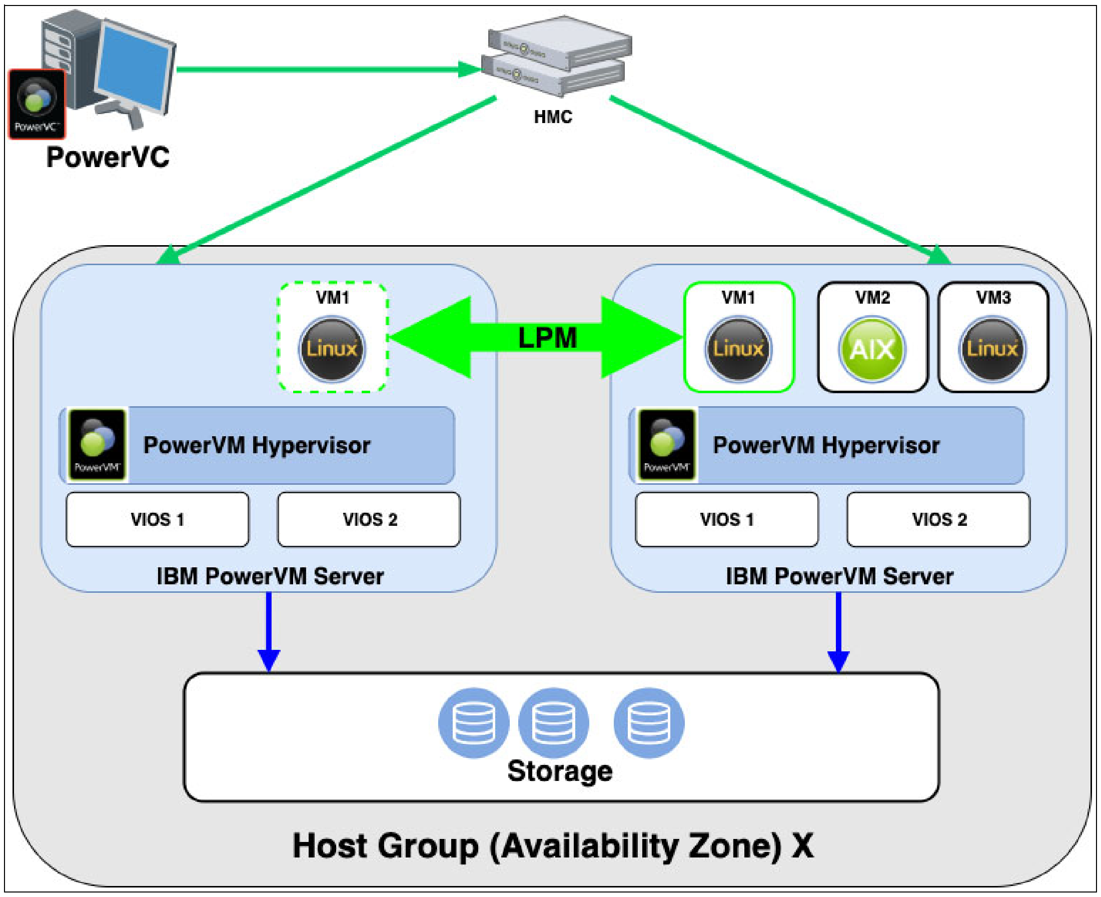


Figure 5-9 Live Partition Mobility (LPM) with PowerVC and HMC

- Master/Infrastructure nodes are configured with the automatic remote restart; in case of host failure, PowerVC automatically restarts the VMs (LPARs) on different hosts.
- Create Server Groups for OpenShift VMs and configure soft-anti-affinity rules.

OpenShift HA

OpenShift is also deployed for HA. In this reference architecture, the `etcd` state database is colocated across the master nodes. The `etcd` requires a minimum of three nodes for HA. All master nodes are configured in PowerVC to automatically restart in case of any host failure.

This reference architecture also uses three infrastructure nodes. Infrastructure nodes host OpenShift infrastructure components, such as the registry, containers for log aggregation, and metrics. A minimum of three infrastructure nodes are needed for HA when a shared aggregated logging database is used, and to ensure that service interruptions do not occur during a restart. All infrastructure nodes are configured in PowerVC to automatically restart in case of any host failure.

In deployments with four or more PowerVM Compute nodes, OpenShift must be configured to create application nodes across to all physical servers. The application nodes are not configured to automatically restart in case of host failure. OpenShift is managing the applications availability.

Installation tools

Perform the installation by using the tools that are recommended and supported.

Deployment host

A deployment host is any virtual or physical host that supports Terraform and Ansible.

For this book, the deployment host is a virtual machine that provides a simple method for deploying OpenShift and includes the following features:

- Minimal operating system that is based on Red Hat ALT 7.6 Enterprise Linux.
- Terraform for deploying OpenShift Infrastructure.
- OpenShift-Ansible for installing and configure OpenShift.

Note: The deployment host can run on any architecture (for example, x86) that supports the installation tools.

Terraform

Terraform is an Infrastructure as Code tool for building, changing, and versioning infrastructure safely and efficiently.

Ansible, Chef, Puppet, and SaltStack focus on automating the installation and configuration of the software. Terraform automates provisioning of the infrastructure.

Terraform components are shown in Figure 5-10.

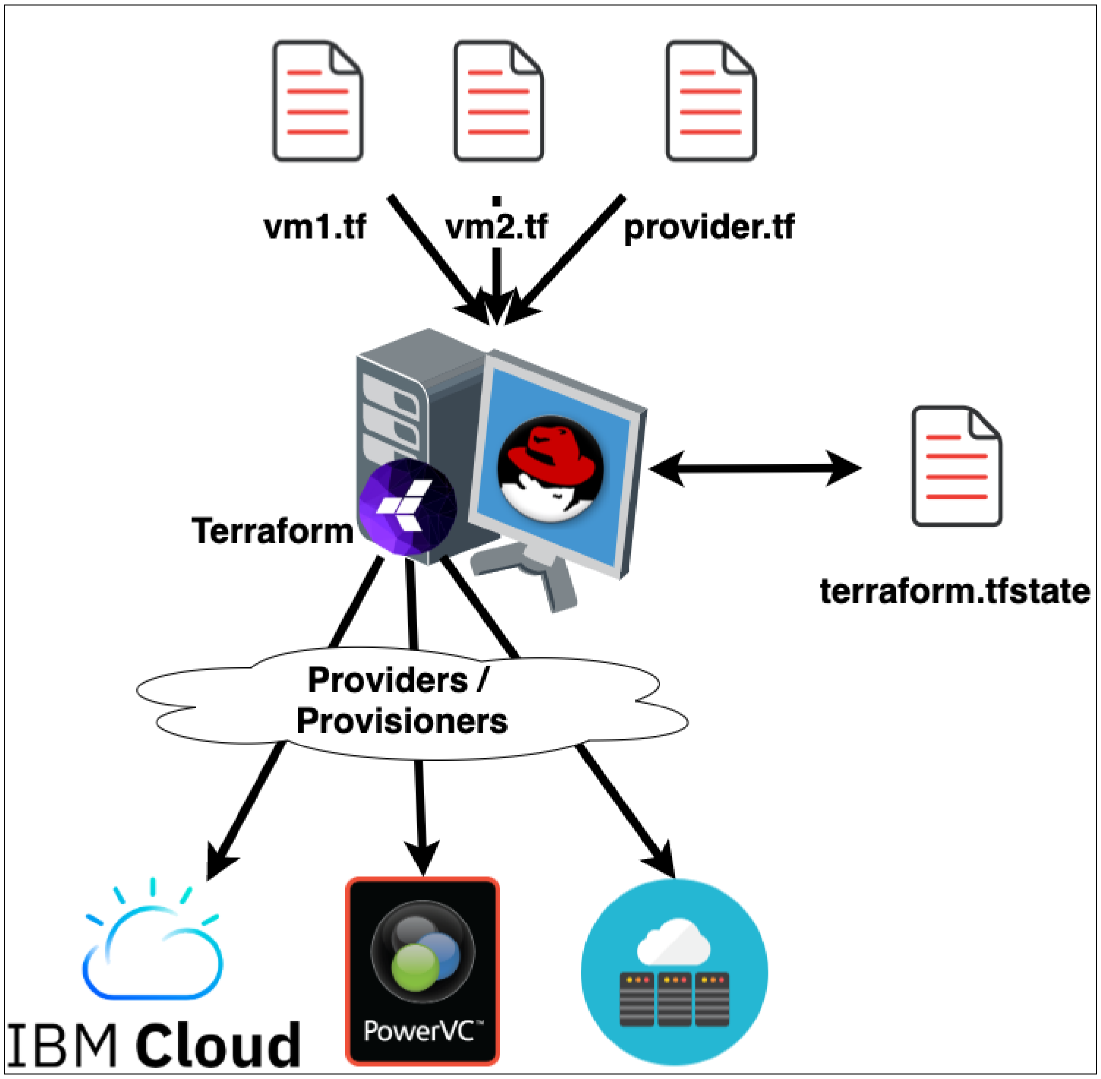


Figure 5-10 Terraform components

Terraform features the following files:

- Configuration files(`.tf`): Terraform uses its own configuration language, which allows concise descriptions of infrastructure. The Terraform language is declarative and describing an intended goal rather than the steps to reach that goal.
- Terraform binary (executable) file: This file is written and compiled in GO language. To install Terraform, find the appropriate package for your system and download it from this web page.

> **Note:** Terraform can run on any platform (including x86) to provision resources. If you have a ppc64le platform, you must compile Terraform and all needed providers (plug-ins).

- Terraform state file (`.tfstate`): This JSON state file contains information about the provisioned infrastructure that Terraform manages.

OpenShift-Ansible

Ansible is a simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.

Ansible uses no agents and no other custom security infrastructure, so it is easy to deploy and most importantly, it uses a simple language (YAML, in the form of Ansible Playbooks) with which you can describe your automation jobs in a way that approaches plain English.

Ansible works by connecting to your nodes and pushing out small programs, called *Ansible modules* to them. These programs are written to be resource models of the wanted state of the system. Ansible then runs these modules (over SSH by default), and removes them when finished. Passwords are supported, but SSH keys with ssh-agent are one of the best ways to use Ansible.

Note: Ansible 2.6 is required for OpenShift 3.11.

OpenShift-Ansible is a set of Ansible playbooks that orchestrate complex deployment tasks, including the following examples:

- Configuring the container runtime environment on virtual machines
- Provisioning storage for an internal registry
- Configuring the OpenShift SDN
- Connecting to authentication systems

Note: Consider the following points:

- OpenStack Provisioning is not supported by PowerVC. OpenShift-Ansible deploys OpenShift on OpenStack by using Heat service, but Heat is not implemented in PowerVC.
- OpenShift-Ansible can also deploy OpenShift directly onto physical servers. This reference architecture deploys to virtual machines as that is the more common deployment model.

Load balancers

This guide uses an external load balancer that is running HAproxy to offer a single entry point for the many Red Hat OpenShift Container Platform components. Organizations can provide their own deployed load balancers if the service exists.

The Red Hat OpenShift Container Platform console, which is provided by the Red Hat OpenShift Container Platform master nodes, can be spread across multiple instances to provide load balancing and HA properties.

Application traffic passes through the Red Hat OpenShift Container Platform Router on its way to the container processes. The Red Hat OpenShift Container Platform Router is a reverse proxy service container that multiplexes the traffic to multiple containers that make up a scaled application that is running inside Red Hat OpenShift Container Platform. The load balancer that is used by infrastructure nodes acts as the public view for the Red Hat OpenShift Container Platform applications.

The destination for the master and application traffic must be set in the load balancer configuration after each instance is created, the floating IP address is assigned, and before the installation. A single HAproxy Load Balancer can forward both sets of traffic to different destinations.

When configuring multiple masters, the cluster installation process supports the native HA method. This method uses the native HA master capabilities that are built into OpenShift Container Platform and can be combined with any Load Balancing solution.

If a host is defined in the `[lb]` section of the inventory file, Ansible installs and configures HAProxy automatically as the load balancing solution. If no host is defined, it is assumed that you pre-configured an external load balancing solution of your choice to balance the master API (port 8443) on all master hosts.

Note: The HAProxy Load Balancer is intended to demonstrate the API server's HA mode and is not recommended for production environments. If you are deploying to a cloud provider, Red Hat recommends deploying a cloud-native TCP-based Load Balancer or take other steps to provide a highly available load balancer.

DNS

DNS service is an important component in the Red Hat OpenShift Container Platform environment. Regardless of the provider of DNS, an organization is required to have certain records in place to serve the various Red Hat OpenShift Container Platform components.

Considering the Load Balancer values for the Red Hat OpenShift Container Platform master service and infrastructure nodes running router Pods are known beforehand, entries must be configured into the DNS before starting the deployment procedure.

DNS for OpenShift applications

Applications that are served by OpenShift are accessible by the router on ports 80/TCP and 443/TCP. The router uses a wildcard record to map all host names under a specific sub domain to the same IP address without requiring a separate record for each name. This process allows Red Hat OpenShift Container Platform to add applications with arbitrary names if they are under that sub domain.

For example, a wildcard record for `*.apps.example.com` causes DNS name lookups for `app1.apps.example.com` and `app2.apps.example.com` to both return the same IP address: `9.109.x.y`. All traffic is forwarded to the OpenShift Infrastructure Nodes (Routers). The Routers examine the HTTP headers of the queries and forward them to the correct destination.

With a load-balancer host address of `9.109.x.y`, the wildcard DNS record for `*.apps.example.com` resolves IP address `9.109.x.y`.

A simple DNS round-robin resolution can be used to spread traffic across infrastructure nodes.

For production environments, it is recommended to have more advanced load balancing capabilities to distribute the traffic among the OpenShift Routers. In those cases, an external Load Balancer is used.

OpenShift Software Defined Networking (SDN)

Red Hat OpenShift Container Platform offers the ability to specify how pods communicate with each other. This process can be done by using Red Hat provided Software-defined networks (SDN) or a third-party SDN.

Deciding on the suitable internal network for an Red Hat OpenShift Container Platform step is a crucial step. Unfortunately, no correct answer exists regarding the suitable pod network to chose because this choice varies based on the specific scenario requirements for how a Red Hat OpenShift Container Platform environment is to be used.

For the purposes of this reference environment, the Red Hat OpenShift Container Platform ovs-networkpolicy SDN plug-in is chosen because of its ability to provide Pod isolation by using Kubernetes NetworkPolicy.

Next, we discuss the important points to consider when deciding between the three popular options for the internal networks: ovs-multitenant, ovs-networkpolicy, and ovs-subnet.

OpenShift SDN plug-ins

This section focuses on multiple plug-ins for pod communication within Red Hat OpenShift Container Platform by using OpenShift SDN. The following plug-in options are available:

- ovs-subnet

  The original plug-in that provides an overlay network that is created to allow Pod-to-Pod communication and services. This pod network is created by using Open vSwitch (OVS).

- ovs-multitenant

  A plug-in that provides an overlay network that is configured by using OVS, which is similar to the ovs-subnet plug-in. However, unlike the ovs-subnet, this plug-in provides Red Hat OpenShift Container Platform project-level isolation for pods and services.

- ovs-networkpolicy

  A plug-in that provides an overlay network that is configured by using OVS that provides the ability for Red Hat OpenShift Container Platform administrators to configure specific isolation policies by using NetworkPolicy objects.

Choosing an OpenShift SDN option

Two other OpenShift SDN options are available: ovs-multitenant and ovs-networkpolicy. The reason ovs-subnet is ruled out as an OpenShift SDN option is because it does not feature network isolation.

Although ovs-multitenant and ovs-networkpolicy provide network isolation, the optimal choice comes down to what type of isolation is required. The ovs-multitenant plug-in provides project-level isolation for pods and services. Therefore, pods and services from different projects cannot communicate with each other.

However, ovs-networkpolicy solves network isolation by providing project administrators the flexibility to create their own network policies by using Kubernetes NetworkPolicy objects. That is, all pods in a project are accessible from other pods and network endpoints by default until NetworkPolicy objects are created. Then, Pods from separate projects can communicate with each other, assuming the suitable NetworkPolicy is in place.

Depending on the level of isolation that is required, you must determine the suitable choice when deciding between ovs-multitenant and ovs-networkpolicy.

OpenShift storage

Container storage is ephemeral by design. Initially, containers were designed for immutable and stateless workloads. Later, the advantages of containerizing stateful applications became apparent. With that realization came the need to support persistent storage.

A similar paradigm occurred with Kubernetes. Initially, it was designed for stateless applications, but it was rapidly extended to support stateful workloads. Supporting these new types of workloads drove the need to support multiple storage options. The storage options for Kubernetes and OpenShift environments are grouped under two classifications: ephemeral storage and persistent storage.

Ephemeral storage

Container images are stored locally on the nodes running Red Hat OpenShift Container Platform pods.

When Docker run time is used, the `/var/lib/docker` mount point is used by active containers and pods. This local storage is where the node maintains a copy of container images that are pulled from a container registry. This mount point is managed by docker-storage and it uses the following naming format: `/var/lib/docker/overlay2/<layer-id>` and `/var/lib/docker/containers/<container-id>`.

Persistent storage

Persistent Volume Claims (PVC) are used to store the application data. These claims can be added to the environment manually or provisioned dynamically by using a StorageClass object.

Storage classes

The StorageClass resource object describes and classifies different types of storage that can be requested. It also provides a means for passing parameters to the backend for dynamically provisioned storage on demand.

StorageClass objects can also serve as a management mechanism for controlling different levels of storage and access to the storage. Cluster Administrators (cluster-admin) or Storage Administrators (storage-admin) define and create the StorageClass objects that users can use without needing any intimate knowledge about the underlying storage volume sources. Therefore, the naming of the storage class that is defined in the StorageClass object must be useful in understanding the type of storage it maps, whether that is storage from PowerVC Cinder or from other storage provider.

Persistent Volumes

PersistentVolumes (PV) objects provide pods with non-ephemeral storage by configuring and encapsulating underlying storage sources. A PersistentVolumeClaim (PVC) abstracts an underlying PV to provide provider-independent storage to OpenShift resources. When successfully fulfilled by the system, a PVC mounts the persistent storage to a specific directory (mountPath) within one or more pods. From the container perspective, the mountPath is connected to the underlying storage mount points by a regular bind mount.

FlexVolumes

FlexVolume is known as an *out-of-tree plug-in interface* because it is developed outside the main Kubernetes source code. The FlexVolume interface enables users to write their own drivers. These drivers can be written in any programming or scripting language.

When an application that is running on OpenShift needs a persistent volume, it submits a persistent volume claim to the PowerVC FlexVolume driver. The PowerVC FlexVolume call is translated into a Cinder API call to create a volume. When the volume is ready, it is presented back to OpenShift and attached to the requesting pod.

The persistent volume claim needs to include only the volume size and access mode. The backend implementation information about how and where the volume is created are handled by PowerVC. The OpenShift API abstracts them from the user that is making the resource claim.

Note: For more information about the PowerVC FlexVolume, see this web page.

Registry

OpenShift can build container images from source code, deploy them, and manage their lifecycle. To enable this process, OpenShift provides an internal, integrated registry that can be deployed in the OpenShift environment to manage images.

The registry stores images and metadata. For production environments, persistent storage must be used for the registry; otherwise, any images that were built or pushed into the registry disappear if the pod restarts.

Aggregated logging

One of the Red Hat OpenShift Container Platform optional components is named Red Hat OpenShift Container Platform aggregated logging. This component collects and aggregates logs from the pods that are running in the Red Hat OpenShift Container Platform cluster and `/var/log/messages` on nodes. This configuration enables Red Hat OpenShift Container Platform users to view the logs of projects that they can view by using a web interface.

Red Hat OpenShift Container Platform aggregated logging component is a modified version of the ELK stack, which is composed of a few pods that are running on the Red Hat OpenShift Container Platform environment:

- Elasticsearch: An object store where all logs are stored.
- Kibana: A web UI for Elasticsearch.
- Curator: Elasticsearch maintenance operations that are performed automatically on a per-project basis.
- Fluentd: Gathers logs from nodes and containers and feeds them to Elasticsearch.

Consider the following basic concepts for aggregated logging:

- Cluster: A set of Elasticsearch nodes that distribute the workload.
- Node: A container that is running an instance of Elasticsearch, which is part of the cluster.
- Index: Collection of documents (container logs).
- Shards and Replicas: Indexes can be divided into sets of data that contain the primary copy of the documents that are stored (primary shards) or backups of that primary copies (replica shards). Sharding allows the application to horizontally scale the information and distributed/paralellized operations. Replication instead provides HA and also better search throughput because searches are also run on replicas.

Note: Using NFS storage as a volume or a persistent volume (or by way of an NAS such as Gluster) is not supported for Elasticsearch storage because Lucene relies on file system behavior that NFS does not supply. Data corruption and other problems can occur.

Red Hat OpenShift Container Platform can gather metrics from kubelet and store the values in Heapster. Red Hat OpenShift Container Platform Metrics provide the ability to view CPU, memory, and network-based metrics and display the values in the user interface. These metrics can allow for the horizontal autoscaling of pods based on parameters that are provided by a Red Hat OpenShift Container Platform user. It is important to understand capacity planning when metrics are deployed into an Red Hat OpenShift Container Platform environment.

Red Hat OpenShift Container Platform metrics is composed by the following pods that are running on the Red Hat OpenShift Container Platform environment:

- Heapster: Heapster scrapes the metrics for CPU, memory, and network usage on every Pod. Then, it exports them into Hawkular Metrics.

- Hawkular Metrics: A metrics engine that stores the data persistently in a Cassandra database.
- Cassandra: Database where the metrics data is stored.

Red Hat OpenShift Container Platform metrics components can be customized for longer data persistence, pods limits, replicas of individual components, custom certificates, and so on.

As a best practice, persistent storage must be used to allow for metric data to be preserved when metrics are deployed. Node selectors must be used to specify where the metrics components must run.

5.3.3 Reference architecture

This section describes a reference architecture for running OpenShift Container Platform 3.11 on PowerVC. In this reference architecture the OpenShift Container Platform is installed on virtual machines that are provisioned by PowerVC to the IBM Power Systems, as shown in Figure 5-11. The architecture is highly available and suitable for production.

This document describes the reference architecture as it was deployed and tested in a lab environment.

Note: At the time of this writing, a cluster must be x86 or POWER, not mixed.

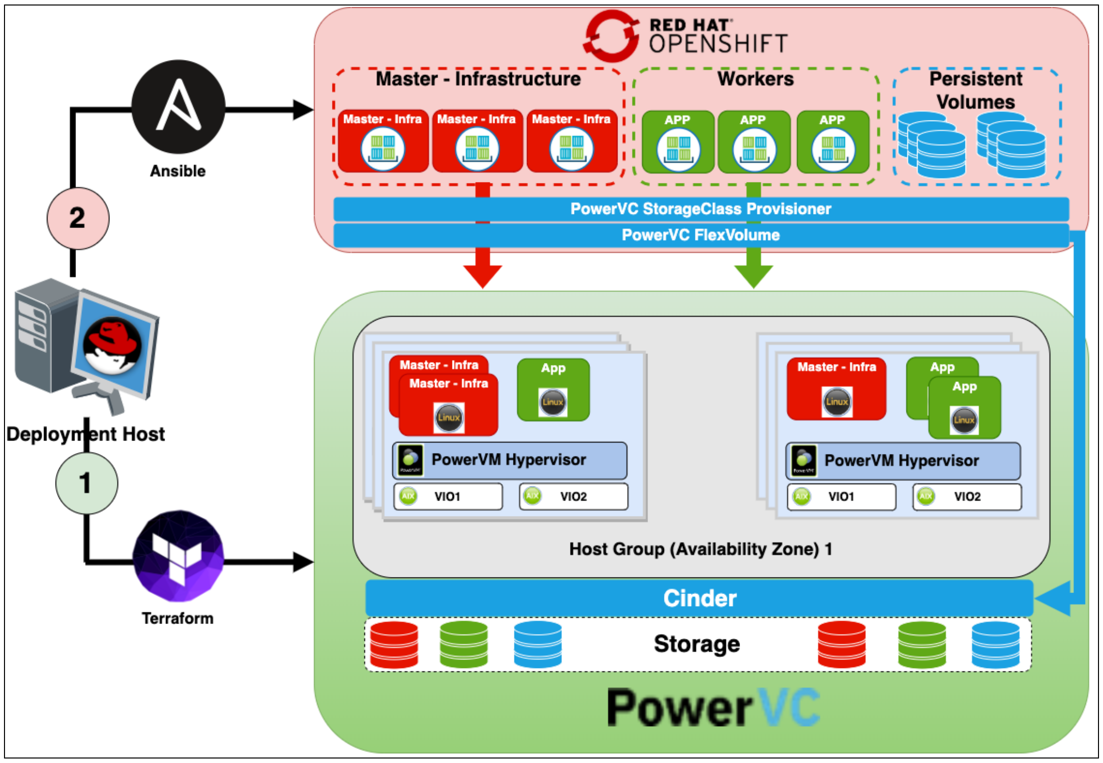


Figure 5-11 OpenShift Container Platform on PowerVC

The relationship between PowerVC and OpenShift is complementary. Consider the following points:

- PowerVC exposes resources that OpenShift uses.
- OpenShift runs its containerized applications on the infrastructure that is provisioned by PowerVC.

Table 5-7 lists the minimum resources that are required for OpenShift Container Platform 3.11 virtual machines. *eCPU* refers to the number of processing units of entitled capacity.

Table 5-7 OpenShift nodes roles and configurations

Role	Count	vCPU	eCPU	Memory (GB)	rootvg (GB)	dockervg (GB)
Deployment	1	2	0.4	8	64	N/A
Load Balancer	1	2	0.4	8	64	N/A
Master-Infra	3	8	1.6	32	64	128
Worker	3	4	0.8	16	64	128

The architecture consists of the following components:

- PowerVC: Provides Infrastructure as a Service and manages:
  - IBM Power Systems (PowerVM):
    - 2 x Virtual I/O Servers: 2 cores that are configured in dedicated donated mode and 16 GB memory
    - 2 x Ethernet cards
    - 2 x Fibre Channel cards for each VIO Server
  - Dual SAN Fabrics Brocade or Cisco
  - Storage backends
- Deployment host: RHEL 7.6 for POWER8 and RHEL-ALT 7.6 for POWER9 minimum installation. Responsible for deploying one or more OpenShift Container Platform environments:
  - Terraform for deploying PowerVC infrastructure (network, storage, virtual machines)
  - OpenShift-Ansible for deploying OpenShift Container Platform
- Load balancer: RHEL 7.6 for POWER8 and RHEL-ALT 7.6 for POWER9 minimum installation. HAproxy offers a single entry point for the many Red Hat OpenShift Container Platform components.

  Note: The HAProxy Load Balancer is intended to demonstrate the API server's HA mode and is not recommended for production environments. If you are deploying to a cloud provider, Red Hat recommends deploying a cloud-native TCP-based Load Balancer or take other steps to provide a highly available Load Balancer.

- Master-Infrastructure: RHEL 7.6 for POWER8 and RHEL-ALT 7.6 for POWER9 minimum installation. Responsible for running the OpenShift master and infrastructure components, including the API server, controller manager server, `etcd`, registry, and routers.
- Worker: RHEL 7.6 for POWER8 and RHEL-ALT 7.6 for POWER9 minimum installation. Responsible for running Application containers that are created by the users of the OpenShift service.

6

Installing Red Hat OpenShift 3.11 on IBM PowerVC

This chapter describes the Red Hat OpenShift 3.11 deployment process on PowerVC by using Terraform.

This chapter includes the following topics:

6.1 Deployment process overview

This section describes the Red Hat OpenShift 3.11 deployment process on PowerVC by using Terraform. The process includes the following steps:

1. Setting up the deployment environment:
   - Configure DNS.
   - Configure PowerVC user and project.
   - Create the virtual machine to host deployment tools.
   - Prepare the deployment host and install Terraform and OpenShift-Ansible.
2. OpenShift Container Platform Deployment:
   - Infrastructure Deployment by using Terraform. Terraform deploys the PowerVC network, virtual machines, and storage. It uses the OpenStack provider (plug-in) to build resources directly to the PowerVC APIs.
   - OpenShift Container Platform Installation using OpenShift-Ansible. OpenShift-Ansible is a set of Ansible playbooks that orchestrate complex deployment tasks for installing the OpenShift Container Platform on the physical or virtual servers.

Note: The Deployment Host can be any host or virtual machine that is running an operating system that is supported by Terraform and OpenShift-Ansible tools. This chapter describes how to install and configure the Deployment Host on PowerVC running Red Hat ALT 7.6 (ppc64le) operating system.

6.2 Setting up the deployment environment

This section shows how to set up the environment.

6.2.1 Setting up the DNS

OpenShift Container Platform requires a fully functional DNS server in the environment. A set of records must be configured in your DNS to provide name resolution for hosts and containers running on the platform. Optionally, you can also configure a wildcard DNS record that points to the load balancer or routers to avoid the need to update your DNS configuration when new routes are added.

Example 6-1 shows a sample DNS Masquerade configuration.

Example 6-1 DNS Masquerade configuration

```
# cat > /etc/dnsmasq.conf <<EOF_/etc/dnsmasq.conf
conf-dir=/etc/dnsmasq.d,.rpmnew,.rpmsave,.rpmorig

strict-order
domain-needed
local=/domain.example.com/
bind-dynamic
log-queries

resolv-file=/etc/resolv.dnsmasq
###----------------------------ocp-aio-----------------------------------###
address=/bsocp01.domain.example.com/192.168.11.223
ptr-record=223.97.108.9.in-addr.arpa,bsocp01.domain.example.com
```

```
address=/apps.bs.ibm.com/192.168.11.223
###----------------------------ocp-aio-----------------------------------###
###----------------------------ocp-7nodes--------------------------------###
# Deployment Node
address=/dplnode01.domain.example.com/192.168.11.220
ptr-record=220.97.108.9.in-addr.arpa,dplnode01.domain.example.com
# Master-Infra Nodes
address=/mstnode01.domain.example.com/192.168.11.202
ptr-record=202.98.108.9.in-addr.arpa,mstnode01.domain.example.com
address=/mstnode02.domain.example.com/192.168.11.203
ptr-record=203.98.108.9.in-addr.arpa,mstnode02.domain.example.com
address=/mstnode03.domain.example.com/192.168.11.204
ptr-record=204.98.108.9.in-addr.arpa,mstnode03.domain.example.com
# Worker Nodes
address=/wrknode01.domain.example.com/192.168.11.208
ptr-record=208.98.108.9.in-addr.arpa,wrknode01.domain.example.com
address=/wrknode02.domain.example.com/192.168.11.209
ptr-record=209.98.108.9.in-addr.arpa,wrknode02.domain.example.com
address=/wrknode03.domain.example.com/192.168.11.210
ptr-record=210.98.108.9.in-addr.arpa,wrknode03.domain.example.com
# Load Balancer Node
address=/lbsnode01.domain.example.com/192.168.11.212
ptr-record=212.98.108.9.in-addr.arpa,lbsnode01.domain.example.com
# Cluster and  wildcard DNS
address=/ocp.domain.example.com/192.168.11.212
address=/apps.domain.example.com/192.168.11.212
###----------------------------ocp-7nodes--------------------------------###
EOF_/etc/dnsmasq.conf
###Configure firewallD to allow DNS####
# firewall-cmd --permanent --add-service=rmc
# firewall-cmd --reload
###Restart DNS Masqurade service ######
# systemctl restart dnsmasq
```

6.2.2 PowerVC configuration

The next steps show how to configure a new user and a project (tenant) in PowerVC to separate OpenShift resources for other resources. This is a logical isolation, and is not required for installation; therefore, you can use any user or project. For the configuration, you can use any PowerVC interface, although this scenario uses only the CLI interface.

Complete the following steps:

1. Set the access variables:

   ```
   source /opt/ibm/powervc/powervcrc
   export OS_USERNAME=root
   export OS_PASSWORD=<SECRET>
   export OS_PROJECT_NAME=<Project Name> #Default is ibm-default
   ```

2. Create a project:

   ```
   openstack project create --description "OpenShift Container Platform" ocp-project
   openstack project create --description "OpenShift Container Platform" ocp-project
   +-------------+----------------------------------+
   | Field       | Value                            |
   +-------------+----------------------------------+
   | description | OpenShift Container Platform     |
   | domain_id   | default                          |
   ```

```
| enabled     | True                             |
| id          | 385ab82655074660b07bb0757e116e39 |
| is_domain   | False                            |
| name        | ocp-project                      |
| parent_id   | default                          |
| tags        | []                               |
+-------------+----------------------------------+
```

3. Create a user and assign admin role for the ocp-project:

```
groupadd -g 3030 ocpadmin
useradd -g ocpadmin -u 3030 -d /home/ocpadmin -c "OpenShift Container Platform
Admin"  ocpadmin
echo "ocpadmin:<password>" | chpasswd
usermod -a -G powervc-filter ocpadmin
openstack role add --project ocp-project --user ocpadmin admin
```

6.2.3 Creating the virtual machine to host deployment tools

Complete the following steps to create a virtual machine (LPAR) by using PowerVC CLI:

1. Set PowerVC access variables using the new user and project:

```
source /opt/ibm/powervc/powervcrc
export OS_PROJECT_NAME=ocp-project
export OS_USERNAME=ocpadmin
export OS_PASSWORD=<password>
openstack project list
+----------------------------------+-------------+
| ID                               | Name        |
+----------------------------------+-------------+
| 385ab82655074660b07bb0757e116e39 | ocp-project |
+----------------------------------+-------------+
```

2. List the flavors:

```
openstack flavor list
+--------------------------------------+---------+--------+------+-----------+-------+-----------+
| ID                                   | Name    |    RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------------------------+---------+--------+------+-----------+-------+-----------+
| 16f4c456-debe-4b7f-a59e-d024667fb74b | medium  |  16384 |    0 |         0 |     4 | True      |
| 3f2f851b-1ae9-4604-bde9-f81984a924fa | xxlarge | 131072 |    0 |         0 |    32 | True      |
| 43032930-4dfb-417e-9501-80b5408076fc | large   |  32768 |    0 |         0 |     8 | True      |
| 962979ba-2ee7-464f-a0ea-856248765758 | tiny    |   4096 |    0 |         0 |     1 | True      |
| d7409716-49e2-426d-acb1-9de141b8d8ea | small   |   8192 |    0 |         0 |     2 | True      |
| e93908f5-b6f9-4bb9-bbff-9136c7a80211 | xlarge  |  65536 |    0 |         0 |    16 | True      |
+--------------------------------------+---------+--------+------+-----------+-------+-----------+
```

3. List the images:

```
openstack image list
+--------------------------------------+---------------------+--------+
| ID                                   | Name                | Status |
+--------------------------------------+---------------------+--------+
| 09ba0030-b6c3-4631-b9f9-19eb3333289c | xiv_p8_image_rhel76 | active |
| 77ad197b-0a4e-4792-a3c4-ea634ffa0fd3 | xiv_p9_image_rhel76 | active |
+--------------------------------------+---------------------+--------+
```

4. List the networks:

```
openstack network list
+--------------------------------------+------------------+--------------------------------------+
| ID                                   | Name             | Subnets                              |
+--------------------------------------+------------------+--------------------------------------+
| 7e2fe367-ade6-424c-967e-893930c5b80c | GDL-VLAN1        | 1fa1fb36-1bec-450c-a94a-3ea3cdb56975 |
+--------------------------------------+------------------+--------------------------------------+
```

5. Create the new server (VM):

```
openstack server create  \
openstack server create  \
--flavor $(openstack flavor list | awk '/ small / {print $2}') \
--image $(openstack image list | awk '/ xiv_p9_image_rhel76 / {print $2}') \
--nic net-id="$(openstack network list|awk '/ VLAN1 / {print $2}'),v4-fixed-ip=x.x.x.x"\
--wait dplnode-01
```

6. List servers (VMs):

```
openstack server list
+----------------------------------+------------+--------+-----------------------+---------------------+-------+
| ID | Name       | Status | Networks               | Image                | Flavor
|+----------------------------------+------------+--------+-----------------------+---------------------+------+
|2f5b8ed8-53e5-406f-93ed-cefddaa0002d| dplnode-01 | ACTIVE | GDL-VLAN1=192.168.11.220 | xiv_p9_image_rhel76 |small
+----------------------------------+------------+--------+-----------------------+---------------------+-------+
```

6.2.4 Preparing the deployment host

For the configuration, you connect by way of SSH as root on the deployment host VM and complete the following steps:

1. Configure the Red Hat Subscription Manager (RHSM) registration:

   a. Register the host to RHSM:

```
subscription-manager register --username=<username> --password=<password>
Registering to: subscription.rhsm.redhat.com:443/subscription
The system has been registered with ID: e214ca01-fbed-43a0-aed5-d46150a5f912
The registered system name is: dplnode01.domain.example.com
```

   b. Identify the available OpenShift subscriptions:

```
subscription-manager refresh
All local data refreshed
subscription-manager list --available --matches '*OpenShift*'
Subscription Name:   Red Hat OpenShift Container Platform for Power, LE Business Partner
NFR, Self-Supported
Provides:            Red Hat Enterprise Linux for Power, little endian - Extended Update
Support
                     Red Hat Enterprise Linux Fast Datapath Beta for Power, little
endian
                     Red Hat Enterprise Linux for Power, little endian
                     Red Hat Ansible Engine
                     Red Hat OpenShift Enterprise Application Node
                     Red Hat Enterprise Linux for Power 9
                     Red Hat Software Collections (for RHEL Server for IBM Power LE)
                     Red Hat OpenShift Container Platform for Power
                     Red Hat Software Collections Beta (for RHEL Server for IBM Power
LE)
                     RHEL for SAP HANA for Power, little endian - Extended Update
Support
                     Red Hat Beta
                     Red Hat OpenShift Container Platform Client Tools for Power
                     Red Hat Enterprise Linux Fast Datapath (for RHEL Server for IBM
Power LE)
                     RHEL for SAP for Power, little endian - Extended Update Support
                     Red Hat Enterprise Linux for Power, little endian Beta
                     Red Hat Container Native Virtualization
                     Red Hat CodeReady Linux Builder for Power, little endian - Extended
Update Support
SKU:                 111111111
Contract:            111111111
Pool ID:             <POOL_ID>
Provides Management: No
Available:           Unlimited
Suggested:           1
Service Level:       Standard
Service Type:        L1-L3
Subscription Type:   Stackable
Starts:              05/31/2019
Ends:                05/31/2020
System Type:         Virtual
```

c. Assign the OpenShift subscription:

```
subscription-manager attach --pool=<POOL_ID>
Successfully attached a subscription for: Red Hat OpenShift Container Platform for
Power, LE Business Partner NFR, Self-Supported
```

d. Enable only the repositories that are required by OpenShift Container Platform 3.11. For IBM POWER9, run the commands that are shown in Example 6-2. For IBM POWER8, run the commands that are shown in Example 6-3 on page 107.

Example 6-2 OpenShift repositories for POWER9 servers

```
# subscription-manager repos --disable="*"
# subscription-manager repos \
    --enable="rhel-7-for-power-9-rpms" \
    --enable="rhel-7-for-power-9-extras-rpms" \
    --enable="rhel-7-for-power-9-optional-rpms" \
    --enable="rhel-7-server-ansible-2.6-for-power-9-rpms" \
    --enable="rhel-7-server-for-power-9-rhscl-rpms" \
    --enable="rhel-7-for-power-9-ose-3.11-rpms"
```

Example 6-3 OpenShift repositories for POWER8 servers

```
# subscription-manager repos --disable="*"
# subscription-manager repos \
    --enable="rhel-7-for-power-le-rpms" \
    --enable="rhel-7-for-power-le-extras-rpms" \
    --enable="rhel-7-for-power-le-optional-rpms" \
    --enable="rhel-7-server-ansible-2.6-for-power-le-rpms" \
    --enable="rhel-7-server-for-power-le-rhscl-rpms" \
    --enable="rhel-7-for-power-le-ose-3.11-rpms"
```

2. Install OpenShift-Ansible:

```
yum clean all
yum -y update --security --exclude=cloud-init\*
yum -y install atomic-openshift-clients openshift-ansible
cp /usr/share/ansible/openshift-ansible/ansible.cfg /etc/ansible/ansible.cfg
```

3. Install Terraform.

Important: At the time this writing, Terraform is not available to download for the ppc64le platform. This section compiles it from the source code. You can check its availability at this web page.

4. Complete the following next steps to compile Terraform and the plug-ins needed for PowerVC:

   a. Install git, tftp, gcc, zip:

```
yum install -y git gcc tftp zip
```

b. Install GO from source:

```
###### Step 1. Grab yourself a binary release from here: https://golang.org/dl/
# wget https://dl.google.com/go/go1.12.9.linux-ppc64le.tar.gz
###### Step 2. Install ######
# tar -C /usr/local -xzf go1.12.9.linux-ppc64le.tar.gz
###### Step 3. Configure Environment ######
# mkdir ~/.go
# vi ~/.bashrc
###### The go binary, so we can actually run it ######
export PATH=$PATH:/usr/local/go/bin;
###### This is where all your go packages live ######
GOPATH=/root/.go;
export GOPATH;
###### Add GOPATH/bin so compiled go libs appear on your PATH ######
export PATH=$PATH:$GOPATH/bin;

source  ~/.bashrc
```

c. Compile Terraform for the ppc64le platform:

```
###### Clone the terraform repository if needed ######
rm -rf /root/.go/*
cd ~
git clone https://github.com/hashicorp/terraform.git
Cloning into 'terraform'...
remote: Enumerating objects: 78, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (48/48), done.
remote: Total 208686 (delta 42), reused 48 (delta 28), pack-reused 208608
Receiving objects: 100% (208686/208686), 162.04 MiB | 7.23 MiB/s, done.
Resolving deltas: 100% (126413/126413), done.
cd terraform
git checkout v0.12.9
Note: checking out 'v0.12.9'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at abec0ac... v0.12.9
XC_OS=linux XC_ARCH=ppc64le make bin
==> Checking that code complies with gofmt requirements...
GO111MODULE=off go get -u golang.org/x/tools/cmd/stringer
GO111MODULE=off go get -u golang.org/x/tools/cmd/cover
GO111MODULE=off go get -u github.com/golang/mock/mockgen
GOFLAGS=-mod=vendor go generate ./...
2019/10/24 17:28:18 Generated command/internal_plugin_list.go
go: downloading github.com/golang/mock v1.3.1
go: extracting github.com/golang/mock v1.3.1
# go fmt doesn't support -mod=vendor but it still wants to populate the
# module cache with everything in go.mod even though formatting requires
# no dependencies, and so we're disabling modules mode for this right
# now until the "go fmt" behavior is rationalized to either support the
# -mod= argument or _not_ try to install things.
GO111MODULE=off go fmt command/internal_plugin_list.go > /dev/null
==> Removing old directory...
which: no gox in
(/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin:/usr/local/go/bin:/ro
ot/.go/bin)
==> Installing gox...
go: finding github.com/mitchellh/gox v1.0.1
...
Output truncated
...
  adding: terraform (deflated 70%)

==> Results:
total 46M
-rwxr-xr-x. 1 root root 46M Oct 24 17:31 terraform

cp /root/.go/bin/terraform /usr/local/bin/
terraform -version
Terraform v0.12.9
```

d. Compile OpenStack plug-in for the ppc64le platform:

```
mkdir -p $GOPATH/src/github.com/terraform-providers; cd
$GOPATH/src/github.com/terraform-providers
git clone https://github.com/terraform-providers/terraform-provider-openstack
Cloning into 'terraform-provider-openstack'...
remote: Enumerating objects: 189, done.
remote: Counting objects: 100% (189/189), done.
remote: Compressing objects: 100% (163/163), done.
remote: Total 17434 (delta 59), reused 101 (delta 21), pack-reused 17245
Receiving objects: 100% (17434/17434), 11.96 MiB | 1.54 MiB/s, done.
Resolving deltas: 100% (10138/10138), done.
cd $GOPATH/src/github.com/terraform-providers/terraform-provider-openstack
git checkout v1.22.0
Note: checking out 'v1.22.0'.
...
Output truncated
...
HEAD is now at 7dcd493... v1.22.0
XC_OS=linux XC_ARCH=ppc64le make build
==> Checking that code complies with gofmt requirements...
go install
```

e. Compile Null plug-in for the ppc64le platform:

```
mkdir -p $GOPATH/src/github.com/terraform-providers ; cd
$GOPATH/src/github.com/terraform-providers/
git clone https://github.com/terraform-providers/terraform-provider-null.git
Cloning into 'terraform-provider-null'...
...
Output truncated
...
Resolving deltas: 100% (2057/2057), done.
cd $GOPATH/src/github.com/terraform-providers/terraform-provider-null
git checkout v2.1.2
Note: checking out 'v2.1.2'.
...
Output truncated
...
HEAD is now at 8d3d85a... v2.1.2
XC_OS=linux XC_ARCH=ppc64le make build
==> Checking that code complies with gofmt requirements...
go install
```

6.3 OpenShift container platform deployment

This section provides an OpenShift container deployment platform.

6.3.1 Deployment scenarios

This section describes the following most common scenarios that can be used to start deploying OpenShift clusters:

- Single node deployment (all-in-one) is not an officially supported OpenShift deployment. The all-in-one (AIO) configuration is considered a testing or development environment. The Master, Infrastructure and Compute Roles are deployed to a single node (see Figure 6-1).

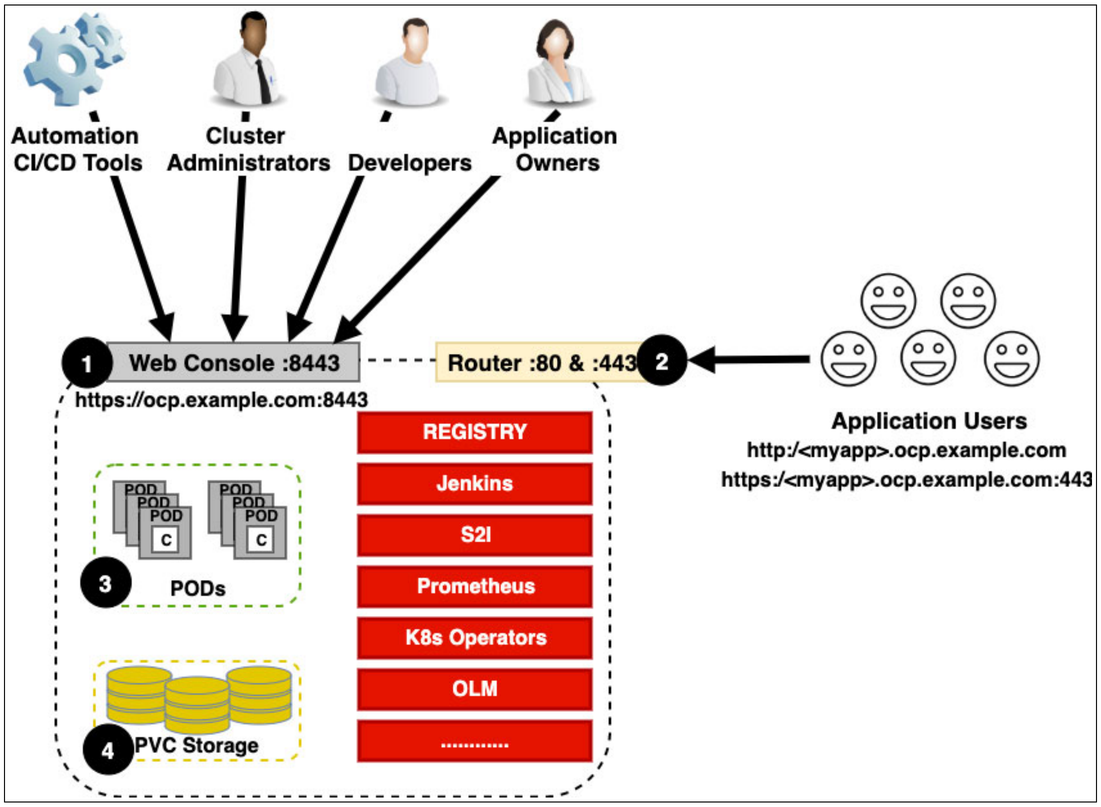


Figure 6-1 OpenShift Container Platform 3.11 all-in-one

- Seven nodes deployment is highly available and suitable for production. The Master and Infrastructure Roles are deployed to three Nodes, the Computer Role is deployed to three Worker Nodes, and the Load Balancer is deployed to a single Node (see Figure 6-2).

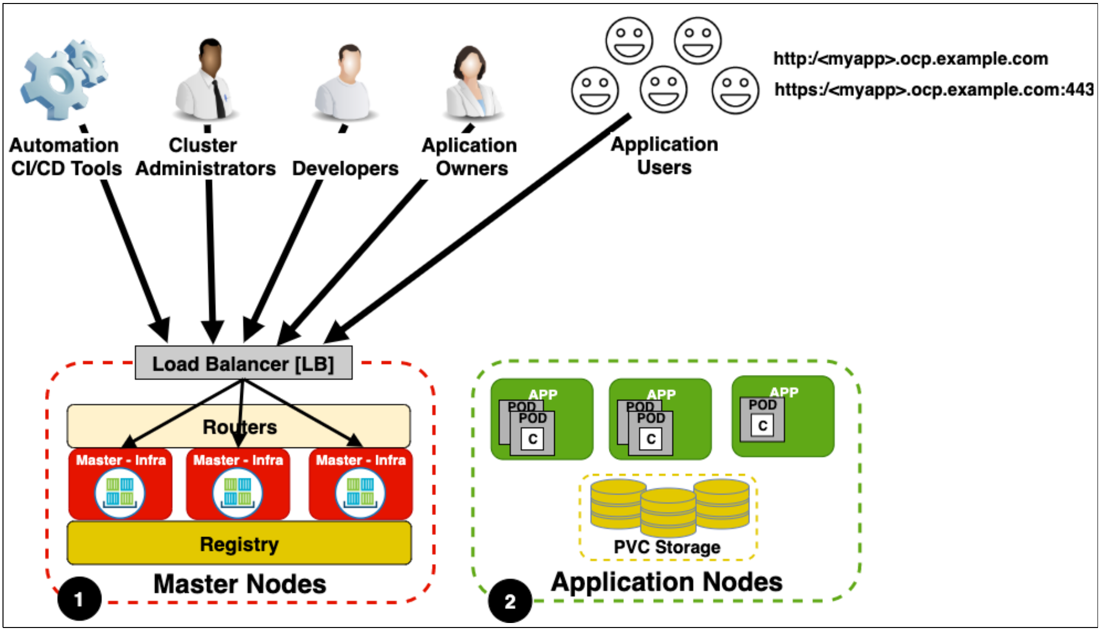


Figure 6-2 OpenShift Container Platform 3.11 6xNodes + Load Balancer

- Three nodes deployment is considered a supported testing and development environment. The Master and Infrastructure Roles are deployed to a single Node, and the Computer Role is deployed to two Worker Nodes (see Figure 6-3).

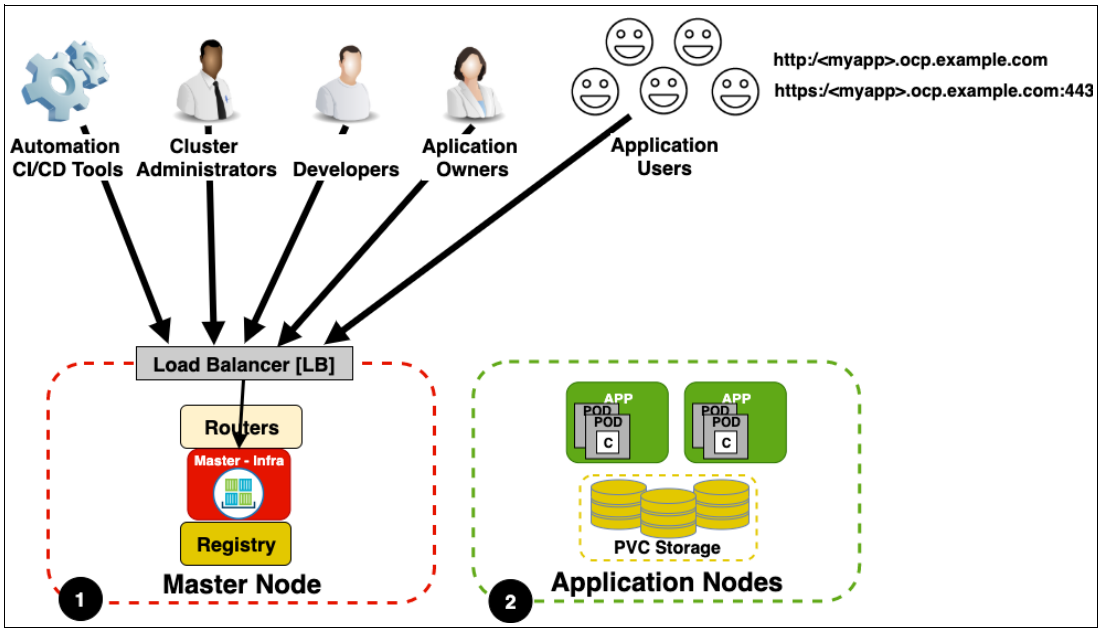


Figure 6-3 OpenShift Container Platform 3xNodes

Important: At the time this publication was written, IBM Cloud Pak for Multicloud Management does not support multiple master nodes. Therefore, the 3xNodes scenario is used for IBM Cloud Pak for Multicloud Management.

6.3.2 Deploying OpenShift Container Platform on PowerVC

Deployment of the OpenShift Container Platform on PowerVC is a two steps process, as shown in Figure 6-4:

1. Provision the infrastructure on PowerVC. Use Terraform to provision the network, compute, and storage on PowerVC.
2. Install OpenShift Container Platform. Use OpenShift-Ansible playbook to install OpenShift Container Platform (OCP) version 3.11.

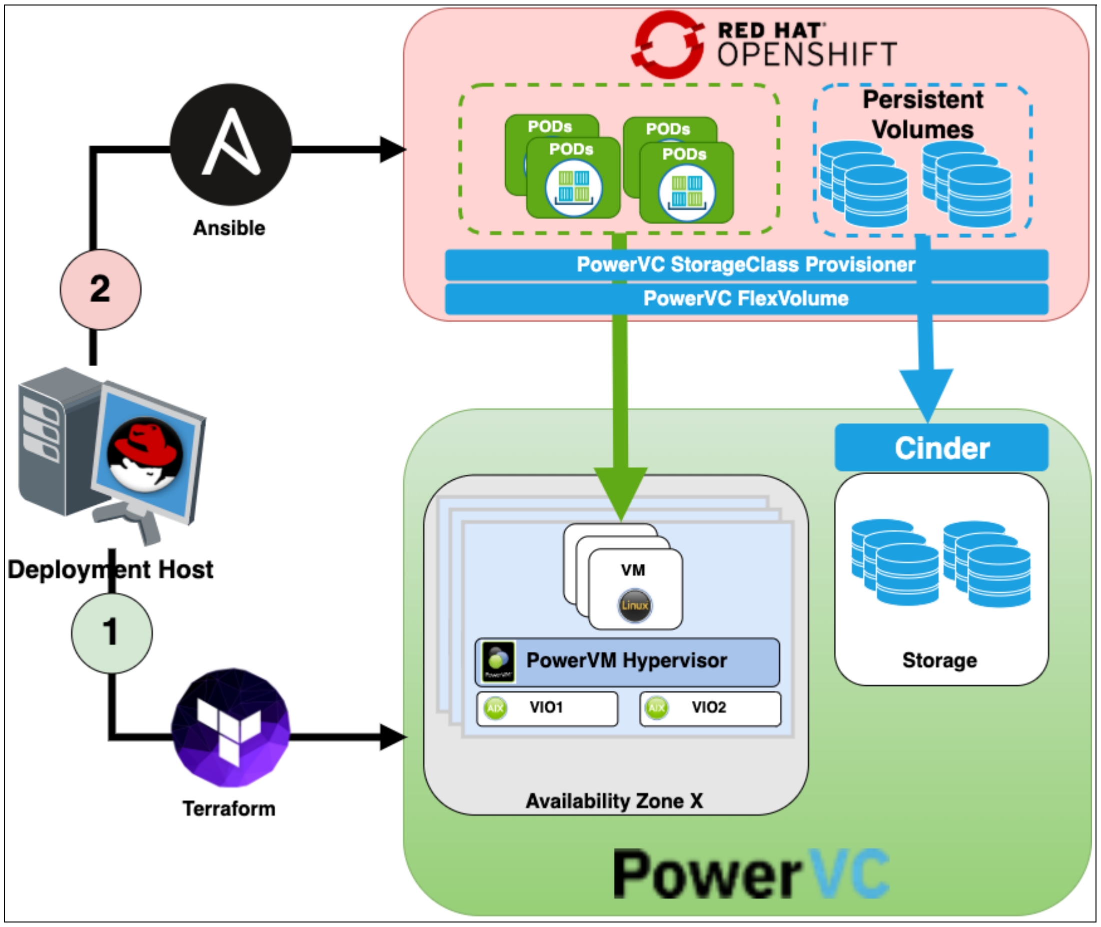


Figure 6-4 Deployment process diagram

The next steps use the following `git` repository, which provides the Terraform templates and configuration examples:

https://bit.ly/2R6nvJP

Complete the following steps:

1. Clone the `git` repository:

```
git clone https://github.com/ppc64le/devops-automation.git
Cloning into 'devops-automation'...
remote: Enumerating objects: 71, done.
remote: Counting objects: 100% (71/71), done.
remote: Total 71 (delta 34), reused 31 (delta 17), pack-reused 0
Unpacking objects: 100% (71/71), done.
```

2. Create a separate folder for each OpenShift Cluster:

```
###All-in-One Cluster
cp devops-automation/terraform/powervc-openshift ocp-aio
###7Nodes Cluster
cp devops-automation/terraform/powervc-openshift ocp-7nodes
```

3. Copy the Terraform plug-ins:

```
cd <PATH> #ocp-aio or ocp-7nodes
mkdir -p .terraform/plugins/linux_ppc64le/
cp $GOPATH/bin/terraform-provider-openstack
.terraform/plugins/linux_ppc64le/terraform-provider-openstack_v1.22.0_x4
cp $GOPATH/bin/terraform-provider-null
.terraform/plugins/linux_ppc64le/terraform-provider-null_v2.1.2_x4
```

6.3.3 Provisioning the infrastructure on PowerVC

Complete the following steps to deploy PowerVC Infrastructure:

Important: Terraform is one option for infrastructure deployment. Many other options are not covered in this book.

1. Create `terrafrom.tfvars` file. You can use the `terraform_aio.tfvars.example` for All-In-One deployment (see Example 6-4), `terraform_7nodes.tfvars.example` for seven nodes deployment (see Example 6-5 on page 116), or `terraform_3nodes.tfvars.example` for three nodes deployment (see Example 6-6 on page 117). For resources configuration, it is recommended to use the values from Table 5-7 on page 99.

Example 6-4 terrafomr.tfvars for All-In-One (AIO) deployment

```
cat terraform.tfvars
#PowerVC (OpenStack)
#---------------------------------
powervc_user        = "ocpadmin"           # PowerVC user
powervc_password    = "<password>"           # PowerVC password
powervc_server      = "192.168.11.31"        # PowerVC IP or hostname
powervc_project     = "ocp-project"         # PowerVC project(tenant) name

#General configuration:
#---------------------------------
ssh_user            = "root"                # Image username
ssh_user_password   = "password"            # Image password
```

```
user_public_key     = "ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAO9+YMqJ8VHX3HC7qy6HSxs3JjTGKbEgK+CExpf811uxsq+uJYbfXEKH19/NCf/U
vpkozJBDDXDIxJ4uqOEBWDG4mUuu5U9a4lXgb6qaPYyXwVTygL/IcBOpoSGEQQaJzhBO5g71uZrya++sG1xHUjSQAQz
hDuKrs4Bc3gcN4184UR+BX1pVgCls3NRn9hLrfLWS37M/kn+b/n6VMYYVpHsZ2XVydAn2nwuzktaEuWYaY/1cNd4xuu
yVuO8GQOon6t5KQ1EZBheADdSsyamulLqW9z4j6Y1wwDe4GPDc5zIW++ASDAZBOeEfbKGDLVdpFsI5YV8nLV1r/TOY/
FiFZqQ== Bogdan Savu;IBMROO45771;IBMROZZ014E826;J;"
dns1                = "192.168.11.210"        # DNS server 1
dns_domain          = "domain.example.com"    # DNS Domain Name

#Network configuration
#---------------------------------
net1_name           = "net_ocp_cluster1"    # Network Name
net1_vlan_id        = "1"                   # VLAN ID
net1_subnet         = "192.168.11.0/21"     # Network/Mask
net1_gateway        = "192.168.11.1"        # Gateway
net1_start          = "192.168.11.223"      # First IP from Pool
net1_end            = "192.168.11.223"      # Last IP from Pool

#VM1 configuration (OCP - Master Nodes)
#---------------------------------
vm1_number         = "1"                    # Number of VMs
vm1_memory         = "32"                   # Memory GB
vm1_cpu            = "8"                    # Virtual CPU
vm1_vcpu_ratio     = "4"                    # vCPU RATIO 1:4 1 vCPU = 0.25 eCPU (cores)
vm1_name           = "bsocp"                # Hostname prefix
vm1_first_ip       = "192.168.11.223"       # Fist IP from a consecutive pool of IPs
vm1_image_name     = "xiv_p9_image_rhel76"  # The image name
vm1_remote_restart = "true"                 # Enable Auto Remote Restart
vm1_storage_name   = "xiv_StoragePool"      # Storage Template
vm1_dockerdisk1    = "0"                    # Docker disk size in GB for ephemeral storage

#VM2 configuration (OCP - Infra Nodes)
#---------------------------------
vm2_number         = "0"                    # Number of VMs
vm2_memory         = "16"                   # Memory GB
vm2_cpu            = "4"                    # Virtual CPU
vm2_vcpu_ratio     = "4"                    # vCPU RATIO 1:4 1 vCPU = 0.25 eCPU (cores)
vm2_name           = "infnode"              # Hostname prefix
vm2_first_ip       = "192.168.11.205"       # Fist IP from a consecutive pool of IPs
vm2_image_name     = "xiv_p9_image_rhel76"  # The image name
vm2_remote_restart = "true"                 # Enable Auto Remote Restart
vm2_storage_name   = "xiv_StoragePool"      # Storage Template
vm2_dockerdisk1    = "68"                   # Docker disk size in GB for ephemeral storage

#VM3 configuration (OCP - Workers(App) Nodes)
#---------------------------------
vm3_number         = "0"                    # Number of VMs
vm3_memory         = "32"                   # Memory GB
vm3_cpu            = "4"                    # Virtual CPU
vm3_vcpu_ratio     = "4"                    # vCPU RATIO 1:4 1 vCPU = 0.25 eCPU (cores)
vm3_name           = "appnode"              # Hostname prefix
vm3_first_ip       = "192.168.11.208"       # Fist IP from a consecutive pool of IPs
vm3_image_name     = "xiv_p9_image_rhel76"  # The image name
vm3_remote_restart = "false"                # Disable Auto Remote Restart
vm3_storage_name   = "xiv_StoragePool"      # Storage Template
vm3_dockerdisk1    = "34"                   # Docker disk size in GB for ephemeral storage

#VM4 configuration (OCP - Load Balancer Node)
#---------------------------------
vm4_number         = "0"                    # Number of VMs
```

```
vm4_memory         = "8"                    # Memory GB
vm4_cpu            = "2"                    # Virtual CPU
vm4_vcpu_ratio     = "4"                    # vCPU RATIO 1:4 1 vCPU = 0.25 eCPU (cores)
vm4_name           = "lbsnode"              # Hostname prefix
vm4_first_ip       = "192.168.11.212"       # Fist IP from a consecutive pool of IPs
vm4_image_name     = "xiv_p9_image_rhel76"  # The image name
vm4_remote_restart = "true"                 # Enable Auto Remote Restart
```

Example 6-5 terrafomr.tfvars for seven nodes deployment

```
cat terraform.tfvars
#PowerVC (OpenStack)
#---------------------------------
powervc_user        = "ocpadmin"            # PowerVC user
powervc_password    = "<password>"          # PowerVC password
powervc_server      = "192.168.11.31"       # PowerVC IP or hostname
powervc_project     = "ocp-project"         # PowerVC project(tenant) name

#General configuration:
#---------------------------------
ssh_user            = "root"                # Image username
ssh_user_password   = "<password>"          # Image password
user_public_key     = "ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAO9+YMqJ8VHX3HC7qy6HSxs3JjTGKbEgK+CExpf811uxsq+uJYbfXEKH19/NCf/U
vpkozJBDDXDIxJ4uqOEBWDG4mUuu5U9a4lXgb6qaPYyXwVTygL/IcBOpoSGEQQaJzhBO5g71uZrya++sG1xHUjSQAQz
hDuKrs4Bc3gcN4184UR+BX1pVgCls3NRn9hLrfLWS37M/kn+b/n6VMYYVpHsZ2XVydAn2nwuzktaEuWYaY/1cNd4xuu
yVuO8GQOon6t5KQ1EZBheADdSsyamulLqW9z4j6Y1wwDe4GPDc5zIW++ASDAZBOeEfbKGDLVdpFsI5YV8nLV1r/TOY/
FiFZqQ== Bogdan Savu;IBMRO045771;IBMROZZ014E826;J;"
dns1                = "192.168.11.210"      # DNS server 1
dns_domain          = "domain.example.com"  # DNS Domain Name

#Network configuration
#---------------------------------
net1_name           = "net_ocp_cluster2"    # Network Name
net1_vlan_id        = "1"                   # VLAN ID
net1_subnet         = "192.168.11.0/21"     # Network/Mask
net1_gateway        = "192.168.11.1"        # Gateway
net1_start          = "192.168.11.202"      # First IP from Pool
net1_end            = "192.168.11.212"      # Last IP from Pool

#VM1 configuration (OCP - Master Nodes)
#---------------------------------
vm1_number         = "3"                    # Number of VMs
vm1_memory         = "64"                   # Memory GB
vm1_cpu            = "8"                    # Virtual CPU
vm1_vcpu_ratio     = "2"                    # vCPU RATIO 1:2 1 vCPU = 0.5 eCPU (cores)
vm1_name           = "mstnode"              # Hostname prefix
vm1_first_ip       = "192.168.11.202"       # Fist IP from a consecutive pool of IPs
vm1_image_name     = "xiv_p9_image_rhel76"  # The image name
vm1_remote_restart = "true"                 # Enable Auto Remote Restart
vm1_storage_name   = "xiv_StoragePool"      # Storage Template
vm1_dockerdisk1    = "512"                  # Docker disk size in GB for ephemeral storage

#VM2 configuration (OCP - Infra Nodes)
#---------------------------------
vm2_number         = "0"                    # Number of VMs
vm2_memory         = "16"                   # Memory GB
vm2_cpu            = "2"                    # Virtual CPU
vm2_vcpu_ratio     = "4"                    # vCPU RATIO 1:4 1 vCPU = 0.25 eCPU (cores)
vm2_name           = "infnode"              # Hostname prefix
```

```
vm2_first_ip       = "192.168.11.205"       # Fist IP from a consecutive pool of IPs
vm2_image_name     = "xiv_p9_image_rhel76"  # The image name
vm2_remote_restart = "true"                 # Enable Auto Remote Restart
vm2_storage_name   = "xiv_StoragePool"      # Storage Template
vm2_dockerdisk1    = "64"                   # Docker disk size in GB for ephemeral storage

#VM3 configuration (OCP - Workers(App) Nodes)
#---------------------------------
vm3_number         = "3"                    # Number of VMs
vm3_memory         = "16"                   # Memory GB
vm3_cpu            = "4"                    # Virtual CPU
vm3_vcpu_ratio     = "2"                    # vCPU RATIO 1:4 1 vCPU = 0.5 eCPU (cores)
vm3_name           = "wrknode"              # Hostname prefix
vm3_first_ip       = "192.168.11.208"       # Fist IP from a consecutive pool of IPs
vm3_image_name     = "xiv_p9_image_rhel76"  # The image name
vm3_remote_restart = "false"                # Disable Auto Remote Restart
vm3_storage_name   = "xiv_StoragePool"      # Storage Template
vm3_dockerdisk1    = "32"                   # Docker disk size in GB for ephemeral storage

#VM4 configuration (OCP - Load Balancer Node)
#---------------------------------
vm4_number         = "1"                    # Number of VMs
vm4_memory         = "8"                    # Memory GB
vm4_cpu            = "2"                    # Virtual CPU
vm4_vcpu_ratio     = "4"                    # vCPU RATIO 1:4 1 vCPU = 0.25 eCPU (cores)
vm4_name           = "lbsnode"              # Hostname prefix
vm4_first_ip       = "192.168.11.212"       # Fist IP from a consecutive pool of IPs
vm4_image_name     = "xiv_p9_image_rhel76"  # The image name
vm4_remote_restart = "true"                 # Enable Auto Remote Restart
```

Example 6-6 terraform.tfvars for three nodes deployment

```
#PowerVC (OpenStack)
#---------------------------------
powervc_user        = "ocpadmin"            # PowerVC user
powervc_password    = "<oassword>"          # PowerVC password
powervc_server      = "192.168.11.31"       # PowerVC IP or hostname
powervc_project     = "ocp-project"         # PowerVC project(tenant) name


#General configuration:
#---------------------------------
ssh_user            = "root"                # Image username
ssh_user_password   = "<password>"          # Image password
user_public_key     = "ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEAO9+YMqJ8VHX3HC7qy6HSxs3JjTGKbEgK+CExpf811uxsq+uJYbfXEKH19/NCf/U
vpkozJBDDXDIxJ4uqOEBWDG4mUuu5U9a4lXgb6qaPYyXwVTygL/IcBOpoSGEQQaJzhBO5g71uZrya++sG1xHUjSQAQz
hDuKrs4Bc3gcN4184UR+BX1pVgCls3NRn9hLrfLWS37M/kn+b/n6VMYYVpHsZ2XVydAn2nwuzktaEuWYaY/1cNd4xuu
yVuO8GQOon6t5KQ1EZBheADdSsyamulLqW9z4j6Y1wwDe4GPDc5zIW++ASDAZBOeEfbKGDLVdpFsI5YV8nLV1r/TOY/
FiFZqQ== Bogdan Savu;IBMRO045771;IBMROZZ014E826;J;"
dns1                = "192.168.11.210"      # DNS server 1
dns_domain          = "domain.example.com"  # DNS Domain Name

#Network configuration
#---------------------------------
net1_name           = "net_ocp_cluster2"    # Network Name
net1_vlan_id        = "1"                   # VLAN ID
net1_subnet         = "192.168.11.0/21"     # Network/Mask
net1_gateway        = "192.168.11.1"        # Gateway
net1_start          = "192.168.11.202"      # First IP from Pool
```

```
net1_end            = "192.168.11.212"      # Last IP from Pool

#VM1 configuration (OCP - Master Nodes)
#----------------------------------
vm1_number         = "1"                    # Number of VMs
vm1_memory         = "32"                   # Memory GB
vm1_cpu            = "8"                    # Virtual CPU
vm1_vcpu_ratio     = "2"                    # vCPU RATIO 1:2 1 vCPU = 0.5 eCPU (cores)
vm1_name           = "mstnode"              # Hostname prefix
vm1_first_ip       = "192.168.11.202"       # Fist IP from a consecutive pool of IPs
vm1_image_name     = "xiv_p9_image_rhel76"  # The image name
vm1_remote_restart = "true"                 # Enable Auto Remote Restart
vm1_storage_name   = "xiv_StoragePool"      # Storage Template
vm1_dockerdisk1    = "256"                  # Docker disk size in GB for ephemeral storage

#VM2 configuration (OCP - Infra Nodes)
#----------------------------------
vm2_number         = "0"                    # Number of VMs
vm2_memory         = "16"                   # Memory GB
vm2_cpu            = "2"                    # Virtual CPU
vm2_vcpu_ratio     = "4"                    # vCPU RATIO 1:4 1 vCPU = 0.25 eCPU (cores)
vm2_name           = "infnode"              # Hostname prefix
vm2_first_ip       = "192.168.11.205"       # Fist IP from a consecutive pool of IPs
vm2_image_name     = "xiv_p9_image_rhel76"  # The image name
vm2_remote_restart = "true"                 # Enable Auto Remote Restart
vm2_storage_name   = "xiv_StoragePool"      # Storage Template
vm2_dockerdisk1    = "64"                   # Docker disk size in GB for ephemeral storage

#VM3 configuration (OCP - Workers(App) Nodes)
#----------------------------------
vm3_number         = "2"                    # Number of VMs
vm3_memory         = "64"                   # Memory GB
vm3_cpu            = "8"                    # Virtual CPU
vm3_vcpu_ratio     = "2"                    # vCPU RATIO 1:2 1 vCPU = 0.5 eCPU (cores)
vm3_name           = "wrknode"              # Hostname prefix
vm3_first_ip       = "192.168.11.208"       # Fist IP from a consecutive pool of IPs
vm3_image_name     = "xiv_p9_image_rhel76"  # The image name
vm3_remote_restart = "false"                # Disable Auto Remote Restart
vm3_storage_name   = "xiv_StoragePool"      # Storage Template
vm3_dockerdisk1    = "128"                  # Docker disk size in GB for ephemeral storage

#VM4 configuration (OCP - Load Balancer Node)
#----------------------------------
vm4_number         = "0"                    # Number of VMs
vm4_memory         = "8"                    # Memory GB
vm4_cpu            = "2"                    # Virtual CPU
vm4_vcpu_ratio     = "4"                    # vCPU RATIO 1:4 1 vCPU = 0.25 eCPU (cores)
vm4_name           = "lbsnode"              # Hostname prefix
vm4_first_ip       = "192.168.11.212"       # Fist IP from a consecutive pool of IPs
vm4_image_name     = "xiv_p9_image_rhel76"  # The image name
vm4_remote_restart = "true"                 # Enable Auto Remote Restart
```

Attention: To use an existing network, you must remove the `network.tf` file before applying the configuration.

Master-Infrastructure and Worker Nodes have a disk DOCKER_DISK_1 for `docker-vg`.

2. Initialize the Terraform working directory:

```
cd <PATH> #ocp-aio /ocp-7nodes ocp-3nodes
terraform init
Initializing the backend...
Initializing provider plugins...
The following providers do not have any version constraints in configuration,
so the latest version was installed.
...
Output truncated
...
* provider.null: version = "~> 2.1"
* provider.openstack: version = "~> 1.22"
Terraform has been successfully initialized!
You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

3. Create an execution plan:

```
terraform plan
Refreshing Terraform state in-memory prior to plan...
...
Output truncated
...
Plan: 33 to add, 0 to change, 0 to destroy.
------------------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save this plan, so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

4. Apply the required changes to reach the wanted state:

```
terraform apply
data.openstack_compute_availability_zones_v2.AZ: Refreshing state...
...
Output truncated
...
Plan: 33 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
...
Output truncated
...
null_resource.vm4_post_install_config[0] (remote-exec): ---start adding
user_public_key---
Apply complete! Resources: 33 added, 0 changed, 0 destroyed.
```

After this process is completed, the virtual machines are up and running.

6.3.4 Installing the OpenShift Container Platform

Complete the following steps to install OpenShift Container Platform:

1. Create the Ansible inventory file. You can use the `aio.inv.example` for All-In-One (AIO) configuration (see Example 6-7), `7nodes.inv.example` for seven nodes configuration (see Example 6-8 on page 122), or `3nodes.inv.example` for three nodes configuration (see Example 6-9 on page 125). Many other configurations are possible that can be used but are not covered in this book.

Example 6-7 Ansible all in one inventory example

```
##------------------------------------------------------------------------##
## All-in-One(AIO):
##------------------------------------------------------------------------##

[OSEv3:vars]
##------------------------------------------------------------------------##
## Ansible Vars
##------------------------------------------------------------------------##
timeout=60
# ansible_user={{CHANGEME_ANSIBLE_SSH_USER}}
ansible_user=root

##------------------------------------------------------------------------##
## OpenShift Basic Vars
##------------------------------------------------------------------------##
# Deployment type
openshift_deployment_type=openshift-enterprise
# WARNING: only disable these checks in LAB/TEST environments
# openshift_disable_check="disk_availability,memory_availability"
# OpenShift Version:
openshift_release=3.11
openshift_pkg_version=-3.11.154
openshift_image_tag=v3.11.154
# firewalld recommended for new installations (default is iptables)
os_firewall_use_firewalld=true
# enable ntp on masters to ensure proper failover
openshift_clock_enabled=true
debug_level=5

##------------------------------------------------------------------------##
## OpenShift Registries Locations
##------------------------------------------------------------------------##
# NOTE: Need credentials from: https://access.redhat.com/terms-based-registry/
oreg_url=registry.redhat.io/openshift3/ose-${component}:${version}
oreg_auth_user={{CHANGEME_REGISTRY_SERVICE_ACCOUNT}}
oreg_auth_password={{CHANGEME_SERVICE_KEY}}

##------------------------------------------------------------------------##
## OpenShift Master Vars
##------------------------------------------------------------------------##
openshift_master_api_port=8443
openshift_master_console_port=8443
# Internal cluster name
openshift_master_cluster_hostname=bsocp01.domain.example.com
# External cluster name
# openshift_master_cluster_public_hostname=ocp-ext.example.com
# Default wildcard domain for applications
openshift_master_default_subdomain=apps.bs.ibm.com
```

```
##------------------------------------------------------------------------##
## OpenShift Authentication Vars
##------------------------------------------------------------------------##
# Available Identity Providers
#
https://docs.openshift.com/container-platform/3.11/install_config/configuring_authenticatio
n.html
##---------------------##
# htpasswd Authentication
##---------------------##
# NOTE: read initial identities in htpasswd format from /root/htpasswd.openshift
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]
# To define initial users directly in the inventory file:
# Note:
https://docs.openshift.com/container-platform/3.3/admin_solutions/master_node_config.html#h
tpasswd
openshift_master_htpasswd_users={'admin':'$apr1$hYehsOQ6$DQWSmGhPdS2LzS5cDJuU21','developer
':'$apr1$IOa9K2vO$ZLPrXnQseMlwTJIYzM8Hd.'}
# To use external htpassword file:
# openshift_master_htpasswd_file=/root/htpasswd.openshift

##------------------------------------------------------------------------##
## Docker Vars
##------------------------------------------------------------------------##
# container_runtime_docker_storage_setup_device=/dev/mapper/DOCKER_DISK_1
# container_runtime_docker_storage_type=overlay2

##------------------------------------------------------------------------##
## OpenShift Router and Registry Vars
##------------------------------------------------------------------------##
# NOTE: Qty should NOT exceed the number of infra nodes
openshift_hosted_router_replicas=1
openshift_hosted_registry_replicas=1
openshift_router_selector='node-role.kubernetes.io/infra=true'
openshift_registry_selector='node-role.kubernetes.io/infra=true'

##------------------------------------------------------------------------##
## OpenShift Network Vars
##------------------------------------------------------------------------##
# Defaults
# osm_cluster_network_cidr=10.1.0.0/16
# openshift_portal_net=172.30.0.0/16
# Configure the multi-tenant SDN plugin (default is 'redhat/openshift-ovs-subnet')
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'

##------------------------------------------------------------------------##
## OpenShift Cockpit
##------------------------------------------------------------------------##
# Disable cockpit
osm_use_cockpit=false

##------------------------------------------------------------------------##
## Metrics Server
##------------------------------------------------------------------------##
# Enable Metrics Server - is requiered for IBM Cloud Pak
openshift_metrics_server_install=true

##------------------------------------------------------------------------##
```

```
## Service Catalog
##------------------------------------------------------------------------##
openshift_enable_service_catalog=true
ansible_service_broker_install=true
template_service_broker_install=true
openshift_template_service_broker_namespaces=['openshift']
ansible_service_broker_local_registry_whitelist=['.*-apb$']
template_service_broker_selector={"node-role.kubernetes.io/infra":"true"}

##------------------------------------------------------------------------##
## OpenShift Hosts
##------------------------------------------------------------------------##
[OSEv3:children]
nfs
masters
etcd
nodes

[nfs]
bsocp01.domain.example.com

[masters]
bsocp01.domain.example.com

[etcd]
bsocp01.domain.example.com

[nodes]
## All-In-One with Docker
bsocp01.domain.example.com openshift_node_group_name='node-config-all-in-one'
openshift_node_problem_detector_install=true
```

Example 6-8 Ansible 7nodes inventory example

```
##------------------------------------------------------------------------##
## 7Nodes (3xMaster-Infra, 3xWorkers(Applications) and 1 Load Balancer):
##------------------------------------------------------------------------##

[OSEv3:vars]
##------------------------------------------------------------------------##
## Ansible Vars
##------------------------------------------------------------------------##
timeout=60
# ansible_user={{CHANGEME_ANSIBLE_SSH_USER}}
# ansible_become=yes
ansible_user=root

##------------------------------------------------------------------------##
## OpenShift Basic Vars
##------------------------------------------------------------------------##
# Deployment type
openshift_deployment_type=openshift-enterprise
# WARNING: only disable these checks in LAB/TEST environments
# openshift_disable_check="disk_availability,memory_availability"
# OpenShift Version:
openshift_release=3.11
openshift_image_tag=v3.11.154
openshift_pkg_version=-3.11.154
# firewalld recommended for new installations (default is iptables)
os_firewall_use_firewalld=True
```

```
# enable ntp on masters to ensure proper failover
openshift_clock_enabled=True

##------------------------------------------------------------------------##
## OpenShift Registries Locations
##------------------------------------------------------------------------##
# NOTE: Need credentials from: https://access.redhat.com/terms-based-registry/
oreg_url=registry.redhat.io/openshift3/ose-${component}:${version}
oreg_auth_user={{CHANGEME_REGISTRY_SERVICE_ACCOUNT}}
oreg_auth_password={{CHANGEME_SERVICE_KEY}}

##------------------------------------------------------------------------##
## OpenShift Master Vars
##------------------------------------------------------------------------##
openshift_master_api_port=8443
openshift_master_console_port=8443
# Internal cluster name
openshift_master_cluster_hostname=ocp.domain.example.com
# Note: When using an external Load Balancer service or device, the FQDN
# of the northbound VIP address must be specified in the inventory file
# using the variable openshift_master_cluster_public_hostname
openshift_master_cluster_public_hostname=ocp.domain.example.com
# Note: The OpenShift Routers at Infrastructure Nodes require a wildcard
# subdomain it will use to dynamically build a URL or Route for applications
# running on the platform and exposing a service outside the cluster
openshift_master_default_subdomain=apps.domain.example.com

##------------------------------------------------------------------------##
## OpenShift Router and Registry Vars
##------------------------------------------------------------------------##
# NOTE: Qty should NOT exceed the number of infra nodes
openshift_hosted_router_replicas=3
openshift_hosted_router_selector='node-role.kubernetes.io/infra=true'
# NOTE: PowerVC FlexVolume Driver doesn't support shared file systems
openshift_hosted_registry_replicas=1
openshift_hosted_registry_selector='node-role.kubernetes.io/infra=true'

##------------------------------------------------------------------------##
## OpenShift Authentication Vars
##------------------------------------------------------------------------##
# Available Identity Providers
#
https://docs.openshift.com/container-platform/3.11/install_config/configuring_authenticatio
n.html
##----------------------##
# htpasswd Authentication
##----------------------##
# NOTE: read initial identities in htpasswd format from /root/htpasswd.openshift
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'True',
'challenge': 'True', 'kind': 'HTPasswdPasswordIdentityProvider'}]
# To define initial users directly in the inventory file:
# Note:
https://docs.openshift.com/container-platform/3.3/admin_solutions/master_node_config.html#h
tpasswd
openshift_master_htpasswd_users={'admin':'$apr1$hYehsOQ6$DQWSmGhPdS2LzS5cDJuU21','developer
':'$apr1$IOa9K2vO$ZLPrXnQseMlwTJIYzM8Hd.'}

##------------------------------------------------------------------------##
## Docker Vars
##------------------------------------------------------------------------##
```

```
container_runtime_docker_storage_setup_device=/dev/mapper/DOCKER_DISK_1
container_runtime_docker_storage_type=overlay2

##-------------------------------------------------------------------------##
## OpenShift Network Vars
##-------------------------------------------------------------------------##
# Defaults
# osm_cluster_network_cidr=10.128.0.0/14
# openshift_portal_net=172.30.0.0/16
# Configure the multi-tenant SDN plugin (default is 'redhat/openshift-ovs-subnet')
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
# Set SDN MTU (default is 1450)
# openshift_node_sdn_mtu=1400

##-------------------------------------------------------------------------##
## Metrics Server
##-------------------------------------------------------------------------##
# Enable Metrics Server
openshift_metrics_server_install=True

##-------------------------------------------------------------------------##
## Service Catalog
##-------------------------------------------------------------------------##
# default=True
openshift_enable_service_catalog=True
# default=True
template_service_broker_install=True
openshift_template_service_broker_namespaces=['openshift']
template_service_broker_selector={"node-role.kubernetes.io/infra":"true"}
# default=True
ansible_service_broker_install=True
ansible_service_broker_local_registry_whitelist=['.*-apb$']

##-------------------------------------------------------------------------##
## Prometheus Cluster Monitoring
##-------------------------------------------------------------------------##
#
https://docs.openshift.com/container-platform/3.11/install_config/prometheus_cluster_monito
ring.html
openshift_cluster_monitoring_operator_install=False
openshift_prometheus_node_selector={"node-role.kubernetes.io/infra":"true"}
# Enable persistent storage of Prometheus time-series data (default False)
openshift_cluster_monitoring_operator_prometheus_storage_enabled=True
# Enable persistent storage of Alertmanager notifications (default False)
openshift_cluster_monitoring_operator_alertmanager_storage_enabled=True
# Dynamic storage allocation for Prometheus services
openshift_cluster_monitoring_operator_prometheus_storage_capacity=32Gi
openshift_cluster_monitoring_operator_alertmanager_storage_capacity=4Gi
# Storage class to use if persistent storage enabled
# NOTE: it will use storageclass default if storage class not specified
openshift_cluster_monitoring_operator_prometheus_storage_class_name='ibm-powervc-k8s-volume
-default'
openshift_cluster_monitoring_operator_alertmanager_storage_class_name='ibm-powervc-k8s-volu
me-default'

##-------------------------------------------------------------------------##
## OpenShift Hosts
##-------------------------------------------------------------------------##
[OSEv3:children]
masters
```

```
etcd
nodes
lb

# host group for masters
[masters]
mstnode0[1:3].domain.example.com

# host group for etcd
[etcd]
mstnode0[1:3].domain.example.com

# Specify load balancer host
[lb]
lbsnode01.domain.example.com

[nodes]
mstnode0[1:3].domain.example.com openshift_node_group_name='node-config-master-infra'
wrknode0[1:3].domain.example.com openshift_node_group_name='node-config-compute'
```

Example 6-9 Ansible 3nodes inventory example

```
##-------------------------------------------------------------------------##
## 3Nodes (1xMaster-Infra, 3xWorkers(Applications):
##-------------------------------------------------------------------------##
[OSEv3:vars]
##-------------------------------------------------------------------------##
## Ansible Vars
##-------------------------------------------------------------------------##
timeout=60
# ansible_user={{CHANGEME_ANSIBLE_SSH_USER}}
# ansible_become=yes
ansible_user=root

##-------------------------------------------------------------------------##
## OpenShift Basic Vars
##-------------------------------------------------------------------------##
# Deployment type
openshift_deployment_type=openshift-enterprise
# WARNING: only disable these checks in LAB/TEST environments
# openshift_disable_check="disk_availability,memory_availability"
# OpenShift Version:
openshift_release=3.11
openshift_image_tag=v3.11.154
openshift_pkg_version=-3.11.154
# firewalld recommended for new installations (default is iptables)
os_firewall_use_firewalld=True
# enable ntp on masters to ensure proper failover
openshift_clock_enabled=True

##-------------------------------------------------------------------------##
## OpenShift Registries Locations
##-------------------------------------------------------------------------##
# NOTE: Need credentials from: https://access.redhat.com/terms-based-registry/
oreg_url=registry.redhat.io/openshift3/ose-${component}:${version}
oreg_auth_user={{CHANGEME_REGISTRY_SERVICE_ACCOUNT}}
oreg_auth_password={{CHANGEME_SERVICE_KEY}}

##-------------------------------------------------------------------------##
## OpenShift Master Vars
```

```
##-----------------------------------------------------------------------##
openshift_master_api_port=8443
openshift_master_console_port=8443
# Internal cluster name
openshift_master_cluster_hostname=ocp.domain.example.com
# Note: When using an external Load Balancer service or device, the FQDN
# of the northbound VIP address must be specified in the inventory file
# using the variable openshift_master_cluster_public_hostname
openshift_master_cluster_public_hostname=ocp.domain.example.com
# Note: The OpenShift Routers at Infrastructure Nodes require a wildcard
# subdomain it will use to dynamically build a URL or Route for applications
# running on the platform and exposing a service outside the cluster
openshift_master_default_subdomain=apps.domain.example.com

##-----------------------------------------------------------------------##
## OpenShift Router and Registry Vars
##-----------------------------------------------------------------------##
# NOTE: Qty should NOT exceed the number of infra nodes
openshift_hosted_router_replicas=1
openshift_hosted_router_selector='node-role.kubernetes.io/infra=true'
# NOTE: PowerVC FlexVolume Driver doesn't support shared file systems
openshift_hosted_registry_replicas=1
openshift_hosted_registry_selector='node-role.kubernetes.io/infra=true'

##-----------------------------------------------------------------------##
## OpenShift Authentication Vars
##-----------------------------------------------------------------------##
# Available Identity Providers
#
https://docs.openshift.com/container-platform/3.11/install_config/configuring_authenticatio
n.html
##---------------------##
# htpasswd Authentication
##---------------------##
# NOTE: read initial identities in htpasswd format from /root/htpasswd.openshift
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'True',
'challenge': 'True', 'kind': 'HTPasswdPasswordIdentityProvider'}]
# To define initial users directly in the inventory file:
# Note:
https://docs.openshift.com/container-platform/3.3/admin_solutions/master_node_config.html#h
tpasswd
openshift_master_htpasswd_users={'admin':'$apr1$hYehsOQ6$DQWSmGhPdS2LzS5cDJuU21','developer
':'$apr1$IOa9K2vO$ZLPrXnQseMlwTJIYzM8Hd.'}

##-----------------------------------------------------------------------##
## Docker Vars
##-----------------------------------------------------------------------##
container_runtime_docker_storage_setup_device=/dev/mapper/DOCKER_DISK_1
container_runtime_docker_storage_type=overlay2

##-----------------------------------------------------------------------##
## OpenShift Network Vars
##-----------------------------------------------------------------------##
# Defaults
# osm_cluster_network_cidr=10.128.0.0/14
# openshift_portal_net=172.30.0.0/16
# SDN plugin (default is 'redhat/openshift-ovs-subnet')
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'
# Set SDN MTU (default is 1450)
# openshift_node_sdn_mtu=1400
```

```
##-------------------------------------------------------------------------##
## Metrics Server
##-------------------------------------------------------------------------##
# Enable Metrics Server
openshift_metrics_server_install=True

##-------------------------------------------------------------------------##
## Service Catalog
##-------------------------------------------------------------------------##
# default=True
openshift_enable_service_catalog=True
# default=True
template_service_broker_install=True
openshift_template_service_broker_namespaces=['openshift']
template_service_broker_selector={"node-role.kubernetes.io/infra":"true"}
# default=True
ansible_service_broker_install=True
ansible_service_broker_local_registry_whitelist=['.*-apb$']

##-------------------------------------------------------------------------##
## Prometheus Cluster Monitoring
##-------------------------------------------------------------------------##
#
https://docs.openshift.com/container-platform/3.11/install_config/prometheus_cluster_monito
ring.html
openshift_cluster_monitoring_operator_install=False
openshift_prometheus_node_selector={"node-role.kubernetes.io/infra":"true"}
# Enable persistent storage of Prometheus time-series data (default False)
openshift_cluster_monitoring_operator_prometheus_storage_enabled=True
# Enable persistent storage of Alertmanager notifications (default False)
openshift_cluster_monitoring_operator_alertmanager_storage_enabled=True
# Dynamic storage allocation for Prometheus services
openshift_cluster_monitoring_operator_prometheus_storage_capacity=32Gi
openshift_cluster_monitoring_operator_alertmanager_storage_capacity=4Gi
# Storage class to use if persistent storage enabled
# NOTE: it will use storageclass default if storage class not specified
openshift_cluster_monitoring_operator_prometheus_storage_class_name='ibm-powervc-k8s-volume
-default'
openshift_cluster_monitoring_operator_alertmanager_storage_class_name='ibm-powervc-k8s-volu
me-default'

##-------------------------------------------------------------------------##
## OpenShift Hosts
##-------------------------------------------------------------------------##
[OSEv3:children]
masters
etcd
nodes

# host group for masters
[masters]
mstnode01.domain.example.com

# host group for etcd
[etcd]
mstnode01.domain.example.com

[nodes]
mstnode01.domain.example.com openshift_node_group_name='node-config-master-infra'
```

```
wrknode0[1:3].domain.example.com openshift_node_group_name='node-config-compute'
```

2. Check whether the VMs are available:

```
ansible -i <inventory_file> nodes -m ping
mstnode02.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
mstnode03.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
wkrnode02.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
wkrnode03.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
mstnode01.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
wkrnode01.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
lbsnode01.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}s
```

3. Register the VMs to RHSM:

```
ansible -i <inventory_file> -a 'subscription-manager register
--username={{REGISTRY_SERVICE_ACCOUNT}}}} --password={{SERVICE_KEY}}'
wkrnode01.domain.example.com | SUCCESS | rc=0 >>
Registering to: subscription.rhsm.redhat.com:443/subscription
The system has been registered with ID: 5aee976f-1345-4e25-8c27-0028b40b6cac
...
Output truncated
...
mstnode03.domain.example.com | SUCCESS | rc=0 >>
Registering to: subscription.rhsm.redhat.com:443/subscription
The system has been registered with ID: 952c0b7d-bd50-4ed4-99a9-47ef762ef29f
The registered system name is: mstnode03.domain.example.com
```

4. Attach to the Red Hat subscription Pool:

```
 ansible -i  <inventory_file> nodes -a 'subscription-manager attach
--pool={{POOL_ID}}'
...
Output truncated
...
wkrnode03.domain.example.com | SUCCESS | rc=0 >>
Successfully attached a subscription for: Red Hat OpenShift Container Platform for
Power, LE Business Partner NFR, Self-Supported
```

5. Enable Red Hat repositories for IBM POWER9:

```
 ansible -i  <inventory_file> nodes -a 'subscription-manager repos --disable="*"
--enable="rhel-7-for-power-9-rpms" --enable="rhel-7-for-power-9-extras-rpms"
--enable="rhel-7-for-power-9-optional-rpms"
--enable="rhel-7-server-ansible-2.6-for-power-9-rpms"
--enable="rhel-7-server-for-power-9-rhscl-rpms"
--enable="rhel-7-for-power-9-ose-3.11-rpms"
wkrnode01.domain.example.com | SUCCESS | rc=0 >>
Repository 'rhel-7-for-power-9-optional-source-rpms' is disabled for this system.
Repository 'rhel-7-for-power9-fast-datapath-beta-debug-rpms' is disabled for this
system.
Repository 'rhel-7-for-power-9-supplementary-debug-rpms' is disabled for this system.
Repository 'rhel-7-for-power-9-satellite-tools-6.3-debug-rpms' is disabled for this
system.
...
Output truncated
...
lbsnode01.domain.example.com | SUCCESS | rc=0 >>
Repository 'rhel-7-for-power-9-optional-source-rpms' is disabled for this system.
Repository 'rhel-7-for-power9-fast-datapath-beta-debug-rpms' is disabled for this
system.
...
Output truncated
...
Repository 'rhel-7-for-power-9-supplementary-rpms' is disabled for this system.
Repository 'rhel-7-server-ansible-2-for-power-9-rpms' is disabled for this system.
Repository 'rhel-7-for-power-9-optional-beta-debug-rpms' is disabled for this system.
Repository 'rhel-7-for-power-9-supplementary-beta-rpms' is disabled for this system.
```

6. Install the latest operating system updates:

```
ansible -i <inventory_file> nodes,lb -a 'yum -y update --security
--exclude=cloud-init\*'
 [WARNING]: Consider using the yum module rather than running yum.  If you need to use
command because yum is insufficient you can add warn=False to this
command task or set command_warnings=False in ansible.cfg to get rid of this message.

mstnode02.domain.example.com | SUCCESS | rc=0 >>
Loaded plugins: product-id, search-disabled-repos, subscription-manager
5 package(s) needed (+0 related) for security, out of 5 available
Resolving Dependencies
--> Running transaction check
---> Package kernel.ppc64le 0:4.14.0-115.14.1.el7a will be installed
...
Output truncated
...
Installed:
  kernel.ppc64le 0:4.14.0-115.14.1.el7a

Updated:
  kernel-bootwrapper.ppc64le 0:4.14.0-115.14.1.el7a
  kernel-tools.ppc64le 0:4.14.0-115.14.1.el7a
  kernel-tools-libs.ppc64le 0:4.14.0-115.14.1.el7a
  python-perf.ppc64le 0:4.14.0-115.14.1.el7a

Complete!
```

7. Update kernel parameters:

```
ansible -i <inventory_file> nodes,lb -m sysctl -a "name=vm.max_map_count
value=262144 state=present sysctl_set=yes reload=yes"
ansible -i <inventory_file> nodes,lb -m sysctl -a "name=net.ipv4.ip_forward
value=1 state=present sysctl_set=yes reload=yes"
ansible -i <inventory_file> nodes,lb -m sysctl -a
"name=fs.inotify.max_user_watches value=65536 state=present sysctl_set=yes
reload=yes"
mstnode02.domain.example.com | SUCCESS => {
    "changed": true
}
...
Output truncated
...
lbsnode01.domain.example.com | SUCCESS => {
    "changed": true
}
wrknode01.domain.example.com | SUCCESS => {
    "changed": true
}
ansible -i <inventory_file> lb -m sysctl -a "name=net.ipv4.ip_nonlocal_bind
value=1 state=present sysctl_set=yes reload=yes"
lbsnode01.domain.example.com | SUCCESS => {
    "changed": true
}
```

8. Update `firewall` to allow HTTP and HTTPS to access the Load Balancer VM:

```
ansible -i <inventory_file> lb -a 'firewall-cmd --permanent --add-service=http'
ansible -i <inventory_file> lb -a 'firewall-cmd --permanent
--add-service=https'
ansible -i <inventory_file> lb -a 'firewall-cmd --reload'
lbsnode01.domain.example.com | SUCCESS | rc=0 >>
success
```

9. Update the DNS options:

```
ansible -i 7nodes.inv nodes,lb -a 'nmcli connection modify "System eth0"
ipv4.dns-options "rotate timeout:1 ndots:5"'
ansible -i 7nodes.inv nodes,lb -a 'nmcli con show "System eth0"' | grep -e
ipv4.dns-search -e ipv4.dns-options
ipv4.dns-search:                        ssm.sdc.gts.ibm.com
ipv4.dns-options:                       "rotate,timeout:1,ndots:5"
ipv4.dns-search:                        ssm.sdc.gts.ibm.com
ipv4.dns-options:                       "rotate,timeout:1,ndots:5"
ipv4.dns-search:                        ssm.sdc.gts.ibm.com
ipv4.dns-options:                       "rotate,timeout:1,ndots:5"
ipv4.dns-search:                        ssm.sdc.gts.ibm.com
ipv4.dns-options:                       "rotate,timeout:1,ndots:5"
ipv4.dns-search:                        ssm.sdc.gts.ibm.com
ipv4.dns-options:                       "rotate,timeout:1,ndots:5"
ipv4.dns-search:                        ssm.sdc.gts.ibm.com
ipv4.dns-options:                       "rotate,timeout:1,ndots:5"
ipv4.dns-search:                        ssm.sdc.gts.ibm.com
ipv4.dns-options:                       "rotate,timeout:1,ndots:5"
```

10. Reboot the VMs:

```
ansible -i <inventory_file> nodes,lb -a 'reboot'
wrknode02.domain.example.com | UNREACHABLE! => {
    "changed": false,
    "msg": "SSH Error: data could not be sent to remote host
\"wrknode02.domain.example.com\". Make sure this host can be reached over ssh",
    "unreachable": true
}

mstnode01.domain.example.com | UNREACHABLE! => {
    "changed": false,
    "msg": "SSH Error: data could not be sent to remote host
\"mstnode01.domain.example.com\". Make sure this host can be reached over ssh",
    "unreachable": true
}
...
Output truncated
...
lbsnode01.domain.example.com | UNREACHABLE! => {
    "changed": false,
    "msg": "SSH Error: data could not be sent to remote host
\"lbsnode01.domain.example.com\". Make sure this host can be reached over ssh",
    "unreachable": true
}
```

11.Check whether the VMs are started:

```
ansible -i <inventory_file> nodes,lb -m ping
mstnode01.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
wrknode01.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
wrknode02.domain.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

12.Uninstall the old kernels from VMs:

```
ansible -i <inventory_file> nodes,lb -a 'package-cleanup -y --oldkernels
--count=1'
mstnode01.domain.example.com | SUCCESS | rc=0 >>
Loaded plugins: product-id, subscription-manager
--> Running transaction check
---> Package kernel.ppc64le 0:4.14.0-115.13.1.el7a will be erased
...
Output truncated
...
Removed:
  kernel.ppc64le 0:4.14.0-115.13.1.el7a
Complete!
```

13.Prepare the VMs for OpenShift installation:

```
ansible-playbook -i <inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/prerequisites.yml
PLAY [Fail openshift_kubelet_name_override for new hosts]
*****************************************************************************************

TASK [Gathering Facts]
*****************************************************************************************
Thursday 28 November 2019  20:00:34 +0000 (0:00:00.103)       0:00:00.103 *****
ok: [wrknode01.domain.example.com]
ok: [mstnode01.domain.example.com]
ok: [wrknode02.domain.example.com]

TASK [Fail when openshift_kubelet_name_override is defined]
*****************************************************************************************
Thursday 28 November 2019  20:00:37 +0000 (0:00:02.502)       0:00:02.605 *****
skipping: [mstnode01.domain.example.com]
skipping: [wrknode01.domain.example.com]
skipping: [wrknode02.domain.example.com]

PLAY [Initialization Checkpoint Start]
*****************************************************************************************

TASK [Set install initialization 'In Progress']
*****************************************************************************************
Thursday 28 November 2019  20:00:37 +0000 (0:00:00.268)       0:00:02.873 *****
ok: [mstnode01.domain.example.com]

PLAY [Populate config host groups]
*****************************************************************************************
...
Output truncated
...
INSTALLER STATUS
**********************************************************************************
Initialization  : Complete (0:04:13)
Wednesday 30 October 2019  19:53:17 +0000 (0:00:00.460)       0:10:03.419 *******
==================================================================================
container_runtime : Install Docker -------------------------------------------- 208.24s
Ensure openshift-ansible installer package deps are installed ------------------ 175.65s
openshift_excluder : Install docker excluder - yum ----------------------------- 43.74s
openshift_repos : refresh cache ------------------------------------------------ 19.29s
container_runtime : Start the Docker service ----------------------------------- 10.05s
container_runtime : Create credentials for oreg_url ------------------------------ 8.60s
openshift_repos : Ensure libselinux-python is installed -------------------------- 7.91s
container_runtime : Fixup SELinux permissions for docker ------------------------- 7.80s
os_firewall : Install firewalld packages ----------------------------------------- 6.54s
Gathering Facts ------------------------------------------------------------------ 5.48s
container_runtime : Get current installed Docker version ------------------------- 2.92s
Gathering Facts ------------------------------------------------------------------ 2.26s
Start and enable ntpd/chronyd ---------------------------------------------------- 1.65s
os_firewall : Ensure iptables services are not enabled --------------------------- 1.59s
Check for NetworkManager service ------------------------------------------------- 1.58s
container_runtime : Setup the docker-storage for overlay ------------------------- 1.48s
openshift_sanitize_inventory : include_tasks ------------------------------------- 1.19s
openshift_sanitize_inventory : include_tasks ------------------------------------- 1.17s
os_firewall : Install iptables packages ------------------------------------------ 1.15s
container_runtime : include_tasks ------------------------------------------------ 1.08s
```

14. Install the OpenShift Container Platform:

```
ansible-playbook -i <inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/deploy_cluster.yml
PLAY [Initialization Checkpoint Start]
*************************************************************************************
...
Output truncated
...
Hosted Install               : Complete (0:00:34)
Cluster Monitoring Operator  : Complete (0:01:44)
Web Console Install          : Complete (0:00:32)
Console Install              : Complete (0:01:22)
metrics-server Install       : Complete (0:00:01)
Service Catalog Install      : Complete (0:04:46)
Wednesday 30 October 2019  20:32:17 +0000 (0:00:00.035)       0:34:41.168 *****
===============================================================================
openshift_node : install needed rpm(s) ----------------------------------------- 177.38s
openshift_node : Install node, clients, and conntrack packages ----------------- 126.12s
template_service_broker : Verify that TSB is running --------------------------- 110.11s
openshift_cluster_monitoring_operator : Wait for the ServiceMonitor CRD to be created
--------------------------------------------------------------------------------- 91.75s
openshift_console : Waiting for console rollout to complete --------------------- 72.95s
openshift_control_plane : Wait for all control plane pods to become ready ------- 67.19s
Run health checks (install) - EL ------------------------------------------------ 63.83s
openshift_node : Install Ceph storage plugin dependencies ----------------------- 59.41s
openshift_service_catalog : Wait for API Server rollout success ----------------- 56.42s
openshift_control_plane : Wait for control plane pods to appear ----------------- 54.33s
openshift_ca : Install the base package for admin tooling ----------------------- 53.89s
openshift_excluder : Install openshift excluder - yum --------------------------- 44.92s
openshift_service_catalog : Wait for Controller Manager rollout success --------- 43.78s
openshift_excluder : Install docker excluder - yum ------------------------------ 37.32s
openshift_node : Install GlusterFS storage plugin dependencies ------------------ 31.13s
openshift_loadbalancer : Install haproxy ---------------------------------------- 30.26s
openshift_node : Install dnsmasq ------------------------------------------------ 29.97s
openshift_node : Install iSCSI storage plugin dependencies ---------------------- 25.03s
openshift_web_console : Verify that the console is running ---------------------- 21.61s
openshift_service_catalog : oc_process ------------------------------------------ 13.35s
```

15. Check the pods status after installation:

```
oc login -u system:admin
Logged into "https://ocp.domain.example.com:8443" as "system:admin" using existing
credentials.
oc get pod --all-namespaces
NAMESPACE                        NAME                                            READY STATUS RESTARTS AGE
default                          docker-registry-1-tmbfn                         1/1       Running   0 19m
default                          registry-console-1-p5clt                        1/1       Running   0 19m
default                          router-1-2hm94                                  1/1       Running   0 19m
default                          router-1-2s9zn                                  1/1       Running   0 19m
default                          router-1-72f7r                                  1/1       Running   0 19m
kube-service-catalog             apiserver-26kt5                                 1/1       Running   0 17m
...
Output truncated
...
openshift-sdn                    sdn-vx867                                       1/1       Running   2 26m
openshift-template-service-broker   apiserver-2s57k                              1/1       Running   0 15m
openshift-template-service-broker   apiserver-4kqhm                              1/1       Running   0 15m
openshift-template-service-broker   apiserver-j7g57                              1/1       Running   0 15m
openshift-web-console            webconsole-6f5f94c675-2kvmv                     1/1       Running   0 19m
openshift-web-console            webconsole-6f5f94c675-jfl2l                     1/1       Running   0 19m
openshift-web-console            webconsole-6f5f94c675-xcx7g                     1/1       Running   0 19m
```

16. For the seven nodes deployment, update the HAProxy configuration to support HTTP and HTTPS, as shown in Example 6-10.

Example 6-10 HAProxy configuration

```
cat > /etc/haproxy/haproxy.cfg <<EOF_haproxy.cfg
# Global settings
#---------------------------------------------------------------------
global
    maxconn     20000
    log         /dev/log local0 info
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    user        haproxy
    group       haproxy
    daemon

    # turn on stats unix socket
    stats socket /var/lib/haproxy/stats

#---------------------------------------------------------------------
# common defaults that all the 'listen' and 'backend' sections will
# use if not designated in their block
#---------------------------------------------------------------------
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
# option http-server-close
#    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          300s
    timeout server          300s
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 20000

listen stats
    bind :9000
    mode http
    stats enable
    stats uri /

frontend  atomic-openshift-api
    bind *:8443
    default_backend atomic-openshift-api
    mode tcp
    option tcplog

backend atomic-openshift-api
    balance source
    mode tcp
    server      master0 192.168.11.202:8443 check
    server      master1 192.168.11.203:8443 check
    server      master2 192.168.11.204:8443 check
```

```
frontend openshift-router80
    bind *:80
    default_backend openshift-router80
    mode tcp
    option tcplog

backend openshift-router80
    balance source
    mode tcp
    server      master0 192.168.11.202:80 check
    server      master1 192.168.11.203:80 check
    server      master2 192.168.11.204:80 check

frontend openshift-router443
    bind *:443
    default_backend openshift-router443
    mode tcp
    option tcplog

backend openshift-router443
    balance source
    mode tcp
    server      master0 192.168.11.202:443 check
    server      master1 192.168.11.203:443 check
    server      master2 192.168.11.204:443 check
EOF_haproxy.cfg
```

17. Restart the haproxy service:

```
systemctl restart haproxy
systemctl status haproxy
? haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/usr/lib/systemd/system/haproxy.service; enabled; vendor preset: disabled)
  Drop-In: /etc/systemd/system/haproxy.service.d
           ··limits.conf
   Active: active (running) since Sun 2019-11-24 06:40:12 UTC; 6s ago
 Main PID: 81816 (haproxy-systemd)
   CGroup: /system.slice/haproxy.service
           ··81816 /usr/sbin/haproxy-systemd-wrapper -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
           ··81817 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -Ds
           ··81818 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -Ds

Nov 24 06:40:12 lbsnode01.domain.example.com systemd[1]: Started HAProxy Load Balancer.
Nov 24 06:40:12 lbsnode01.domain.example.com haproxy-systemd-wrapper[81816]: haproxy-systemd-wrapper: executing
/usr/sbin/haproxy -f /etc/haproxy/hap...d -Ds
Nov 24 06:40:12 lbsnode01.domain.example.com haproxy[81817]: Proxy stats started.
Nov 24 06:40:12 lbsnode01.domain.example.com haproxy[81817]: Proxy atomic-openshift-api started.
Nov 24 06:40:12 lbsnode01.domain.example.com haproxy[81817]: Proxy atomic-openshift-api started.
Nov 24 06:40:12 lbsnode01.domain.example.com haproxy[81817]: Proxy openshift-router80 started.
Nov 24 06:40:12 lbsnode01.domain.example.com haproxy[81817]: Proxy openshift-router80 started.
Nov 24 06:40:12 lbsnode01.domain.example.com haproxy[81817]: Proxy openshift-router443 started.
Nov 24 06:40:12 lbsnode01.domain.example.com haproxy[81817]: Proxy openshift-router443 started.
Hint: Some lines were ellipsized, use -l to show in full.
```

18. Associate the cluster-admin role to the admin user by using the following command:

```
oc adm policy add-cluster-role-to-user cluster-admin admin
```

6.3.5 Uninstalling the OpenShift Container Platform

Complete the following steps to uninstall the OpenShift Container Platform:

1. Unregister OpenShift VMs from the RHSM:

```
ansible -i <inventory_file> nodes,lb -a 'subscription-manager remove --all'
ansible -i <inventory_file> nodes,lb -a 'subscription-manager unregister'
ansible -i <inventory_file> nodes,lb -a 'subscription-manager clean'
wrknode01.domain.example.com | SUCCESS | rc=0 >>
Unregistering from: subscription.rhsm.redhat.com:443/subscription
System has been unregistered.
...
Output truncated
...
lbsnode01.domain.example.com | SUCCESS | rc=0 >>
Unregistering from: subscription.rhsm.redhat.com:443/subscription
System has been unregistered.
```

2. Destroy the PowerVC infrastructure using Terraform:

```
terrafrom destroy
data.openstack_images_image_v2.vm1-image-name: Refreshing state...
...
Output truncated
...
Plan: 0 to add, 0 to change, 33 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
null_resource.vm4_post_install_config[0]: Destroying... [id=1118127418209700949]
null_resource.vm4_post_install_config[0]: Destruction complete after 0s
openstack_compute_volume_attach_v2.vm1_va_dockerdisk1[2]: Destroying...
[id=5d929233-f9fc-4224-8960-5f1f050e4174/2563294f-d9ce-44a6-ac04-afc73a07593e]
openstack_compute_volume_attach_v2.vm1_va_dockerdisk1[0]: Destroying...
[id=f7c4966f-7036-4494-bd35-8e66b84ed18e/3f5bc9d4-c91e-42db-b529-f7b03a1e963c]
openstack_compute_volume_attach_v2.vm3_va_dockerdisk1[0]: Destroying...
[id=c47782d9-2d1b-4430-8d58-a755d652b07d/23c91f6b-790c-4af1-bda8-ee471ff75f1b]
openstack_compute_volume_attach_v2.vm3_va_dockerdisk1[1]: Destroying...
[id=8cf475c5-9315-4032-9400-e46a1c6a3b7d/4b3c6d75-985b-4ba2-8363-8dd16e76e131]
openstack_compute_volume_attach_v2.vm1_va_dockerdisk1[1]: Destroying...
[id=d6305c81-efd1-4257-bea1-2eb9a5ad8145/0569bceb-753f-48ee-82d0-9e00a3245b96]
openstack_compute_volume_attach_v2.vm3_va_dockerdisk1[2]: Destroying...
[id=4d28442a-fa3d-4389-80d8-2a2fe3199b19/a80c34a6-42fa-4d3d-866b-0a6a4548ea56]

...
Output truncated
...
openstack_networking_subnet_v2.net1-subnet: Destruction complete after 1m58s
openstack_networking_network_v2.net1: Destroying...
[id=42b5da3c-98ff-4f07-ba52-8fcd61e161ab]
openstack_networking_network_v2.net1: Destruction complete after 6s

Destroy complete! Resources: 33 destroyed.
```

6.4 IBM PowerVC FlexVolume Driver

This section describes how to install the IBM PowerVC FlexVolume Driver. After installation, configure the persistent storage for Registry, and install the Prometheus Cluster Monitoring.

6.4.1 Deploying the IBM PowerVC FlexVolume Driver

Complete the following steps to install IBM PowerVC FlexVolume driver:

3. Create a project for PowerVC FlexVolume Driver:

```
export NS=powervc-flexvoldrv
oc login -u system:admin
Logged into "https://ocp.domain.example.com:8443" as "system:admin" using existing
credentials.

You have access to the following projects and can switch between them with 'oc project
<projectname>':
...
Output truncated
...
Using project "default".
oc new-project $NS --description="PowerVC FlexVolume Driver for Containers"
--display-name="PowerVC FlexVolume Driver"
Now using project "powervc-flexvoldrv" on server "https://ocp.domain.example.com:8443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby
oc project $NS
Already on project "powervc-flexvoldrv" on server "https://ocp.domain.example.com:8443
```

4. Patch the project to clear the `nodeSelector` tag:

```
oc patch namespace $NS -p '{"metadata": {"annotations":
{"openshift.io/node-selector": ""}}}'
namespace/powervc-flexvoldrv patched
```

5. Assign the cluster-admin role to the $NS project's default service account:

```
oc adm policy add-cluster-role-to-user cluster-admin
system:serviceaccount:$NS:default
cluster role "cluster-admin" added: "system:serviceaccount:powervc-flexvoldrv:default"
```

6. Set the `hostmount-anyuid` source code control to the user root, or the user ID that was used to create the $NS project:

```
oc adm policy add-scc-to-user hostmount-anyuid
system:serviceaccount:$NS:default
scc "hostmount-anyuid" added to: ["system:serviceaccount:powervc-flexvoldrv:default"]
```

7. Create a secret with the PowerVC user name and password:

```
oc create secret generic -n $NS powervc-secret \
    --from-literal=OS_USERNAME=ocpadmin \
    --from-literal=OS_PASSWORD=<password>
secret/powervc-secret created
```

8. Download the `power-openstack-k8s-volume-driver` file from this web page.

```
wget
https://raw.githubusercontent.com/IBM/power-openstack-k8s-volume-driver/master/
template/ibm-powervc-k8s-volume-driver-template.yaml
--2019-11-24 10:18:54--
https://raw.githubusercontent.com/IBM/power-openstack-k8s-volume-driver/master/template/ibm-powervc-k8s-volume-dri
ver-template.yaml
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 199.232.36.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|199.232.36.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 14005 (14K) [text/plain]
Saving to: ‘ibm-powervc-k8s-volume-driver-template.yaml’

100%[======================================================================>] 14,005      --.-K/s   in 0.07s

2019-11-24 10:18:55 (189 KB/s) - ‘ibm-powervc-k8s-volume-driver-template.yaml’ saved [14005/14005]
```

9. Check the driver configuration parameters:

```
oc process --parameters -n $NS -f ibm-powervc-k8s-volume-driver-template.yaml
NAME                            DESCRIPTION GENERATOR          VALUE
OPENSTACK_IP_OR_HOSTNAME        IP address or host name of the PowerVC management server. It will be used to
construct the PowerVC authentication URL.
OPENSTACK_CRED_SECRET_NAME      Name of the pre-created Secret object that contains the PowerVC admin username and
password.
OPENSTACK_CERT_DATA             Paste the contents (in PEM) from the /etc/pki/tls/certs/powervc.crt file. If left
blank, certificate verification is not done, which is insecure.
OPENSTACK_PROJECT_NAME          The project to use. The specified PowerVC user must have the administrator role in
this project.                                                              ibm-default
OPENSTACK_DOMAIN_NAME           The name of the project domain for the PowerVC user. Default
DRIVER_FLEX_PLUGIN_DIR          The directory that Kubernetes uses for the FlexVolume driver. Keep the default
value unless the --flex-volume-plugin-dir flag has been overridden.
/usr/libexec/kubernetes/kubelet-plugins/volume/exec/
DRIVER_VOLUME_TYPE              The name or ID of the default storage class's volume type (storage template). This
can specify the default storage details for persistent volumes.
DRIVER_DFLT_STG_CLASS           The default storage class is used if no storage class is specified when creating a
persistent volume claim.                                                   true
IMAGE_PROVISIONER_REPO    Name and location of the provisioner docker image repository.
ibmcom/power-openstack-k8s-volume-provisioner
IMAGE_PROVISIONER_TAG           Tag or label for the provisioner docker image. The default value is the only
supported version.                                                               1.0.2
IMAGE_FLEX_VOLUME_REPO    Name and location of the flexvolume docker image repository.
ibmcom/power-openstack-k8s-volume-flex
IMAGE_FLEX_VOLUME_TAG           Tag or label for the flexvolume docker image. The default value is the only
supported version.                                                                1.0.2
IMAGE_PROVISIONER_PULL          Pull policy for the provisioner docker image. IfNotPresent
IMAGE_FLEX_VOLUME_PULL          Pull policy for the flexvolume docker image. IfNotPresent
SECURITY_SERVICE_ACCOUNT_NAME   Name of the service account to use default
```

To get the value of OPENSTACK_CERT_DATA, run the following command on the PowerVC server:

```
cat /etc/pki/tls/certs/powervc.crt | tr '\n' ' '
-----BEGIN CERTIFICATE----- MIIDfTCCAmWgAwIBAg...[Output truncated]...1a3MBWLVmHzQ+OwFTN
-----END CERTIFICATE-----
```

10.Create the template environment file to store the PowerVC details:

```
cat > ibm-powervc-k8s-volume-driver.env <<EOF_ibm-powervc-k8s-volume-driver.env
OPENSTACK_IP_OR_HOSTNAME=rbpvc01.domain.example.com
OPENSTACK_CRED_SECRET_NAME=powervc-secret
OPENSTACK_CERT_DATA=-----BEGIN CERTIFICATE----- MIIDf...[Output truncated]...sOwFTN
-----END CERTIFICATE-----
OPENSTACK_PROJECT_NAME=ocp-project
IMAGE_PROVISIONER_REPO=ibmcom/power-openstack-k8s-volume-provisioner
IMAGE_PROVISIONER_TAG=1.0.2
IMAGE_FLEX_VOLUME_REPO=ibmcom/power-openstack-k8s-volume-flex
IMAGE_FLEX_VOLUME_TAG=1.0.2
SECURITY_SERVICE_ACCOUNT_NAME=default
EOF_ibm-powervc-k8s-volume-driver.env
```

11.Deploy the PowerVC FlexVolume Driver application from the template by using the environment file:

```
oc process -f ibm-powervc-k8s-volume-driver-template.yaml
--param-file=ibm-powervc-k8s-volume-driver.env |  oc create -f -
configmap/ibm-powervc-config created
storageclass.storage.k8s.io/ibm-powervc-k8s-volume-default created
deployment.apps/ibm-powervc-k8s-volume-provisioner created
daemonset.apps/ibm-powervc-k8s-volume-flex created
```

12.Verify the installation. You have one running volume-flex Pod per each node, and one volume-provisioner:

```
oc get pods
NAME                                                  READY     STATUS    RESTARTS   AGE
ibm-powervc-k8s-volume-flex-2jx8t                     1/1       Running   0          23s
ibm-powervc-k8s-volume-flex-fgjz6                     1/1       Running   0          23s
ibm-powervc-k8s-volume-flex-gb5xq                     1/1       Running   0          23s
ibm-powervc-k8s-volume-flex-kpqfk                     1/1       Running   0          23s
ibm-powervc-k8s-volume-flex-lp84r                     1/1       Running   0          23s
ibm-powervc-k8s-volume-flex-trl5t                     1/1       Running   0          23s
ibm-powervc-k8s-volume-provisioner-6cc7d64c4b-dfxsn   1/1       Running   0          23s
```

13.(Optional) Uninstall PowerVC FlexVolume Driver:

```
####### delete pods for powervc flex volume and start over #######
oc delete deployment ibm-powervc-k8s-volume-provisioner
oc delete ds ibm-powervc-k8s-volume-flex
oc delete cm ibm-powervc-config
oc delete sc ibm-powervc-k8s-volume-default
###### delete the template #######
oc delete template ibm-powervc-k8s-volume-driver
oc delete secret powervc-secret
```

6.4.2 Creating the persistent storage for the Registry

Complete the following steps to set up the persistent storage for the Registry:

1. Create a Persistent Volume Claim:

```
oc login -u system:admin
Logged into "https://mstnode01.domain.example.com:8443" as "system:admin" using existing
credentials.
You have access to the following projects and can switch between them with 'oc project
<projectname>':
...
Output truncated
...
Using project "powervc-flexvoldrv".
oc project default
Now using project "default" on server "https://ocp.domain.example.com:8443".
cat > registry-pvc.yml <<EOF_/registry-pvc.yml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: registry-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1024Gi
EOF_/registry-pvc.yml
oc create -f registry-pvc.yml
persistentvolumeclaim/registry-pvc created
oc get pvc
NAME           STATUS    VOLUME                                     CAPACITY   ACCESS
MODES   STORAGECLASS AGE
registry-pvc   Bound     pvc-fb388f7a-0fb4-11ea-a12a-faa21eff8220   1Ti        RWO
ibm-powervc-k8s-volume-default   7s
oc get dc
NAME               REVISION   DESIRED   CURRENT   TRIGGERED BY
docker-registry    1          1         1         config
registry-console   1          1         1         config
router             1          3         3         config
oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
docker-registry-1-tmbfn    1/1       Running   0          41m
registry-console-1-p5clt   1/1       Running   0          41m
router-1-2hm94             1/1       Running   0          41m
router-1-2s9zn             1/1       Running   0          41m
router-1-72f7r             1/1       Running   0          41m
oc set volume dc/docker-registry
deploymentconfigs/docker-registry
  empty directory as registry-storage
    mounted at /registry
  secret/registry-certificates as registry-certificates
    mounted at /etc/secrets
```

2. Assign the Persistent Volume Claim to the Registry:

```
oc set volume dc/docker-registry --add --name=registry-storage -t pvc
--claim-name=registry-pvc --mount-path=/registry --overwrite
deploymentconfig.apps.openshift.io/docker-registry volume updated
oc set volume dc/docker-registry
deploymentconfigs/docker-registry
  secret/registry-certificates as registry-certificates
    mounted at /etc/secrets
  pvc/registry-pvc (allocated 1TiB) as registry-storage
    mounted at /registry
oc get pod
NAME                        READY     STATUS    RESTARTS   AGE
docker-registry-2-8h9cf     1/1       Running   0          1m
registry-console-1-p5clt    1/1       Running   0          46m
router-1-2hm94              1/1       Running   0          47m
router-1-2s9zn              1/1       Running   0          47m
router-1-72f7r              1/1       Running   0          47m
oc exec -ti docker-registry-2-8h9cf bash
bash-4.2$ df -h
Filesystem                        Size  Used Avail Use% Mounted on
/dev/mapper/mpatha               1008G   77M  957G   1% /registry
bash-4.2$ mount | grep regi
/dev/mapper/mpatha on /registry type ext4 (rw,relatime,seclabel,data=ordered)
bash-4.2$ exit
exit
```

6.4.3 Deploying the Prometheus Cluster Monitoring

This section describes the Prometheus Cluster Monitoring open source deployment.

Overview

OpenShift Container Platform includes a pre-configured and self-updating monitoring stack (see Figure 6-5 on page 143) that is based on the Prometheus open source project and its wider system.

OpenShift provides monitoring of cluster components, and includes a set of alerts to immediately notify the cluster administrator about any occurring problems, and a set of Grafana dashboards.

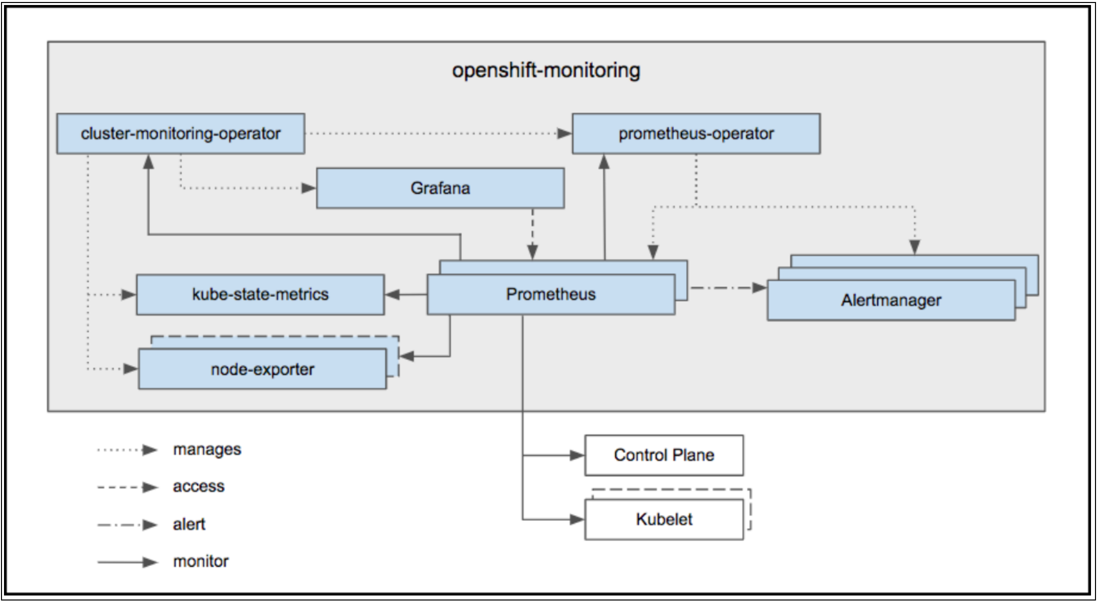


Figure 6-5 OpenShift monitoring

As highlighted in Figure 6-5, at the heart of the monitoring stack sits the OpenShift Container Platform Cluster Monitoring Operator (CMO), which watches over the deployed monitoring components and resources and ensures that these are always up-to-date.

The Prometheus Operator (PO) creates, configures, and manages Prometheus and Alertmanager instances. It also automatically generates monitoring target configurations based on familiar Kubernetes label queries.

In addition to Prometheus and Alertmanager, OpenShift Container Platform Monitoring includes node-exporter and kube-state-metrics. Node-exporter is an agent that is deployed on every node to collect metrics about it. The kube-state-metrics exporter agent converts Kubernetes objects to metrics consumable by Prometheus.

The following targets are monitored as part of the cluster monitoring:

- Prometheus itself
- Prometheus-Operator
- cluster-monitoring-operator
- Alertmanager cluster instances
- Kubernetes apiserver
- kubelets (the kubelet embeds cAdvisor for per container metrics)
- kube-controllers
- kube-state-metrics
- node-exporter
- etcd (if etcd monitoring is enabled)

All of these components are automatically updated. For more information about the OpenShift Container Platform Cluster Monitoring Operator, see this web page.

Configuring OpenShift Container Platform cluster monitoring

The OpenShift Cluster Monitoring is deployed by Ansible Playbooks. You can choose the base playbook or the openshift-monitoring playbook for the deployment.

The Monitoring stack is installed with OpenShift Container Platform by default. You can prevent it from being installed by setting `openshift_cluster_monitoring_operator_install=False` in the Ansible inventory file.

By default, persistent storage is disabled for Prometheus time-series data and for Alertmanager notifications and silences. You can configure the cluster to persistently store any one of them or both.

To enable persistent storage of Prometheus time-series data, set `openshift_cluster_monitoring_operator_prometheus_storage_enabled=True` in the Ansible inventory file.

To enable persistent storage of Alertmanager notifications and silences, set `openshift_cluster_monitoring_operator_alertmanager_storage_enabled=True` in the Ansible inventory file.

To specify the size of the persistent volume claim for Prometheus and Alertmanager, change the following Ansible variables:

- `openshift_cluster_monitoring_operator_prometheus_storage_capacity (default: 50Gi)`
- `openshift_cluster_monitoring_operator_alertmanager_storage_capacity (default: 2Gi)`

Each of these variables applies only if its corresponding `storage_enabled` variable is set to `True`.

After you enable dynamic storage, you can also set the `storageclass` for the persistent volume claim for each component in the Ansible inventory file:

- `openshift_cluster_monitoring_operator_prometheus_storage_class_name (default: "")`
- `openshift_cluster_monitoring_operator_alertmanager_storage_class_name (default: "")`

Each of these variables applies only if its corresponding `storage_enabled` variable is set to `True`.

Example 6-11 shows a sample Cluster Monitoring in the Ansible inventory file.

Example 6-11 Cluster Monitoring in the Ansible inventory file

```
##-----------------------------------------------------------------------##
## Prometheus Cluster Monitoring
##-----------------------------------------------------------------------##
https://docs.openshift.com/container-platform/3.11/install_config/prometheus_cluster_monito
ring.html
openshift_cluster_monitoring_operator_install=False
openshift_prometheus_node_selector={"node-role.kubernetes.io/infra":"true"}
# Enable persistent storage of Prometheus time-series data (default False)
openshift_cluster_monitoring_operator_prometheus_storage_enabled=False
# Enable persistent storage of Alertmanager notifications (default False)
openshift_cluster_monitoring_operator_alertmanager_storage_enabled=False
# Dynamic storage allocation for Prometheus services
openshift_cluster_monitoring_operator_prometheus_storage_capacity=32Gi
openshift_cluster_monitoring_operator_alertmanager_storage_capacity=4Gi
# Storage class to use if persistent storage enabled
# NOTE: it will use storageclass default if storage class not specified
openshift_cluster_monitoring_operator_prometheus_storage_class_name='ibm-powervc-k8s-volume
-default'
```

```
openshift_cluster_monitoring_operator_alertmanager_storage_class_name='ibm-powervc-k8s-volu
me-default'
```

The monitoring playbook is available at the following directory:

`/usr/share/ansible/openshift-ansible/playbooks/openshift-monitoring/`

Complete the following steps:

1. Deploying OpenShift Cluster Monitoring:

```
ansible-playbook -i <inventory_file>
/usr/share/ansible/openshift-ansible/playbooks/openshift-monitoring/config.yml\
  -e openshift_cluster_monitoring_operator_install=True
PLAY [Initialization Checkpoint Start]
****************************************************************************************
...
Output truncated
...
INSTALLER STATUS openshift_cluster_monitoring_operator : Wait for the ServiceMonitor CRD
to be created ------------------------------------------------------------------ 31.03s
Gathering Facts ----------------------------------------------------------------- 2.98s
set_fact ------------------------------------------------------------------------ 1.78s
openshift_cluster_monitoring_operator : Add monitoring project ------------------ 1.66s
openshift_cluster_monitoring_operator : Label monitoring namespace -------------- 1.42s
openshift_sanitize_inventory : include_tasks ------------------------------------ 1.13s
openshift_sanitize_inventory : include_tasks ------------------------------------ 1.11s
Gather Cluster facts ------------------------------------------------------------ 1.11s
openshift_sanitize_inventory : At least one master is schedulable --------------- 1.09s
Initialize openshift.node.sdn_mtu ----------------------------------------------- 1.09s
openshift_control_plane : Retrieve list of schedulable nodes matching selector -- 1.05s
openshift_sanitize_inventory : Check for usage of deprecated variables ---------- 1.00s
set_fact openshift_portal_net if present on masters ----------------------------- 0.96s
get openshift_current_version --------------------------------------------------- 0.95s
set_fact ------------------------------------------------------------------------ 0.95s
openshift_cluster_monitoring_operator : Copy files to temp directory ------------ 0.93s
Validate openshift_node_groups and openshift_node_group_name -------------------- 0.82s
```

2. Verify the Cluster Monitoring installation:

```
oc get pod -n openshift-monitoring
NAME                                           READY     STATUS    RESTARTS   AGE
alertmanager-main-0                            3/3       Running   0          4m
alertmanager-main-1                            3/3       Running   0          4m
alertmanager-main-2                            3/3       Running   0          3m
cluster-monitoring-operator-68fb779747-b6nd7   1/1       Running   0          6m
grafana-5756774f8f-7lrlm                       2/2       Running   0          6m
kube-state-metrics-79f458bd6c-qtv88            3/3       Running   0          2m
node-exporter-724ww                            2/2       Running   0          2m
node-exporter-jbjm5                            2/2       Running   0          2m
node-exporter-nrszt                            2/2       Running   0          2m
node-exporter-shclk                            2/2       Running   0          2m
node-exporter-vfzjr                            2/2       Running   0          2m
node-exporter-wbjd7                            2/2       Running   0          2m
prometheus-k8s-0                               4/4       Running   1          6m
prometheus-k8s-1                               4/4       Running   1          5m
prometheus-operator-88dcddf7d-22hd2            1/1       Running   0          6m
oc get pvc -n openshift-monitoring
NAME                                       STATUS    VOLUME                                     CAPACITY   ACCESS
MODES   STORAGECLASS                      AGE
alertmanager-main-db-alertmanager-main-0  Bound    pvc-569cfeb2-0fc2-11ea-ac13-faa21eff8220   4Gi      RWO
ibm-powervc-k8s-volume-default   5m
alertmanager-main-db-alertmanager-main-1  Bound    pvc-690150be-0fc2-11ea-ac13-faa21eff8220   4Gi      RWO
ibm-powervc-k8s-volume-default   5m
alertmanager-main-db-alertmanager-main-2  Bound    pvc-7b98bfa8-0fc2-11ea-ac13-faa21eff8220   4Gi      RWO
ibm-powervc-k8s-volume-default   4m
prometheus-k8s-db-prometheus-k8s-0       Bound    pvc-20342da0-0fc2-11ea-ac13-faa21eff8220   32Gi      RWO
ibm-powervc-k8s-volume-default   7m
prometheus-k8s-db-prometheus-k8s-1       Bound    pvc-364ff7f6-0fc2-11ea-ac13-faa21eff8220   32Gi      RWO
ibm-powervc-k8s-volume-default   6m
```

6.5 Managing OpenShift Resources using CLI

OpenShift Container Platform organizes entities in the OpenShift cluster as objects that are managed by the master node, which are collectively known as resources:

- Projects (namespaces)
- Users
- Deployment Configuration
- Nodes
- Services
- Pods

These resources are available to the OpenShift Container Platform.

Red Hat OpenShift Container Platform includes a command-line tool that enables system administrators and developers to work with an OpenShift cluster. The oc command-line tool provides the ability to modify and manage resources throughout the delivery lifecycle of a software development project. Common operations with this tool include deploying applications, scaling applications, and checking the status of projects.

The `oc` command-line tool is installed on all master and node machines. You can also install the `oc` client on systems that are not part of the OpenShift cluster, such as administrator machines. When it is installed, you can issue commands after authenticating to any master node.

On Red Hat Enterprise Linux (RHEL), the `oc` tool is available as an RPM file and installable by using the **`yum install`** command:

```
yum install -y atomic-openshift-clients
Loaded plugins: product-id, search-disabled-repos, subscription-manager
Resolving Dependencies
--> Running transaction check
---> Package atomic-openshift-clients.ppc64le 0:3.11.153-1.git.0.aaf3f71.el7 will be
installed
--> Finished Dependency Resolution
...
Output truncated
...
Complete!
```

After the `oc` CLI tool is installed, use the **`oc help`** command to display help information.

You can use the **`oc login`** command to log in interactively, which prompts you for a server name, a user name, and a password. You also can include the required information about the command line:

```
oc login https://ocp.domain.example.com:8443 -u admin
The server uses a certificate signed by an unknown authority.
You can bypass the certificate check, but any data you send to the server could be
intercepted by others.
Use insecure connections? (y/n): y
Authentication required for https://ocp.domain.example.com:8443 (openshift)
Username: admin
Password:
Login successful.
```

Important: It is possible to log in as the OpenShift cluster administrator from any master node without a password by connecting by way of **`ssh`** to the master node.

After successful authentication from a client, OpenShift saves an authorization token in the user's home folder.

To check your current credentials, run the **`oc whoami`** command:

```
oc whoami
admin
```

To log out of the OpenShift cluster, use the **`oc logout`** command:

```
oc logout
Logged "admin" out on "https://ocp.domain.example.com:8443"
```

As an administrator, the **`oc get RESOURCE_TYPE [RESOURCE_NAME]`** command is used most frequently. This command helps to get information about resources in the cluster. Generally, this command displays only the most important characteristics of the resources and omits more detailed information.

If the RESOURCE_NAME parameter is omitted, all resources of the specified RESOURCE_TYPE are summarized, as shown in Example 6-12.

Example 6-12 oc get pod

```
# oc get pod
NAME                      READY     STATUS    RESTARTS   AGE
docker-registry-3-4flql   1/1       Running   2          1d
router-2-4gnmj            1/1       Running   3          1d
router-2-cp5sf            1/1       Running   3          1d
router-2-slkjf            1/1       Running   3          1d
```

Use the **oc types** command for a quick refresher on the concepts of the available RESOURCE_TYPES, as shown in Example 6-13.

Example 6-13 oc types

```
# oc types
Command "types" is deprecated, refer to official documentation instead
Concepts and Types

Kubernetes and OpenShift help developers and operators build, test, and deploy applications
in a containerized cloud
environment. Applications may be composed of all of the components below, although most
developers will be concerned
with Services, Deployments, and Builds for delivering changes.

Concepts:

* Containers:
    A definition of how to run one or more processes inside of a portable Linux
    environment. Containers are started from an Image and are usually isolated
    from other containers on the same machine.

* Image:
    A layered Linux filesystem that contains application code, dependencies,
    and any supporting operating system libraries. An image is identified by
    a name that can be local to the current cluster or point to a remote Docker
    registry (a storage server for images).

* Pods [pod]:
    A set of one or more containers that are deployed onto a Node together and
    share a unique IP and Volumes (persistent storage). Pods also define the
    security and runtime policy for each container.

* Labels:
    Labels are key value pairs that can be assigned to any resource in the
    system for grouping and selection. Many resources use labels to identify
    sets of other resources.

* Volumes:
    Containers are not persistent by default - on restart their contents are
    cleared. Volumes are mounted filesystems available to Pods and their
    containers which may be backed by a number of host-local or network
    attached storage endpoints. The simplest volume type is EmptyDir, which
    is a temporary directory on a single machine. Administrators may also
    allow you to request a Persistent Volume that is automatically attached
    to your pods.

* Nodes [node]:
    Machines set up in the cluster to run containers. Usually managed
    by administrators and not by end users.

* Services [svc]:
    A name representing a set of pods (or external servers) that are
    accessed by other pods. The service gets an IP and a DNS name, and can be
    exposed externally to the cluster via a port or a Route. It's also easy
    to consume services from pods because an environment variable with the
    name <SERVICE>_HOST is automatically injected into other pods.

* Routes [route]:
    A route is an external DNS entry (either a top level domain or a
    dynamically allocated name) that is created to point to a service so that
    it can be accessed outside the cluster. The administrator may configure
    one or more Routers to handle those routes, typically through an Apache
    or HAProxy load balancer / proxy.

* Replication Controllers [rc]:
    A replication controller maintains a specific number of pods based on a
    template that match a set of labels. If pods are deleted (because the
    node they run on is taken out of service) the controller creates a new
    copy of that pod. A replication controller is most commonly used to
    represent a single deployment of part of an application based on a
    built image.

* Deployment Configuration [dc]:
    Defines the template for a pod and manages deploying new images or
    configuration changes whenever those change. A single deployment
    configuration is usually analogous to a single micro-service. Can support
    many different deployment patterns, including full restart, customizable
    rolling updates, and fully custom behaviors, as well as pre- and post-
    hooks. Each deployment is represented as a replication controller.

* Build Configuration [bc]:
    Contains a description of how to build source code and a base image into a
    new image - the primary method for delivering changes to your application.
    Builds can be source based and use builder images for common languages like
    Java, PHP, Ruby, or Python, or be Docker based and create builds from a
    Dockerfile. Each build configuration has web-hooks and can be triggered
    automatically by changes to their base images.

* Builds [build]:
    Builds create a new image from source code, other images, Dockerfiles, or
    binary input. A build is run inside of a container and has the same
    restrictions normal pods have. A build usually results in an image pushed
    to a Docker registry, but you can also choose to run a post-build test that
    does not push an image.

* Image Streams and Image Stream Tags [is,istag]:
    An image stream groups sets of related images under tags - analogous to a
    branch in a source code repository. Each image stream may have one or
    more tags (the default tag is called "latest") and those tags may point
    at external Docker registries, at other tags in the same stream, or be
    controlled to directly point at known images. In addition, images can be
    pushed to an image stream tag directly via the integrated Docker
    registry.

* Secrets [secret]:
    The secret resource can hold text or binary secrets for delivery into
    your pods. By default, every container is given a single secret which
    contains a token for accessing the API (with limited privileges) at
    /var/run/secrets/kubernetes.io/serviceaccount. You can create new
    secrets and mount them in your own pods, as well as reference secrets
    from builds (for connecting to remote servers) or use them to import
    remote images into an image stream.

* Projects [project]:
    All of the above resources (except Nodes) exist inside of a project.
    Projects have a list of members and their roles, like viewer, editor,
    or admin, as well as a set of security controls on the running pods, and
    limits on how many resources the project can use. The names of each
    resource are unique within a project. Developers may request projects
    be created, but administrators control the resources allocated to
    projects.

For more, see https://docs.openshift.com

Usage:
  oc types [flags]

Examples:
  # View all projects you have access to
  oc get projects

  # See a list of all services in the current project
  oc get svc

  # Describe a deployment configuration in detail
  oc describe dc mydeploymentconfig

  # Show the images tagged into an image stream
  oc describe is ruby-centos7

Use "oc options" for a list of global command-line options (applies to all commands).
```

To get a summary of the most important components of the cluster, use the **oc get all** command. This command iterates through the major resource types and prints out an information summary:

```
oc get all
NAME                        READY     STATUS    RESTARTS   AGE
pod/docker-registry-3-4flql   1/1       Running   2          1d
pod/router-2-4gnmj          1/1       Running   3          1d
pod/router-2-cp5sf          1/1       Running   3          1d
pod/router-2-slkjf          1/1       Running   3          1d

NAME                                   DESIRED   CURRENT   READY     AGE
replicationcontroller/docker-registry-1   0         0         0         1d
replicationcontroller/docker-registry-2   0         0         0         1d
replicationcontroller/docker-registry-3   1         1         1         1d
replicationcontroller/router-1         0         0         0         1d
replicationcontroller/router-2         3         3         3         1d

NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                   AGE
service/docker-registry   ClusterIP   172.30.26.207   <none>        5000/TCP                  1d
service/kubernetes       ClusterIP   172.30.0.1      <none>        443/TCP,53/UDP,53/TCP     1d
service/router           ClusterIP   172.30.1.163    <none>        80/TCP,443/TCP,1936/TCP   1d

NAME                                                 REVISION   DESIRED   CURRENT   TRIGGERED BY
deploymentconfig.apps.openshift.io/docker-registry   3          1         1         config
deploymentconfig.apps.openshift.io/router            2          3         3         config
```

A useful option is available that you can add to the **oc get** command: the **-w** option. This option continuously watches the output in real-time. This option is useful, for example, for monitoring the output of an **oc get pod** command continuously instead of running it multiple times.

If the details that are provided by the **oc get** command are not sufficient, more information about the resource can be retrieved by using the **oc describe RESOURCE_TYPE RESOURCE_NAME** command. Unlike the **oc get** command, there is no way to iterate through all of the different resources by type. Although most major resources can be described, this functionality is not available across all resources. To display detailed information about a Pod resource use the following command:

```
oc describe pod docker-registry-3-4flql
Name:               docker-registry-3-4flql
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               mstnode02.domain.example.com/192.168.11.203
Start Time:         Thu, 07 Nov 2019 16:09:12 +0000
Labels:             deployment=docker-registry-3
                    deploymentconfig=docker-registry
                    docker-registry=default
Annotations:        openshift.io/deployment-config.latest-version=3
                    openshift.io/deployment-config.name=docker-registry
                    openshift.io/deployment.name=docker-registry-3
                    openshift.io/scc=restricted
Status:             Running
IP:                 10.130.0.48
...
Output truncated
...
Volumes:
  registry-storage:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
     ClaimName:  registry-pvc
    ReadOnly:   false
  registry-token-zg2tb:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  registry-token-zg2tb
    Optional:    false
QoS Class:       Burstable
Node-Selectors:  node-role.kubernetes.io/infra=true
Tolerations:     node.kubernetes.io/memory-pressure:NoSchedule
Events:          <none>
```

Use the **oc export RESOURCE_TYPE RESOURCE_NAME [-o OUTPUT_FORMAT]** command to export a definition of a resource. Typical use cases include creating a backup, or to aid in modifying a definition. By default, the export command prints out the object representation in YAML format, but this can be changed by providing the **-o** option.

Use the **oc create** command to create resources from a resource definition. Typically, this is paired with the **oc export** command for editing definitions.

Use the **oc delete RESOURCE_TYPE RESOURCE_NAME** command to remove a resource from the OpenShift cluster.

Note: A fundamental understanding of the OpenShift architecture is needed because deleting managed resources, such as pods, results in newer instances of those resources being automatically recreated.

The **oc new-app** command can create application pods to run on OpenShift in many different ways. It can create pods from Docker images, Docker files, and raw source code by using the Source-to-Image (S2I) process. The command can create a service and a deployment configuration, and a build configuration if source code is used.

Use the **oc exec POD_NAME COMMAND** command to execute commands against a Pod:

```
oc exec docker-registry-3-4flql hostname
docker-registry-3-4flq
```

Use the **oc rsh POD_NAME** command to start a remote shell connection to the Pod. This is useful for logging in and investigating issues in a running Pod:

```
oc rsh docker-registry-3-4flql
sh-4.2$ hostname
docker-registry-3-4flql
sh-4.2$ ls /
bin  boot  config.yml  dev  etc  home  lib  lib64  media  mnt  opt  proc  registry  root
run  sbin  srv  sys  tmp  usr  var
sh-4.2$ exit
exit
```

Use the **oc status** command to get a high-level status of the current project:

```
oc status
In project default on server https://ocp.domain.example.com:8443
svc/docker-registry - 172.30.26.207:5000
  dc/docker-registry deploys registry.redhat.io/openshift3/ose-docker-registry:v3.11.153
    deployment #3 deployed 2 days ago - 1 pod
    deployment #2 deployed 2 days ago
    deployment #1 deployed 2 days ago
svc/kubernetes - 172.30.0.1 ports 443->8443, 53->8053, 53->8053

svc/router - 172.30.1.163 ports 80, 443, 1936
  dc/router deploys registry.redhat.io/openshift3/ose-haproxy-router:v3.11.153
    deployment #2 deployed 2 days ago - 3 pods
    deployment #1 deployed 2 days ago
1 warning identified, use 'oc status --suggest' to see details.
```

The oc command-line client is the primary tool used by administrators to detect and troubleshoot issues in an OpenShift cluster. It has a number of options that enable you to detect, diagnose, and fix issues on the master and worker nodes, services, and resources that are managed by the cluster.

Events allow OpenShift to record information about lifecycle events in a cluster. They help developers and administrators to view information about OpenShift components in an unified way. The **oc get events** command provides information about events in an OpenShift namespace. The following events are captured and reported:

- Master and worker nodes status
- Pod creation and deletion
- Pod placement scheduling

Events are useful during troubleshooting. You can get high-level information about failures and issues in the cluster, and then proceed to investigate by using log files and other **oc** subcommands.

You can get a list of events in a project by using the **oc events** command as follows:

```
oc get events -n default
LAST SEEN   FIRST SEEN   COUNT    NAME                                KIND
SUBOBJECT    TYPE      REASON           SOURCE                               MESSAGE
7m          4d           704      template-service-broker.15d413288a764433
ClusterServiceBroker                Normal    FetchedCatalog
service-catalog-controller-manager   Successfully fetched catalog entries from broker.
2m          4d           1403     ansible-service-broker.15d41326aef38cda
ClusterServiceBroker                Normal    FetchedCatalog
service-catalog-controller-manager   Successfully fetched catalog entries from broker.
```

Note: For more information about events in OpenShift Container Platform 3.11, see this web page.

The **oc logs RESOURCE_NAME** command retrieves the log output for a specific build, deployment, or pod. This command works for builds, build configurations, deployment configurations, and pods.

To view the logs for the registry Pod use the **oc logs**:

```
oc logs docker-registry-3-4flql
time="2019-11-09T19:08:13.824438413Z" level=info msg=response go.version=go1.9.7
http.request.host="10.130.0.48:5000"
http.request.id=274b894e-93b8-4186-9757-f4736ba792a9 http.request.method=GET
http.request.remoteaddr="10.130.0.1:48328" http.request.uri=/healthz
http.request.useragent=kube-probe/1.11+
http.response.duration="53.286µs" http.response.status=200 http.response.written=0
instance.id=29752895-706b-4169-951c-351af53ad214
```

The **oc rsync** command copies the contents to or from a directory in a running pod. If a pod has multiple containers, you can specify the container ID by using the `-c` option. Otherwise, it defaults to the first container in the pod. This is useful for transferring log files and configuration files from the container.

To copy contents from a directory in a pod to a local directory, run the following command:

```
oc rsync <pod>:<pod_dir> <local_dir> [-c <container>]
```

To copy contents from a local directory to a directory in a pod, run the following command:

```
oc rsync <local_dir> <pod>:<pod_dir> -c [<container>]
```

Use the **oc port-forward** command to forward one or more local ports to a pod. This allows to listen on a specific or random port locally, and have data forwarded to and from specific ports in the pod. This command features the following format:

```
oc port-forward <pod> [<local_port>:]<remote_port>
```

Note: The OpenShift Container Platform 3.11 CLI Reference is available at this web page.

6.6 Installing the IBM Cloud Pak for Multicloud Management

This section shows an offline installation of the IBM Cloud Pak for Multicloud Management 3.2.1. Complete the following steps:

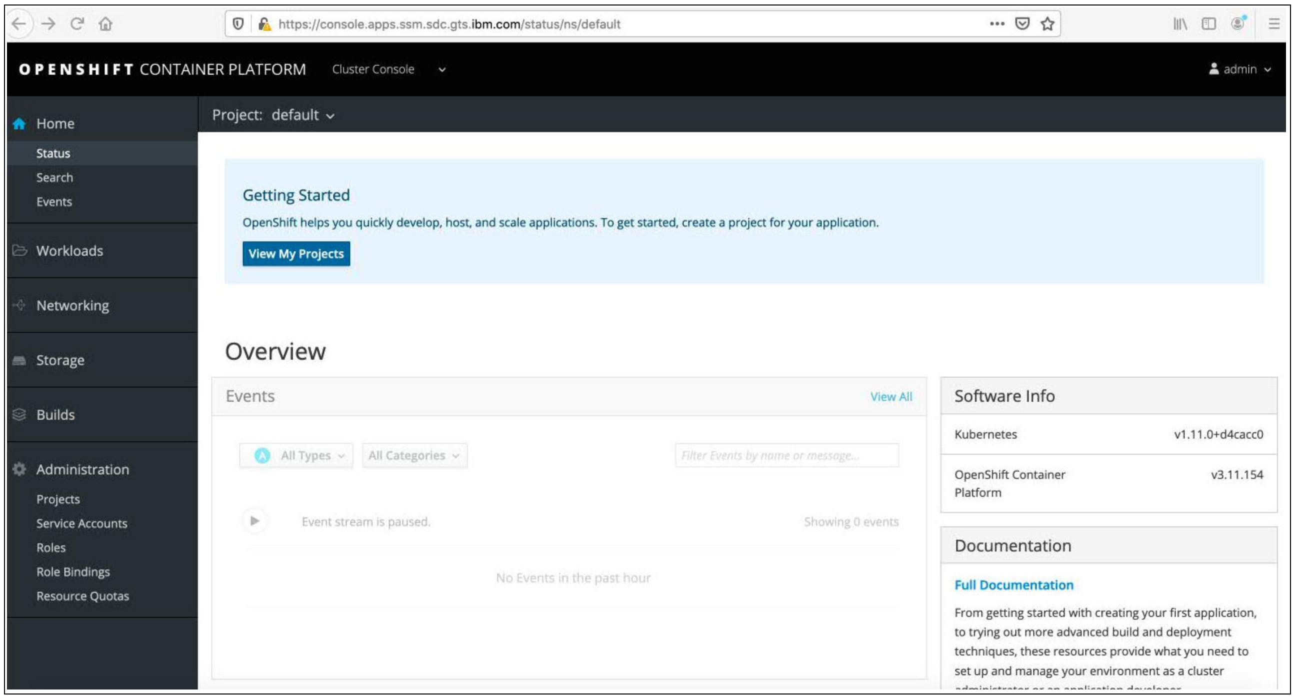
1. Update the firewall to allow HTTP and HTTPS access to the Master VM:

```
firewall-cmd --permanent --add-service=https
success
firewall-cmd --permanent --add-service=http
success
firewall-cmd --reload
success
```

2. Access the OpenShift console to check that the OpenShift cluster is set up correctly. The OpenShift console can access by running the following command:

```
oc -n openshift-console get route
NAME      HOST/PORT                        PATH      SERVICES   PORT     TERMINATION          WILDCARD
console   console.apps.domain.example.com            console    https    reencrypt/Redirect   None
```

The console URL in this example is `console.apps.ssm.sdc.gts.ibm.com`. Open the URL with your browser, as shown in Figure 6-6.



Getting Started

OpenShift helps you quickly develop, host, and scale applications. To get started, create a project for your application.

Event stream is paused.

Showing 0 events

No Events in the past hour

Software Info

Kubernetes	v1.11.0+d4cacc0
OpenShift Container Platform	v3.11.154

Documentation

From getting started with creating your first application, to trying out more advanced build and deployment techniques, these resources provide what you need to set up and manage your environment as a cluster

Figure 6-6 OpenShift Console

You must have a pre-configured StorageClass in OpenShift that can be used for creating storage:

```
oc get sc
NAME                                   PROVISIONER                          AGE
ibm-powervc-k8s-volume-default (default)   ibm/powervc-k8s-volume-provisioner   2h
```

3. Expose the OpenShift image registry with a route:

```
oc get route -n default
NAME             HOST/PORT                                         PATH      SERVICES          PORT
TERMINATION   WILDCARD
docker-registry    docker-registry-default.apps.domain.example.com            docker-registry    <all>
passthrough   None
registry-console   registry-console-default.apps.domain.example.com           registry-console   <all>
passthrough   None
```

4. For Elasticsearch, ensure that the `vm.max_map_count` setting is at least `262144` on all nodes. Run the following command to check:

```
sysctl  vm.max_map_count
vm.max_map_count = 262144
```

5. For the metrics server, check that the OpenShift Container Platform metrics server was installed before you install Cloud Pak Foundation. You can verify that the server was installed correctly by running:

```
oc adm top node
NAME                            CPU(cores)   CPU%      MEMORY(bytes)   MEMORY%
mstnode01.domain.example.com   579m         0%        4742Mi          15%
wrknode01.domain.example.com   85m          0%        1530Mi          2%
wrknode02.domain.example.com   147m         0%        1533Mi          2%
```

6. Check that the admission *w*ebhooks are enabled on the OpenShift Container Platform all master nodes:

```
cp /etc/origin/master/master-config.yaml
/etc/origin/master/master-config.yaml.orig
vi /etc/origin/master/master-config.yaml
admissionConfig:
   pluginConfig:
    MutatingAdmissionWebhook:
      configuration:
        apiVersion: apiserver.config.k8s.io/v1alpha1
        kubeConfigFile: /dev/null
        kind: WebhookAdmission
    ValidatingAdmissionWebhook:
      configuration:
        apiVersion: apiserver.config.k8s.io/v1alpha1
        kubeConfigFile: /dev/null
        kind: WebhookAdmission

### Restart your apiserver and controllers
/usr/local/bin/master-restart api
2
/usr/local/bin/master-restart controllers
2
```

7. Download the installation file (`ibm-cloud-private-ppc64le-3.2.1.tar.gz`) to the master node. Then, load the container images into the local registry:

```
tar xf ibm-cloud-private-ppc64le-3.2.1.tar.gz -O | sudo docker load
22f2e14aaf41: Loading layer [==================================================>] 279.8 MB/279.8 MB
4153cc3de084: Loading layer [==================================================>] 20.48 kB/20.48 kB
ceaa1e685ae1: Loading layer [==================================================>] 21.49 MB/21.49 MB
e5c43e70143d: Loading layer [==================================================>] 170.9 MB/170.9 MB
35f172df1a41: Loading layer [==================================================>] 23.95 MB/23.95 MB
bde32f0ec81a: Loading layer [==================================================>]  34.3 kB/34.3 kB
af4acfce5260: Loading layer [==================================================>] 100.7 MB/100.7 MB
500de4764474: Loading layer [==================================================>] 28.38 MB/28.38 MB
...
Output truncated
...
3f2ac47d1346: Loading layer [==================================================>] 611.3 kB/611.3 kB
e9fd350aaef5: Loading layer [==================================================>]  37.7 MB/37.7 MB
8b58cfdade26: Loading layer [==================================================>]  5.12 kB/5.12 kB
2aa89bbaaba4: Loading layer [==================================================>] 31.39 MB/31.39 MB
dc0532bb043a: Loading layer [==================================================>] 19.74 MB/19.74 MB
93379faa8e65: Loading layer [==================================================>] 3.072 kB/3.072 kB
ca43800e3ee2: Loading layer [==================================================>] 3.072 kB/3.072 kB
Loaded image: ibmcom/nginx-ingress-controller-ppc64le:0.23.1
```

8. Because of a limitation with OpenShift, to deploy IBM Multicloud Manager on the OpenShift Master-Infrastructure node, you must label the node as an OpenShift compute node by using the following command:

```
oc label node mstnode01.domain.example.com node-role.kubernetes.io/compute=true
```

9. Create an installation directory on the boot node:

```
mkdir /opt/ibm-multicloud-manager-3.2.1; cd /opt/ibm-multicloud-manager-3.2.1
```

10. Extract the cluster directory:

```
docker run --rm -v $(pwd):/data:z -e LICENSE=accept --security-opt
label:disable ibmcom/icp-inception-ppc64le:3.2.1-ee cp -r cluster /data
```

11. Copy the OpenShift `admin.kubeconfig` file to the cluster directory. The OpenShift `admin.kubeconfig` file can be found in the **`/etc/origin/master/admin.kubeconfig`** directory:

```
cp /etc/origin/master/admin.kubeconfig
/opt/ibm-multicloud-manager-3.2.1/cluster/kubeconfig
```

12. Use the `power.openshift.config.yaml` file to replace the `config.yaml` for Linux on Power (ppc64le) before you deploy the services:

```
cd /opt/ibm-multicloud-manager-3.2.1/cluster/
echo y |cp power.openshift.config.yaml config.yaml
```

13. Update the `config.yaml` file:

```
# A list of OpenShift nodes that used to run ICP components
cluster_nodes:
  master:
    - mstnode01.domain.example.com
  proxy:
    - mstnode01.domain.example.com
  management:
    - mstnode01.domain.example.com
  va:
    - mstnode01.domain.example.com

storage_class: ibm-powervc-k8s-volume-default

default_admin_password: <password>

password_rules:
    - '(.*)'

ingress_http_port: 3080
ingress_https_port: 3443
```

14. The installer fails trying to create `multicluster-hub-security-advisor-onboarding-xxxx` on the worker node. To prevent this failure, mark the worker Nodes as unschedulable:

```
oc  get events --all-namespaces | grep multicluster-hub-security-advisor
kube-system                      39m       39m       1
multicluster-hub-security-advisor-onboarding-pndbt.15db965a517cb822  Pod
Normal    Scheduled                default-scheduler
Successfully assigned kube-system/multicluster-hub-security-advisor-onboarding-pndbt to
wrknode02.domain.example.com
kube-system                      39m       39m       1
multicluster-hub-security-advisor-onboarding.15db965a5137ad18     Job
Normal    SuccessfulCreate         job-controller
Created pod: multicluster-hub-security-advisor-onboarding-pndbt
kube-system                      38m       38m       6
multicluster-hub-security-advisor-onboarding-pndbt.15db965ef3c989bd   Pod
spec.containers{multicluster-hub-security-advisor-onboarding}   Warning   Failed                   kubelet,
wrknode02.domain.example.com                                    Error: secrets "icp-serviceid-apikey-secret" not
found
oc get nodes
NAME                             STATUS    ROLES                  AGE       VERSION
mstnode01.domain.example.com   Ready     compute,infra,master   13h
v1.11.0+d4cacc0
wrknode01.domain.example.com   Ready     compute                13h
v1.11.0+d4cacc0
wrknode02.domain.example.com   Ready     compute                13h
v1.11.0+d4cacc0

oc adm manage-node  wrknode01.domain.example.com  --schedulable=false
NAME                             STATUS                     ROLES     AGE       VERSION
wrknode01.domain.example.com   Ready,SchedulingDisabled   compute   2h
v1.11.0+d4cacc0
oc adm manage-node  wrknode02.domain.example.com  --schedulable=false
NAME                             STATUS                     ROLES     AGE       VERSION
wrknode02.domain.example.com   Ready,SchedulingDisabled   compute   2h
v1.11.0+d4cacc0
```

15. Deploy IBM Multicloud Manager:

```
docker run -t --net=host -e LICENSE=accept -v $(pwd):/installer/cluster:z -v
/var/run:/var/run:z -v /etc/docker:/etc/docker:z --security-opt label:disable
ibmcom/icp-inception-ppc64le:3.2.1-ee install-with-openshift
PLAY [Deploying addon services] ******************************************************

TASK [Gathering Facts] ***************************************************************
ok: [localhost]

TASK [kubectl-config : Configuring kubectl]
**************************************************************************************
changed: [localhost -> localhost]

TASK [k8s-detection : Detecting Kubernetes type]
**************************************************************************************
changed: [localhost]

TASK [k8s-detection : Detect cluster vendor]
**************************************************************************************
changed: [localhost]

TASK [k8s-detection : Setting Kubernetes type] ***************************************
ok: [localhost]

TASK [check-password : Checking if setting password or not] **************************
skipping: [localhost]

TASK [check-password : Creating check password script] *******************************
changed: [localhost]
...
Output truncated
...
TASK [icam-workaround : Create ingress secret for icam] ******************************
skipping: [localhost]

TASK [openshift-archive-addon : include_tasks] ***************************************

PLAY RECAP
**************************************************************************************
1.1.1.1                    : ok=0    changed=0    unreachable=0    failed=0
localhost                  : ok=446  changed=288  unreachable=0    failed=0


POST DEPLOY MESSAGE
**************************************************************************************

The Dashboard URL: https://icp-console.apps.domain.example.com:443, please use
credentials in config.yaml to login.

Playbook run took 0 days, 1 hours, 13 minutes, 40 seconds
```

16. If you encounter errors during installation, uninstall it by running the following command:

```
docker run -t --net=host -e LICENSE=accept -v $(pwd):/installer/cluster:z -v
/var/run:/var/run:z -v /etc/docker:/etc/docker:z --security-opt label:disable
ibmcom/icp-inception-ppc64le:3.2.1-ee uninstall-with-openshift
```

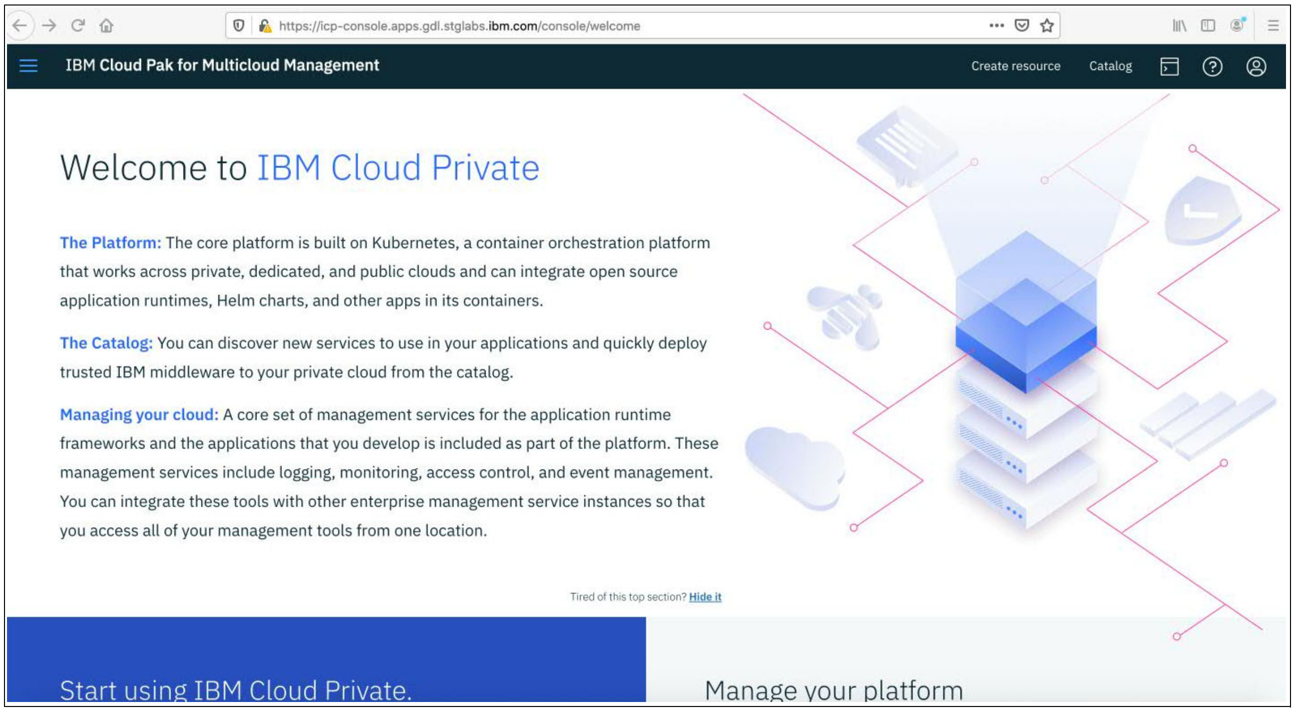
17. Mark the worker Nodes as schedulable:

```
oc adm manage-node  wrknode01.domain.example.com  --schedulable=true
NAME                            STATUS     ROLES     AGE       VERSION
wrknode01.domain.example.com   Ready      compute   3h        v1.11.0+d4cacc0
oc adm manage-node  wrknode02.domain.example.com  --schedulable=true
NAME                            STATUS     ROLES     AGE       VERSION
wrknode02.domain.example.com   Ready      compute   3h        v1.11.0+d4cacc0
```

18. Remove the compute label from the OpenShift Master-Infrastructure node:

```
oc label node mstnode01.domain.example.com
node-role.kubernetes.io/compute=false --overwrite
node/mstnode01.domain.example.com labeled
```

19. Use the URL (*htt*ps://icp-console.apps.domain.example.com:443) to connect to the IBM Multicloud Manager management console, as shown in Figure 6-7.


https://icp-console.apps.gdl.stglabs.ibm.com/console/welcome
IBM Cloud Pak for Multicloud Management
Create resource
Catalog

Welcome to IBM Cloud Private

The Platform: The core platform is built on Kubernetes, a container orchestration platform that works across private, dedicated, and public clouds and can integrate open source application runtimes, Helm charts, and other apps in its containers.

The Catalog: You can discover new services to use in your applications and quickly deploy trusted IBM middleware to your private cloud from the catalog.

Managing your cloud: A core set of management services for the application runtime frameworks and the applications that you develop is included as part of the platform. These management services include logging, monitoring, access control, and event management. You can integrate these tools with other enterprise management service instances so that you access all of your management tools from one location.

Tired of this top section? Hide it

Start using IBM Cloud Private.

Manage your platform

Figure 6-7 IBM Cloud Private console

For more information about IBM Cloud Pak for Multicloud Management installation, see IBM Knowledge Center.

Part 3

Practical scenarios

This part builds on the previous sections by describing some of the typical next steps and use cases. It also provides an overview of some specific deployment scenarios.

The following chapters are included in this part:

7

Use cases

This chapter describes a few use cases. The first use case shows a node.js application build that is run in a container by using OpenShift, and as a backend the application uses MongoDB 3.6.

The second case shows a scenario that can be used for migration in a multicloud environment in particular from one OpenShift environment to another on other site, and it can be used even for failover purposes. This use case also shows running the same backend database on different architectures.

This chapter includes the following topics:

7.1 Building Cloud Native Applications on IBM Power Systems: Rapid development of new applications

The Red Hat OpenShift Container Platform gives developers a self-service platform on which to build and run containerized applications. With Red Hat OpenShift, you can quickly start creating cloud-native applications or cloud-enabling existing applications, and spawning an environment for a new microservice in minutes.

This section deploys a sample Node.js Application from github (`https://github.com/sclorg/nodejs-ex`) to count the page views, and a MongoDB database from the docker registry (`registry.access.redhat.com/rhscl/mongodb-36-rhel7`) to store the views count.

To run the example, complete the following steps:

1. Log in as developer, as shown in Example 7-1.

Example 7-1 Login as developer

```
# oc login -u developer
Authentication required for https://ocp.domain.example.com:8443 (openshift)
Username: developer
Password:
Login successful.

You don't have any projects. You can try to create a new project, by running

    oc new-project <projectname>
```

2. Create the nodejs-example project, as shown in Example 7-2.

Example 7-2 Create nodejs-example project

```
# oc new-project nodejs-example --display-name="nodejs" --description="Sample
Node.js Application"
Now using project "nodejs-example" on server "https://ocp.domain.example.com:8443".

You can add applications to this project with the 'new-app' command. For example, try:

    oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.
# oc project nodejs-example
Already on project "nodejs-example" on server "https://ocp.domain.example.com:8443".
```

3. Deploy the nodejs-ex application from the source code, as shown in Example 7-3.

Example 7-3 deploy nodejs-ex application

```
# oc new-app https://github.com/sclorg/nodejs-ex -l name=myapp
--> Found image e19be86 (2 months old) in image stream "openshift/nodejs" under tag "10"
for "nodejs"

    Node.js 10.16.3
    ---------------
    Node.js 10.16.3 available as a container is a base platform for building and running
various Node.js 10.16.3 applications and frameworks. Node.js is a platform built on
Chrome's JavaScript runtime for easily building fast, scalable network applications.
Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and
```

```
efficient, perfect for data-intensive real-time applications that run across distributed
devices.

    Tags: builder, nodejs, nodejs-10.16.3

    * The source repository appears to match: nodejs
    * A source build using source code from https://github.com/sclorg/nodejs-ex will be
created
      * The resulting image will be pushed to image stream tag "nodejs-ex:latest"
      * Use 'start-build' to trigger a new build
    * This image will be deployed in deployment config "nodejs-ex"
    * Port 8080/tcp will be load balanced by service "nodejs-ex"
      * Other containers can access this service through the hostname "nodejs-ex"

--> Creating resources with label name=myapp ...
    imagestream.image.openshift.io "nodejs-ex" created
    buildconfig.build.openshift.io "nodejs-ex" created
    deploymentconfig.apps.openshift.io "nodejs-ex" created
    service "nodejs-ex" created
--> Success
    Build scheduled, use 'oc logs -f bc/nodejs-ex' to track its progress.
    Application is not exposed. You can expose services to the outside world by executing
one or more of the commands below:
     'oc expose svc/nodejs-ex'
    Run 'oc status' to view your app.
```

4. Run the **`oc status`** command to verify the status of the project, as shown in Example 7-4.

Example 7-4 Run oc status command

```
# oc status
In project nodejs (nodejs-example) on server https://ocp.domain.example.com:8443

svc/nodejs-ex - 172.30.186.102:8080
  dc/nodejs-ex deploys istag/nodejs-ex:latest <-
    bc/nodejs-ex source builds https://github.com/sclorg/nodejs-ex on openshift/nodejs:10
    deployment #1 pending 6 seconds ago

3 infos identified, use 'oc status --suggest' to see details.
```

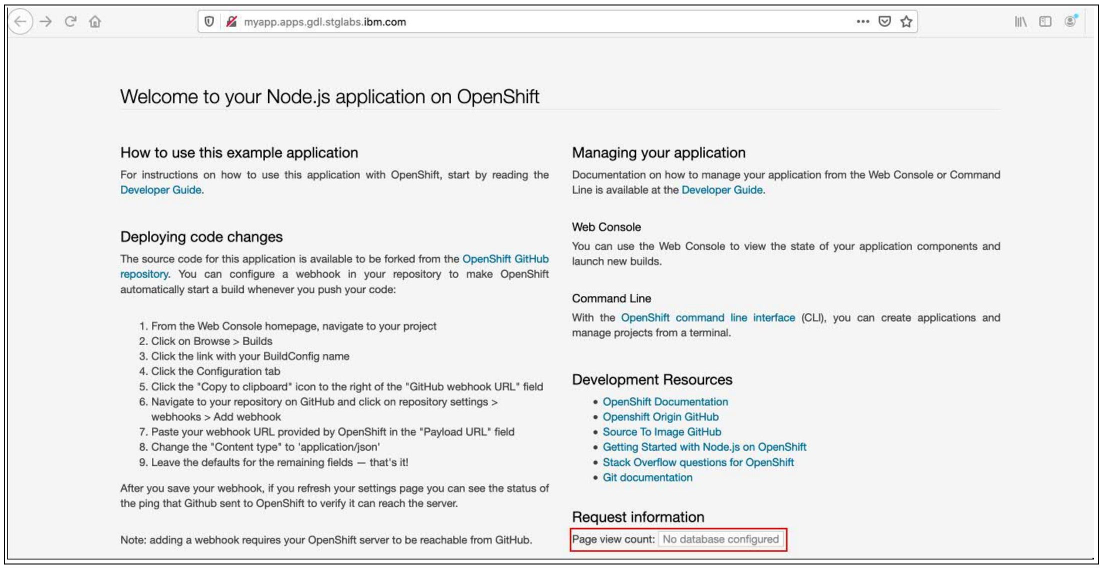
5. Expose the nodejs-ex service, as shown in Example 7-5.

Example 7-5 Expose nodejs-ex

```
# oc get svc
NAME        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
nodejs-ex   ClusterIP   172.30.186.102   <none>        8080/TCP   7m
# oc expose svc/nodejs-ex --hostname=myapp.apps.domain.example.com
route.route.openshift.io/nodejs-ex exposed
# oc get routes
NAME        HOST/PORT PATH      SERVICES    PORT       TERMINATION   WILDCARD
nodejs-ex   myapp.apps.domain.example.com nodejs-ex   8080-tcp                 None
```

Note: The **`oc expose`** command supports generating only unsecured routes. For generating secured (edge, pass-through, re-encrypt) routes, use the **`oc create route`** command.

When you access the application link `http://myapp.apps.domain.example.com`, as shown in Figure 7-1, notice the index page Page view count reads “No database configured”. To fix this issue, add a MongoDB service.



myapp.apps.gdl.stglabs.ibm.com

Welcome to your Node.js application on OpenShift

How to use this example application

For instructions on how to use this application with OpenShift, start by reading the Developer Guide.

Deploying code changes

The source code for this application is available to be forked from the OpenShift GitHub repository. You can configure a webhook in your repository to make OpenShift automatically start a build whenever you push your code:

1. From the Web Console homepage, navigate to your project
2. Click on Browse > Builds
3. Click the link with your BuildConfig name
4. Click the Configuration tab
5. Click the "Copy to clipboard" icon to the right of the "GitHub webhook URL" field
6. Navigate to your repository on GitHub and click on repository settings > webhooks > Add webhook
7. Paste your webhook URL provided by OpenShift in the "Payload URL" field
8. Change the "Content type" to 'application/json'
9. Leave the defaults for the remaining fields — that's it!

After you save your webhook, if you refresh your settings page you can see the status of the ping that Github sent to OpenShift to verify it can reach the server.

Note: adding a webhook requires your OpenShift server to be reachable from GitHub.

Managing your application

Documentation on how to manage your application from the Web Console or Command Line is available at the Developer Guide.

Web Console

You can use the Web Console to view the state of your application components and launch new builds.

Command Line

With the OpenShift command line interface (CLI), you can create applications and manage projects from a terminal.

Development Resources

- OpenShift Documentation
- Openshift Origin GitHub
- Source To Image GitHub
- Getting Started with Node.js on OpenShift
- Stack Overflow questions for OpenShift
- Git documentation

Request information

Page view count: No database configured

Figure 7-1 http://myapp.apps.domain.example.com without MongoDB

6. Deploy the mongodb-36-rhel7 application from the Red Hat registry, as shown in Example 7-6.

Example 7-6 Deploy mongodb application

```
# oc new-app \
    -e MONGODB_USER=admin \
    -e MONGODB_PASSWORD=secret \
    -e MONGODB_DATABASE=mongo_db\
    -e MONGODB_ADMIN_PASSWORD=super-secret \
    registry.access.redhat.com/rhscl/mongodb-36-rhel7
--> Found Docker image 3414ee2 (4 weeks old) from registry.access.redhat.com for 
"registry.access.redhat.com/rhscl/mongodb-36-rhel7"

    MongoDB 3.6
    -----------
    MongoDB (from humongous) is a free and open-source cross-platform document-oriented 
database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents 
with schemas. This container image contains programs to run mongod server.

    Tags: database, mongodb, rh-mongodb36

    * An image stream tag will be created as "mongodb-36-rhel7:latest" that will track this 
image
    * This image will be deployed in deployment config "mongodb-36-rhel7"
    * Port 27017/tcp will be load balanced by service "mongodb-36-rhel7"
      * Other containers can access this service through the hostname "mongodb-36-rhel7"
    * This image declares volumes and will default to use non-persistent, host-local 
storage.
      You can add persistent volumes later by running 'volume dc/mongodb-36-rhel7 --add 
...'

--> Creating resources ...
    imagestream.image.openshift.io "mongodb-36-rhel7" created
    deploymentconfig.apps.openshift.io "mongodb-36-rhel7" created
    service "mongodb-36-rhel7" created
--> Success
    Application is not exposed. You can expose services to the outside world by executing
one or more of the commands below:
     'oc expose svc/mongodb-36-rhel7'
    Run 'oc status' to view your app.
```

7. Configure the persistent volume for MongoDB, as shown in Example 7-7.

Example 7-7 MongoDB persistent volume

```
# cat > mongodb-pvc.yml <<EOF_/mongodb-pvc.yml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mongodb-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 17Gi
EOF_/mongodb-pvc.yml
# oc get dc
NAME               REVISION   DESIRED   CURRENT   TRIGGERED BY
mongodb-36-rhel7   1          1         1         config,image(mongodb-36-rhel7:latest)
nodejs-ex          1          1         1         config,image(nodejs-ex:latest)
# oc set volume dc/mongodb-36-rhel7
deploymentconfigs/mongodb-36-rhel7
  empty directory as mongodb-36-rhel7-volume-1
    mounted at /var/lib/mongodb/data
# oc create -f mongodb-pvc.yml
persistentvolumeclaim/mongodb-pvc created
# oc get pvc
NAME          STATUS    VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS                     AGE
mongodb-pvc   Bound     pvc-424befe9-0403-11ea-b8f2-fa47ee288120   17Gi       RWO
ibm-powervc-k8s-volume-default   1m
# oc set volume dc/mongodb-36-rhel7 --add --name=mongodb-storage -t pvc
--claim-name=mongodb-pvc  --overwrite
warning: volume "mongodb-storage" did not previously exist and was not overriden. A new
volume with this name has been created
instead.deploymentconfig.apps.openshift.io/mongodb-36-rhel7 volume updated
# oc set volume dc/mongodb-36-rhel7
deploymentconfigs/mongodb-36-rhel7
  empty directory as mongodb-36-rhel7-volume-1
    mounted at /var/lib/mongodb/data
  pvc/mongodb-pvc (allocated 17GiB) as mongodb-storage
```

8. Set up the environment variables for the nodejs-ex application, as shown in Example 7-8.

Example 7-8 Setting environment variables

```
# oc status
In project nodejs (nodejs-example) on server https://ocp.domain.example.com:8443

svc/mongodb-36-rhel7 - 172.30.155.22:27017
  dc/mongodb-36-rhel7 deploys istag/mongodb-36-rhel7:latest
```

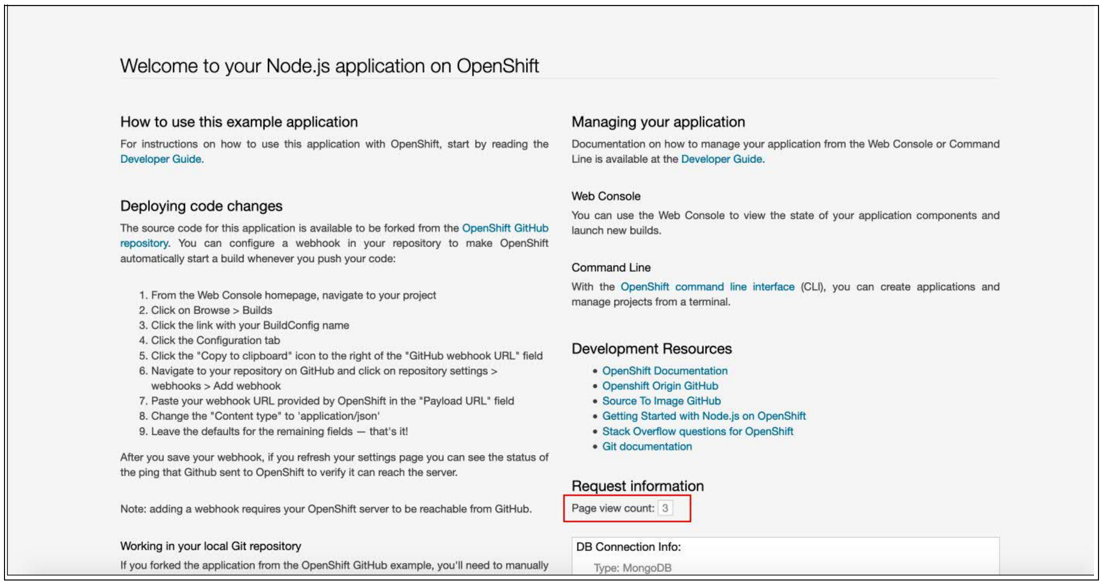
```
    deployment #2 deployed 5 minutes ago - 1 pod
    deployment #1 deployed 13 minutes ago

http://myapp.apps.domain.example.com to pod port 8080-tcp (svc/nodejs-ex)
  dc/nodejs-ex deploys istag/nodejs-ex:latest <-
    bc/nodejs-ex source builds https://github.com/sclorg/nodejs-ex on openshift/nodejs:10
    deployment #1 deployed about an hour ago - 1 pod
# oc set env dc/nodejs-ex
MONGO_URL='mongodb://admin:secret@172.30.155.22:27017/mongo_db'
deploymentconfig.apps.openshift.io/nodejs-ex updated
# oc status
In project nodejs (nodejs-example) on server https://ocp.domain.example.com:8443

svc/mongodb-36-rhel7 - 172.30.155.22:27017
  dc/mongodb-36-rhel7 deploys istag/mongodb-36-rhel7:latest
    deployment #2 deployed 7 minutes ago - 1 pod
    deployment #1 deployed 15 minutes ago

http://myapp.apps.domain.example.com to pod port 8080-tcp (svc/nodejs-ex)
  dc/nodejs-ex deploys istag/nodejs-ex:latest <-
    bc/nodejs-ex source builds https://github.com/sclorg/nodejs-ex on openshift/nodejs:10
    deployment #2 deployed about a minute ago - 1 pod
    deployment #1 deployed about an hour ago
```

9. Check the Page view count by accessing the application link `http://myapp.apps.domain.example.com`, as shown in Figure 7-2.



Welcome to your Node.js application on OpenShift

How to use this example application

For instructions on how to use this application with OpenShift, start by reading the Developer Guide.

Deploying code changes

The source code for this application is available to be forked from the OpenShift GitHub repository. You can configure a webhook in your repository to make OpenShift automatically start a build whenever you push your code:

1. From the Web Console homepage, navigate to your project
2. Click on Browse > Builds
3. Click the link with your BuildConfig name
4. Click the Configuration tab
5. Click the "Copy to clipboard" icon to the right of the "GitHub webhook URL" field
6. Navigate to your repository on GitHub and click on repository settings > webhooks > Add webhook
7. Paste your webhook URL provided by OpenShift in the "Payload URL" field
8. Change the "Content type" to 'application/json'
9. Leave the defaults for the remaining fields — that's it!

After you save your webhook, if you refresh your settings page you can see the status of the ping that Github sent to OpenShift to verify it can reach the server.

Note: adding a webhook requires your OpenShift server to be reachable from GitHub.

Working in your local Git repository

If you forked the application from the OpenShift GitHub example, you'll need to manually

Managing your application

Documentation on how to manage your application from the Web Console or Command Line is available at the Developer Guide.

Web Console

You can use the Web Console to view the state of your application components and launch new builds.

Command Line

With the OpenShift command line interface (CLI), you can create applications and manage projects from a terminal.

Development Resources

- OpenShift Documentation
- Openshift Origin GitHub
- Source To Image GitHub
- Getting Started with Node.js on OpenShift
- Stack Overflow questions for OpenShift
- Git documentation

Request information

Page view count: 3

DB Connection Info:

Type: MongoDB

Figure 7-2 http://myapp.apps.domain.example.com with MongoDB

10. Scale the nodejs-ex application to three nodes, as shown in Example 7-9.

Example 7-9 Scale nodejs-ex application

```
# oc get dc
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
mongodb-36-rhel7  2          1         1         config,image(mongodb-36-rhel7:latest)
nodejs-ex         2          1         1         config,image(nodejs-ex:latest)
# oc scale dc/nodejs-ex --replicas=3
deploymentconfig.apps.openshift.io/nodejs-ex scaled
```

```
# oc get dc
NAME              REVISION   DESIRED   CURRENT   TRIGGERED BY
mongodb-36-rhel7  2          1         1         config,image(mongodb-36-rhel7:latest)
nodejs-ex         2          3         3         config,image(nodejs-ex:latest)
```

7.2 Hybrid architecture and multicloud applications: A true hybrid multicloud feel for the user

For containers, different architectures do not mean different types of deployments, even in a multicloud environment with different infrastructures. This section gives an example of how two different environments operate on different infrastructures, architectures, and OpenShift versions. Ephemeral containers that do not use persistent volumes are easy to migrate across different Kubernetes environments.

This section also demonstrates migrating data that is on persistent volumes that are used by persistent application containers across environments to show the complete stack of a truly multicloud hybrid architecture.

7.2.1 Multicloud approach by using stateful MongoDB database

MongoDB is a NoSQL database that is being used widely as a backend for many applications. This scenario was selected because it shows the flexibility of working with containers on IBM Power Systems servers while using multicloud functions, even on stateful applications.

The intent is to show with simple Yet Another Markup Language (yaml) files that the same user experience can be achieved, even by the administrator of the clusters. The yaml files can even be converted in templates later if needed for self service deployment. The user transparent experience is shown that uses MongoDB Compass client. The on-premise test environment is in Guadalajara (Mexico), the AWS zone is in Ohio (USA), and the client is in New York (US).

7.2.2 OpenShift Container Platform 3.11 ppc64le on-premises

For demonstration purposes, the on-premises scenario uses a small Power Systems OpenShift cluster with a single node, as shown in Example 7-10.

Example 7-10 OpenShift 3.11 cluster on a Power Systems node

```
[root@dcocp01 ~]# oc get nodes -o json |grep -A10 "nodeInfo"
                "nodeInfo": {
                    "architecture": "ppc64le",
                    "bootID": "5d13a724-e909-4481-a617-ebd9acf4e020",
                    "containerRuntimeVersion": "docker://1.13.1",
                    "kernelVersion": "4.14.0-115.13.1.el7a.ppc64le",
                    "kubeProxyVersion": "v1.11.0+d4cacc0",
                    "kubeletVersion": "v1.11.0+d4cacc0",
                    "machineID": "28803bf7ab784a0796e22f40f33a2827",
                    "operatingSystem": "linux",
                    "osImage": "OpenShift Enterprise",
                    "systemUUID": "IBM,037892CBA\u0000"
[root@dcocp01 ~]#
```

For a stateful container, you need the PV backing up the PVC requested for the pod. Other scenarios showed the FlexVolume use with PowerVC, but this case uses the NFS attachment for you to see another supported backend for volumes. This type of backend is still available if not in use on any pod. The deployment file is shown in Example 7-11. Also, check the policy of the PV is Retain and not Delete so the PVC can be used, even after migrated to another cloud. This makes it available if and when the container is brought back.

Example 7-11 Yaml file for NFS backed PV on the Power Systems cluster

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: appmongo-ibm-mongodb-dev-datavolume
spec:
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: dcocp1
    path: /nfsmongofs/mongo
```

To verify the claim bounds to the intended PV, leave the PVC created, as shown in Example 7-12.

Example 7-12 Yaml file for NFS backed PVC on the Power Systems cluster

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: "appmongo-ibm-mongodb-dev-datavolume"
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
```

MongoDB needs users and passwords for the database. To confirm that the password is not on the Pod specification, use a secret entry, as shown in Example 7-13. Do not use this hash in your environment because this is a password for demonstration purposes only.

Example 7-13 Yaml file for Secret password safeguard

```
apiVersion: v1
data:
  password: c3RhcnQxMjM0
kind: Secret
metadata:
  labels:
    app: appmongo-ibm-mongodb-dev
  name: appmongo-ibm-mongodb-dev
type: Opaque
```

To generate a hash for your environment, use the **base64** command, as shown in Example 7-14.

Example 7-14 Creating base64 hash for password stored in secrets

```
[root@dcocp01 ~]# echo -n start1234|base64
c3RhcnQxMjM0
[root@dcocp01 ~]#
```

This scenario uses MongoDB 3.6.3 image available at the redhat registry (`registry.access.redhat.com/rhscl/mongodb-36-rhel7`). As shown in Example 7-15, our scenario creates a yaml file that picks any architecture between POWER and x86, but a different weight can be set to give preference to one architecture. This pod definition makes it transparent to the cluster operator if the MongoDB is being deployed on-premise, in the cloud, on POWER, or on x86. The image has a unique name, although a container for each architecture exists and is pulled accordingly without any user input.

Example 7-15 Yaml file for the MongoDB Pod

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    productName: MongoDB
    productVersion: 3.6.3
  labels:
    app: appmongo-ibm-mongodb-dev
  name: appmongo-ibm-mongodb-dev
  namespace: default
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - preference:
          matchExpressions:
          - key: beta.kubernetes.io/arch
            operator: In
            values:
            - amd64
        weight: 2
      - preference:
          matchExpressions:
          - key: beta.kubernetes.io/arch
            operator: In
            values:
            - ppc64le
        weight: 2
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: beta.kubernetes.io/arch
            operator: In
            values:
            - amd64
            - ppc64le
  containers:
```

```
- env:
  - name: LICENSE
    value: accept
  - name: MONGODB_USER
    value: mongo
  - name: MONGODB_PASSWORD
    valueFrom:
      secretKeyRef:
        key: password
        name: appmongo-ibm-mongodb-dev
  - name: MONGODB_ADMIN_PASSWORD
    valueFrom:
      secretKeyRef:
        key: password
        name: appmongo-ibm-mongodb-dev
  - name: MONGODB_DATABASE
    value: admin
  image: registry.access.redhat.com/rhscl/mongodb-36-rhel7
  imagePullPolicy: IfNotPresent
  name: appmongo-ibm-mongodb-dev
  ports:
  - containerPort: 27017
    name: mongodb
    protocol: TCP
  resources:
    limits:
      cpu: "2"
      memory: 4Gi
    requests:
      cpu: 100m
      memory: 256Mi
  securityContext:
    privileged: true
    capabilities:
      drop:
      - KILL
      - MKNOD
      - SETGID
      - SETUID
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  volumeMounts:
  - mountPath: /var/lib/mongodb/data
    name: appmongo-ibm-mongodb-dev-datavolume
dnsPolicy: ClusterFirst
priority: 0
restartPolicy: Always
schedulerName: default-scheduler
securityContext:
  privileged: true
  fsGroup: 10000
  runAsUser: 10000
  seLinuxOptions:
    level: s0:c1,c0
terminationGracePeriodSeconds: 30
```

```
  tolerations:
  - effect: NoSchedule
    key: node.kubernetes.io/memory-pressure
    operator: Exists
  volumes:
  - name: appmongo-ibm-mongodb-dev-datavolume
    persistentVolumeClaim:
      claimName: appmongo-ibm-mongodb-dev-datavolume
```

The yaml file that is shown in Example 7-15 on page 171 uses the user as uid 10000 that refers to the mongo user in our machines. Give access to that user to write files, as shown in Example 7-16.

Example 7-16 Permission for user mongo set on the PV

```
[root@dcocp01 ~]# ls -la /nfsmongofs/mongo
total 8
drwx------. 2 mongo mongo 4096 Nov  9 14:34 .
drwxr-xr-x. 4 root  root  4096 Oct 31 12:28 ..
[root@dcocp01 ~]# id mongo
uid=10000(mongo) gid=10000(mongo) groups=10000(mongo)
[root@dcocp01 ~]#
```

Finally, to verify customers can access mongodb, expose the port by using the NodePort type, as shown in Example 7-17. In our case, we use this type so the service is accessible on the external network. Other means of connection also can be used. MongoDB compass client can even use SSH tunneling.

Example 7-17 Yaml file for service on the Power Systems cluster

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: appmongo-ibm-mongodb-dev
  name: appmongo-ibm-mongodb-dev
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: ibm-mongodb-dev
    nodePort: 32767
    port: 27017
    protocol: TCP
    targetPort: 27017
  selector:
    app: appmongo-ibm-mongodb-dev
  sessionAffinity: None
  type: NodePort
```

Applying all yaml files as shown in Example 7-18 makes MongoDB available for connections on the node IP address at port 32767.

Example 7-18 Applying yaml files on the POWER OpenShift cluster

```
[root@dcocp01 daniel]# oc apply -f nfspv.yaml
persistentvolume/appmongo-ibm-mongodb-dev-datavolume created
```

```
[root@dcocp01 daniel]# oc apply -f nfspvc.yaml
persistentvolumeclaim/appmongo-ibm-mongodb-dev-datavolume created
[root@dcocp01 daniel]# oc apply -f secret.yaml
secret/appmongo-ibm-mongodb-dev created
[root@dcocp01 daniel]# oc apply -f service.yaml
service/appmongo-ibm-mongodb-dev created
[root@dcocp01 daniel]# oc apply -f mongo_pod.yaml
pod/appmongo-ibm-mongodb-dev created
[root@dcocp01 daniel]#
```

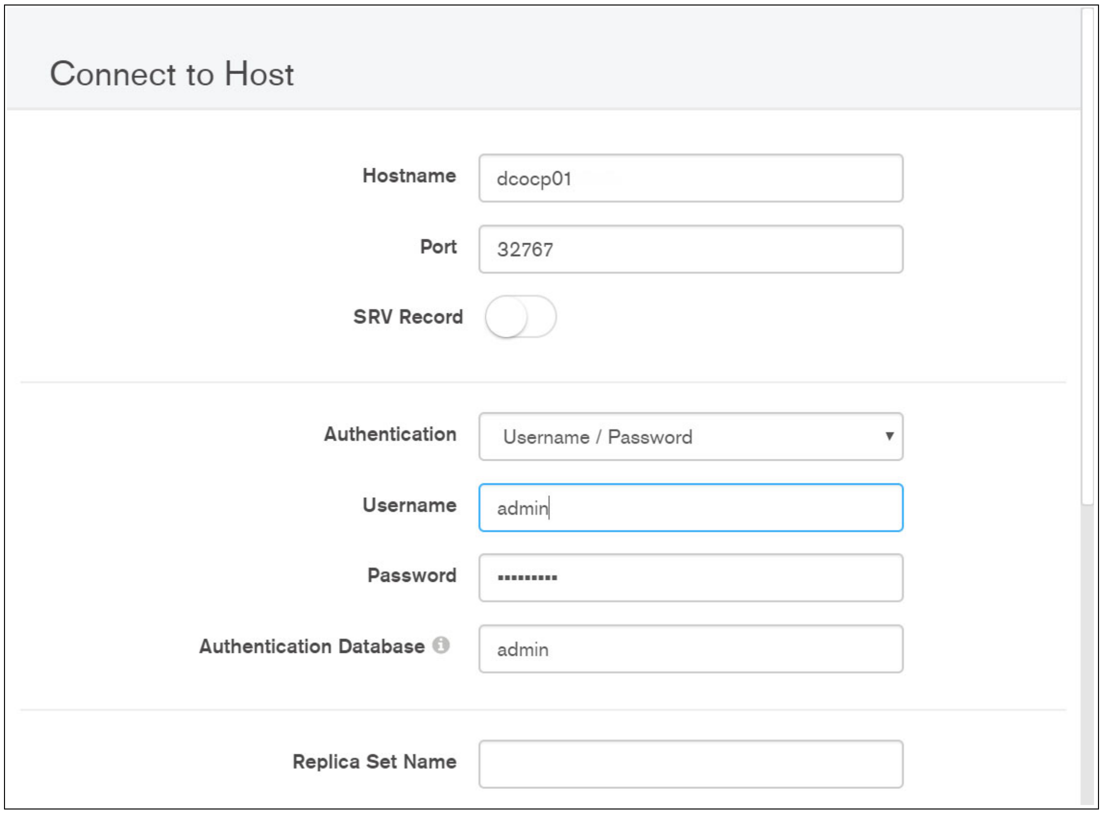
Example 7-19 shows the MongoDB Pod running, and the service correctly listening to the expected port.

Example 7-19 MongoDB Pod running and port 32767 listening for connections

```
[root@dcocp01 daniel]# oc get pod appmongo-ibm-mongodb-dev
NAME                       READY     STATUS    RESTARTS   AGE
appmongo-ibm-mongodb-dev   1/1       Running   0          7m
[root@dcocp01 daniel]# telnet localhost 32767
Trying ::1...
Connected to localhost.
Escape character is '^]'.
^]
telnet> quit
Connection closed.
[root@dcocp01 daniel]#
```

The first time you run the MongoDB with a clean pv, it creates the stateful files on the container path `/var/lib/mongodb/data`.

Using MongoDB Compass v1.16.4, access the database on the IP and port configured on the service, as shown in Figure 7-3. Use the admin user and the password that are configured on the secret file, as shown in Example 7-13 on page 170.

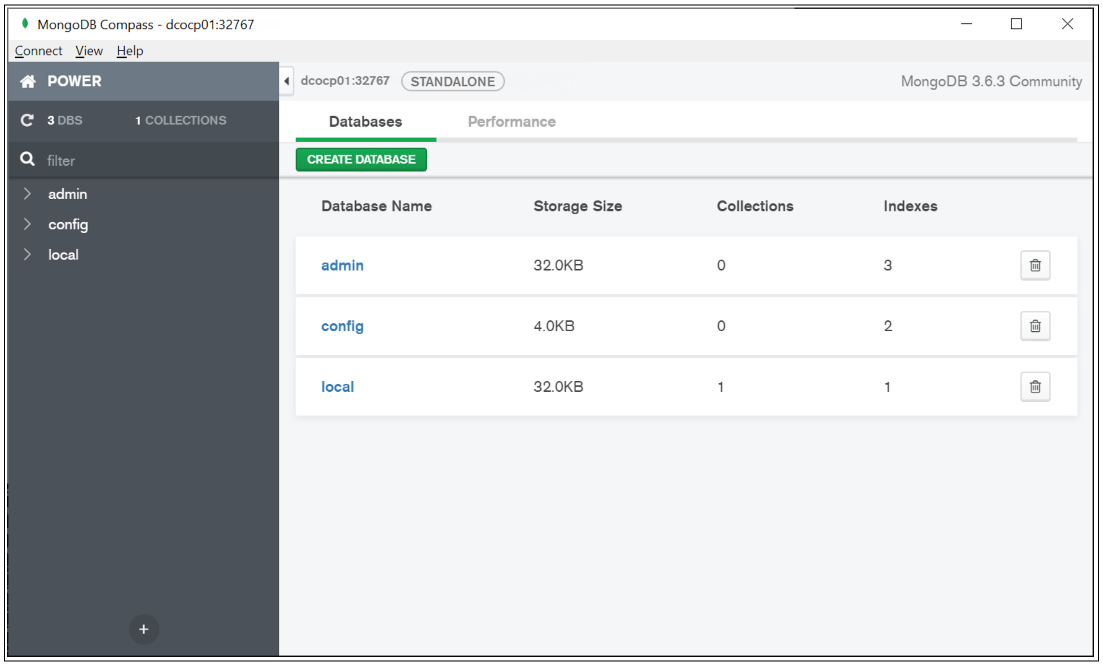

Connect to Host

Hostname: dcocp01
Port: 32767
SRV Record
Authentication: Username / Password
Username: admin
Password: ••••••••••
Authentication Database: admin
Replica Set Name:

Figure 7-3 Connecting to the MongoDB database on the Power Systems cluster

For tests purposes, download the NYC Restaurant data set that is available at the city page at this web page.

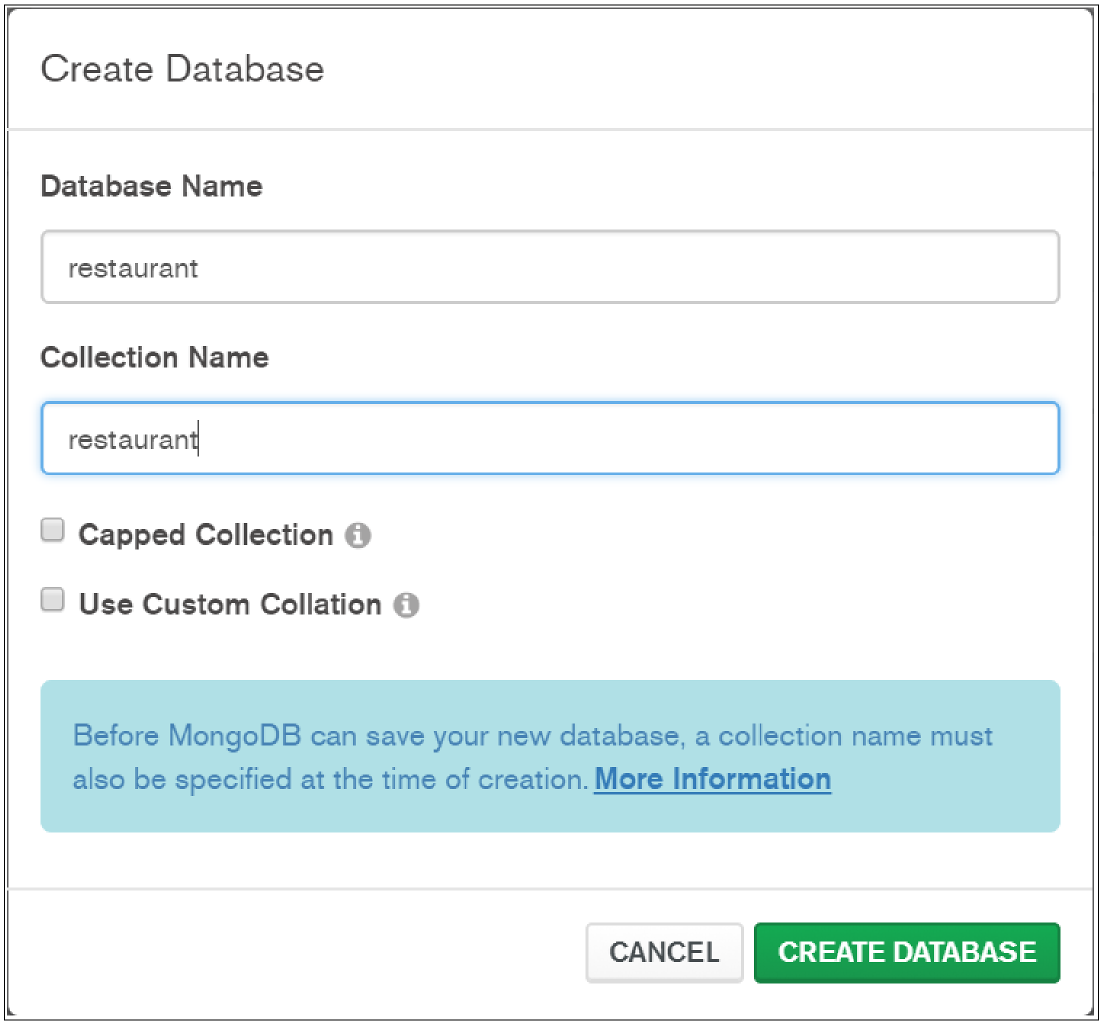
Export the data by using the CSV format to import it on MongoDB compass. The first time you open MongoDB, you see three databases. Click **Create Database** (as shown in Figure 7-4) to insert the downloaded data set as a new collection.



Database Name	Storage Size	Collections	Indexes
admin	32.0KB	0	3
config	4.0KB	0	2
local	32.0KB	1	1

Figure 7-4 Click Create Database on MongoDB Compass

Enter the database name and the collection name, as shown in Figure 7-5. In our case, both are called `restaurant`.


Create Database

Database Name: restaurant

Collection Name: restaurant

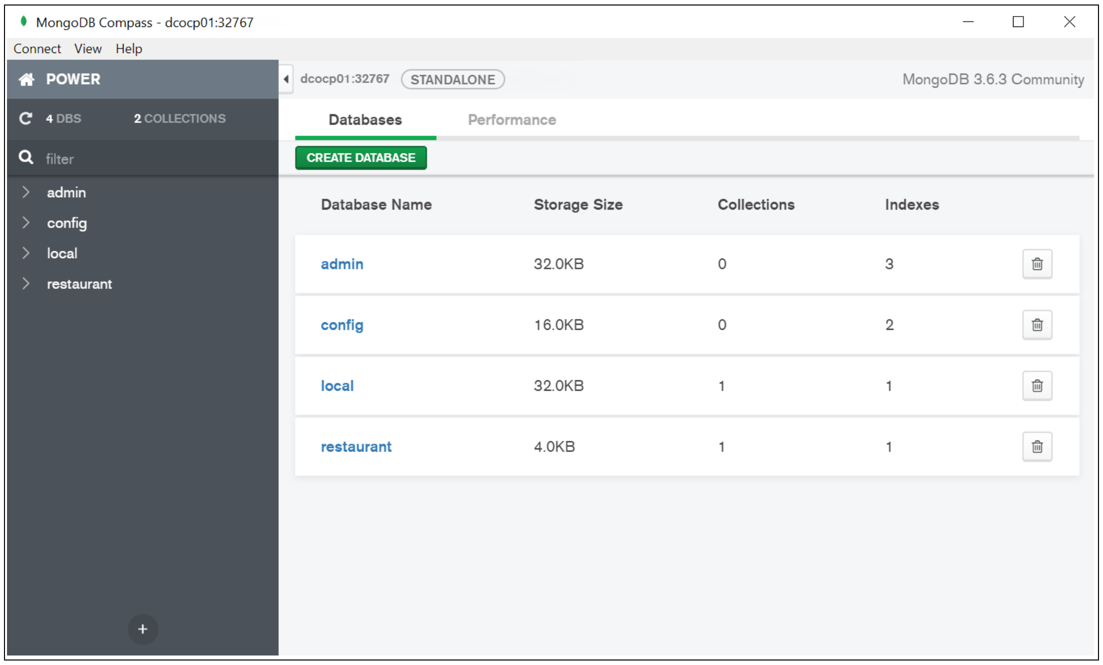
☐ Capped Collection

☐ Use Custom Collation

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. More Information

Figure 7-5 Creating restaurant Database and Collection

The new database appears empty at 4 K, as shown in Figure 7-6.



MongoDB Compass - dcocp01:32767

Database Name	Storage Size	Collections	Indexes
admin	32.0KB	0	3
config	16.0KB	0	2
local	32.0KB	1	1
restaurant	4.0KB	1	1

Figure 7-6 Newly created restaurant database

Enter the collection by clicking the restaurant database, and on the restaurant collection. Figure 7-7 shows the top bar menu changes and a Collection Item appears. Click **Collection** → **Import Data** to import the downloaded data set.

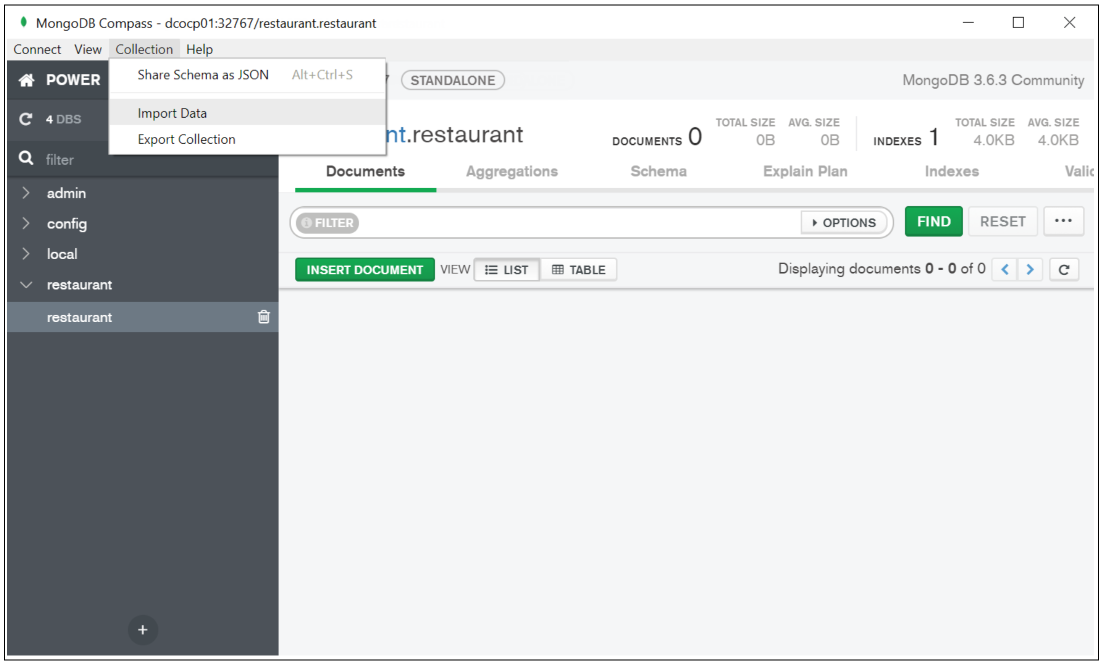
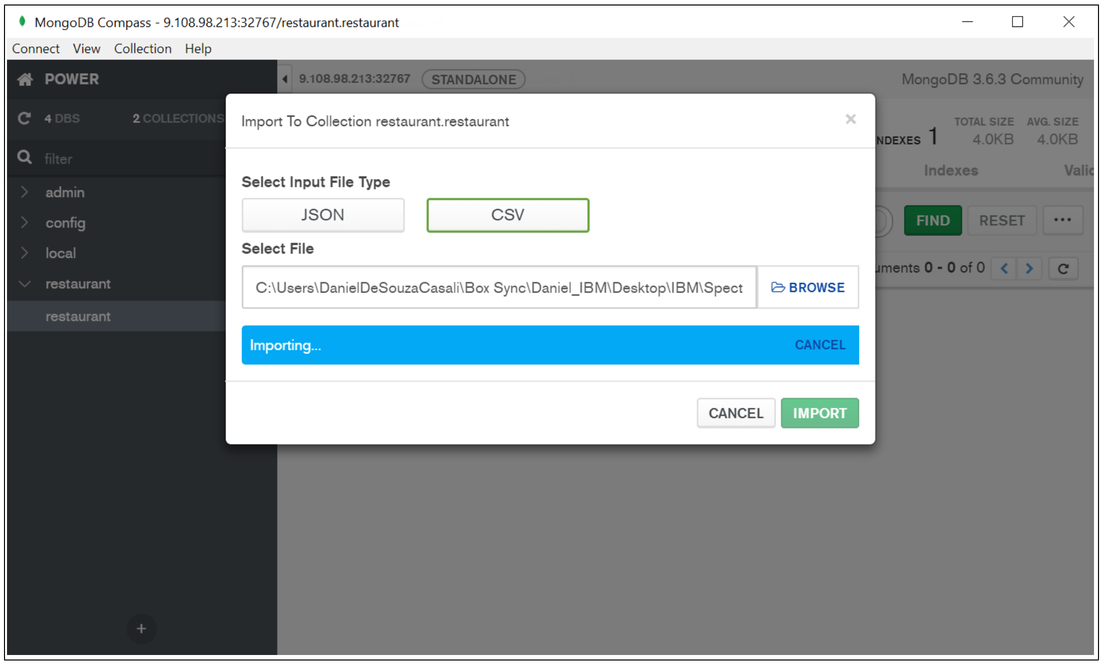


Figure 7-7 Import data into collection menu

You are prompted to insert the file to be imported, as shown in Figure 7-8. Browse for the file, select it, and click **Import**. This scenario uses file type CSV because this format is the downloaded format.



MongoDB Compass - 9.108.98.213:32767/restaurant.restaurant

Import To Collection restaurant.restaurant

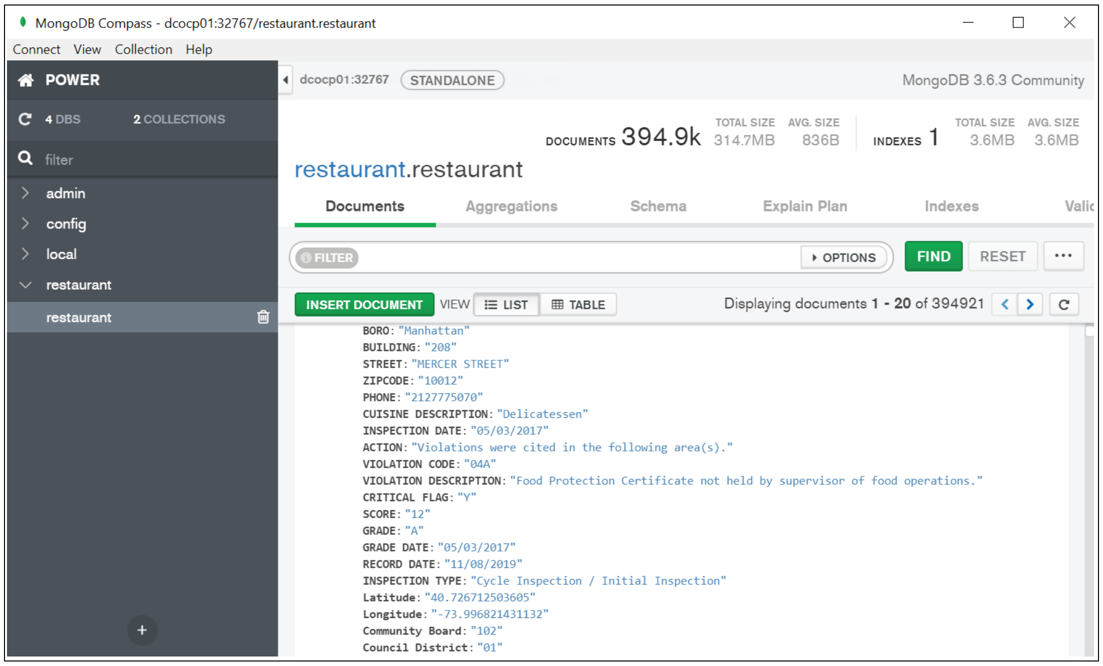
Select Input File Type: JSON [✓ CSV]

Select File: C:\Users\DanielDeSouzaCasali\Box Sync\Daniel_IBM\Desktop\IBM\Spect

Importing...

Figure 7-8 Importing data set into the collection

Figure 7-9 shows almost 400 K documents and a little more than 300 MB database, which serves as background data for the multicloud tests.



MongoDB Compass - dcocp01:32767/restaurant.restaurant

DOCUMENTS 394.9k TOTAL SIZE 314.7MB AVG. SIZE 836B INDEXES 1 TOTAL SIZE 3.6MB AVG. SIZE 3.6MB

restaurant.restaurant

Displaying documents 1 - 20 of 394921

```
BORO: "Manhattan"
BUILDING: "208"
STREET: "MERCER STREET"
ZIPCODE: "10012"
PHONE: "2127775070"
CUISINE DESCRIPTION: "Delicatessen"
INSPECTION DATE: "05/03/2017"
ACTION: "Violations were cited in the following area(s)."
VIOLATION CODE: "04A"
VIOLATION DESCRIPTION: "Food Protection Certificate not held by supervisor of food operations."
CRITICAL FLAG: "Y"
SCORE: "12"
GRADE: "A"
GRADE DATE: "05/03/2017"
RECORD DATE: "11/08/2019"
INSPECTION TYPE: "Cycle Inspection / Initial Inspection"
Latitude: "40.726712503605"
Longitude: "-73.996821431132"
Community Board: "102"
Council District: "01"
```

Figure 7-9 Data imported on the container running on Power Systems OpenShift

Now that the database is running, it contains loaded data in the Power Systems cluster on-premises. Delete the pod and keep the data to use it on the other cluster, as shown in Example 7-20.

Delete the pod so that it no longer runs on-premises, and start the workload on AWS.

Example 7-20 Deleting Pod and maintaining the data

```
[root@dcocp01 ~]# oc get pods
NAME                       READY     STATUS    RESTARTS   AGE
appmongo-ibm-mongodb-dev   1/1       Running   0          21h
docker-registry-2-7dxwx    1/1       Running   1          13d
router-1-g9btn             1/1       Running   1          13d
[root@dcocp01 ~]# oc delete pod appmongo-ibm-mongodb-dev
pod "appmongo-ibm-mongodb-dev" deleted
[root@dcocp01 ~]#ls -la /nfsmongofs/mongo/
total 146028
drwx------. 4 mongo mongo      4096 Nov 10 12:07 .
drwxr-xr-x. 4 root  root       4096 Oct 31 12:28 ..
-rw-------. 1 mongo root      16384 Nov 10 12:07
collection-0--1629751576978663310.wt
-rw-------. 1 mongo root      24576 Nov 10 12:07
collection-0-7744448727083060016.wt
-rw-------. 1 mongo root      32768 Nov 10 12:07
collection-2--1629751576978663310.wt
-rw-------. 1 mongo root  145391616 Nov 10 12:07
collection-3-7744448727083060016.wt
-rw-------. 1 mongo root      16384 Nov 10 12:07
collection-4--1629751576978663310.wt
drwx------. 2 mongo root       4096 Nov 10 12:07 diagnostic.data
-rw-------. 1 mongo root      16384 Nov 10 12:07 index-1--1629751576978663310.wt
-rw-------. 1 mongo root      24576 Nov 10 12:07 index-1-7744448727083060016.wt
-rw-------. 1 mongo root      24576 Nov 10 12:07 index-2-7744448727083060016.wt
-rw-------. 1 mongo root      32768 Nov 10 12:07 index-3--1629751576978663310.wt
-rw-------. 1 mongo root    3735552 Nov 10 12:07 index-4-7744448727083060016.wt
-rw-------. 1 mongo root      16384 Nov  9 14:38 index-5--1629751576978663310.wt
-rw-------. 1 mongo root      16384 Nov 10 12:07 index-6--1629751576978663310.wt
drwx------. 2 mongo root       4096 Nov  9 16:23 journal
-rw-------. 1 mongo root      36864 Nov 10 12:07 _mdb_catalog.wt
-rw-------. 1 mongo root          0 Nov 10 12:07 mongod.lock
-rw-------. 1 mongo root      36864 Nov 10 12:07 sizeStorer.wt
-rw-------. 1 mongo root        114 Nov  9 14:38 storage.bson
-rw-------. 1 mongo root         48 Nov  9 14:38 WiredTiger
-rw-------. 1 mongo root       4096 Nov 10 12:07 WiredTigerLAS.wt
-rw-------. 1 mongo root         21 Nov  9 14:38 WiredTiger.lock
-rw-------. 1 mongo root       1049 Nov 10 12:07 WiredTiger.turtle
-rw-------. 1 mongo root      69632 Nov 10 12:07 WiredTiger.wt
```

7.2.3 OpenShift Container Platform v4.1 on x86 at AWS

OpenShift v4.1 integrates with AWS by using APIs; therefore, the scenario uses this version. However, the steps for installing OpenShift on AWS is out of the scope for this publication. You can see the result of the deployment in Example 7-21.

Example 7-21 Cluster deployed at AWS

```
[root@ip-10-0-2-217 ~]# oc get nodes -o json |grep -A10 "nodeInfo"
                "nodeInfo": {
                    "architecture": "amd64",
                    "bootID": "fa77c564-55a0-41e1-9da2-1eaca280223b",
                    "containerRuntimeVersion":
"cri-o://1.13.11-0.7.dev.rhaos4.1.git9cb8f2f.el8-dev",
                    "kernelVersion": "4.18.0-80.11.2.el8_0.x86_64",
                    "kubeProxyVersion": "v1.13.4+12ee15d4a",
                    "kubeletVersion": "v1.13.4+12ee15d4a",
                    "machineID": "b39138ad5cff4497bdd372953c39d7eb",
                    "operatingSystem": "linux",
                    "osImage": "Red Hat Enterprise Linux CoreOS 410.8.20190920.2
(Ootpa)",
                    "systemUUID": "ec2f1bd5-0c7b-ee9d-2580-9268d105ad71"
--
                "nodeInfo": {
                    "architecture": "amd64",
                    "bootID": "3a3a43b4-aea3-4509-a99f-6e885c5bd70b",
                    "containerRuntimeVersion":
"cri-o://1.13.11-0.11.dev.rhaos4.1.git3338d4d.el7-dev",
                    "kernelVersion": "3.10.0-957.21.3.el7.x86_64",
                    "kubeProxyVersion": "v1.13.4+493dbf621",
                    "kubeletVersion": "v1.13.4+493dbf621",
                    "machineID": "e84c8b64636b4bde8584cf1430e8cb80",
                    "operatingSystem": "linux",
                    "osImage": "OpenShift Enterprise",
                    "systemUUID": "EC28C516-373A-6D9B-C6E1-5DCE702EFC62"
--
                "nodeInfo": {
                    "architecture": "amd64",
                    "bootID": "516efa29-8b96-4527-9080-11f499dff4dc",
                    "containerRuntimeVersion":
"cri-o://1.13.11-0.7.dev.rhaos4.1.git9cb8f2f.el8-dev",
                    "kernelVersion": "4.18.0-80.11.2.el8_0.x86_64",
                    "kubeProxyVersion": "v1.13.4+12ee15d4a",
                    "kubeletVersion": "v1.13.4+12ee15d4a",
                    "machineID": "af6b56b6d5cc44cfbf3c6813422d9ea3",
                    "operatingSystem": "linux",
                    "osImage": "Red Hat Enterprise Linux CoreOS 410.8.20190920.2
(Ootpa)",
                    "systemUUID": "ec2a5444-3f82-5928-1e98-f719eda0b577"
--
                "nodeInfo": {
                    "architecture": "amd64",
                    "bootID": "4eaf7d68-011d-4236-aef0-a253fac84c42",
                    "containerRuntimeVersion":
"cri-o://1.13.11-0.7.dev.rhaos4.1.git9cb8f2f.el8-dev",
                    "kernelVersion": "4.18.0-80.11.2.el8_0.x86_64",
                    "kubeProxyVersion": "v1.13.4+12ee15d4a",
                    "kubeletVersion": "v1.13.4+12ee15d4a",
                    "machineID": "fcc49500c2f440e2950f703d4837df63",
                    "operatingSystem": "linux",
                    "osImage": "Red Hat Enterprise Linux CoreOS 410.8.20190920.2
(Ootpa)",
                    "systemUUID": "ec25be78-a9c2-1a9b-2cdf-3c084618b7b0"
--
                "nodeInfo": {
                    "architecture": "amd64",
                    "bootID": "2ae44826-30f6-4736-b489-e0cc7a697588",
                    "containerRuntimeVersion":
"cri-o://1.13.11-0.11.dev.rhaos4.1.git3338d4d.el7-dev",
                    "kernelVersion": "3.10.0-957.21.3.el7.x86_64",
                    "kubeProxyVersion": "v1.13.4+493dbf621",
                    "kubeletVersion": "v1.13.4+493dbf621",
                    "machineID": "7b71ef5a93d247e998a6da6ea4eb8a3d",
                    "operatingSystem": "linux",
                    "osImage": "OpenShift Enterprise",
                    "systemUUID": "EC2F9F96-B5C2-F6E3-896F-CB35F1D6A06A"
```

Achieving asynchronous data migration where the data can be seamlessly, in both clouds and at the same time, is needed for a complete multicloud environment. This scenario uses IBM Spectrum Scale Active File Management in a technology preview integration with the Spectrum Scale CSI driver. The Active File Management feature creates caches of your data in the cloud, during the time saving on egress charges only writing back the changes made, and asynchronously back on-premises.

The asynchronous data migration is intended to be supported when the support for the GUI on AWS is available, and also the use of PVC on directories inside AFM file sets. The configuration of Spectrum Scale Active File Management is outside the scope of this publication. Example 7-22 shows the creation of the persistent volume by using Spectrum Scale CSI driver.

Example 7-22 CSI persistent volume yaml file

```
apiVersion: v1
kind: PersistentVolume
metadata:
        name: appmongo-ibm-mongodb-dev-datavolume
spec:
  storageClassName: manual
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  csi:
    driver: csi-spectrum-scale
    volumeHandle:
"7794843418738962737;0A0002D9:5DBA3A3A;path=/gpfs/mongoafm/mongo"
```

Spectrum Scale enforces the existence of the files that were created on the Power Systems cluster in the AWS cluster, as shown in Example 7-23. Spectrum Scale also updates the writes on both clusters (sides). There is a **prefetch** command that brings all the data before,; therefore, only the changes are updated after prefetch is done. This makes accessing the full database faster in the cloud.

Example 7-23 Files as a cache and prefetched in the cloud

```
[root@ip-10-0-2-217 ~]# ls -la /gpfs/mongoafm/mongo/
total 146083
drwx------. 4 mongo mongo      8192 Nov  9 16:59 .
drwxr-xr-x. 7 root  root       4096 Oct 31 12:28 ..
-rw-------. 1 mongo root      16384 Nov  9 14:38
collection-0--1629751576978663310.wt
-rw-------. 1 mongo root      36864 Nov  9 16:58
collection-0-7744448727083060016.wt
-rw-------. 1 mongo root      32768 Nov  9 14:39
collection-2--1629751576978663310.wt
-rw-------. 1 mongo root  145391616 Nov  9 16:24
collection-3-7744448727083060016.wt
-rw-------. 1 mongo root      16384 Nov  9 14:38
collection-4--1629751576978663310.wt
drwx------. 2 mongo root       8192 Nov 10 12:01 diagnostic.data
-rw-------. 1 mongo root      16384 Nov  9 14:38 index-1--1629751576978663310.wt
-rw-------. 1 mongo root      36864 Nov  9 16:58 index-1-7744448727083060016.wt
-rw-------. 1 mongo root      36864 Nov  9 16:58 index-2-7744448727083060016.wt
-rw-------. 1 mongo root      32768 Nov  9 14:39 index-3--1629751576978663310.wt
-rw-------. 1 mongo root    3735552 Nov  9 16:24 index-4-7744448727083060016.wt
-rw-------. 1 mongo root      16384 Nov  9 14:38 index-5--1629751576978663310.wt
-rw-------. 1 mongo root      16384 Nov  9 15:27 index-6--1629751576978663310.wt
drwx------. 2 mongo root       8192 Nov  9 16:23 journal
-rw-------. 1 mongo root      36864 Nov  9 15:31 _mdb_catalog.wt
-rw-------. 1 mongo root          2 Nov  9 14:38 mongod.lock
-rw-------. 1 mongo root      36864 Nov  9 16:59 sizeStorer.wt
-rw-------. 1 mongo root        114 Nov  9 14:38 storage.bson
-rw-------. 1 mongo root         48 Nov  9 14:38 WiredTiger
-rw-------. 1 mongo root       4096 Nov  9 14:38 WiredTigerLAS.wt
-rw-------. 1 mongo root         21 Nov  9 14:38 WiredTiger.lock
-rw-------. 1 mongo root       1049 Nov  9 16:59 WiredTiger.turtle
-rw-------. 1 mongo root      69632 Nov  9 16:59 WiredTiger.wt
```

Check that the link to the Persistent Volume claim was created, bound to the PV that was just created, and also applied the file, as shown in Example 7-24.

Example 7-24 Persistent volume claim yaml file

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: appmongo-ibm-mongodb-dev-datavolume
spec:
  storageClassName: manual
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 2Gi
```

This scenario uses the same secret file used in the on-premise Power Systems cluster, as shown in Example 7-13 on page 170. The mongoDB Pod yaml file is also the same as used before as shown in Example 7-15 on page 171.

The service on AWS uses the load balancer that connects correctly the service on a predefined name on port 27017. To accomplish this, use the yaml file, as shown in Example 7-25.

Example 7-25 Service file for AWS

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: appmongo-ibm-mongodb-dev
  name: appmongo-ibm-mongodb-dev
spec:
  ports:
  - name: ibm-mongodb-dev
    port: 27017
    protocol: TCP
    targetPort: 27017
  selector:
    app: appmongo-ibm-mongodb-dev
  type: LoadBalancer
```

After applying all yaml files (as shown in Example 7-26), Mongodb is available in AWS.

Example 7-26 Applying all yaml files

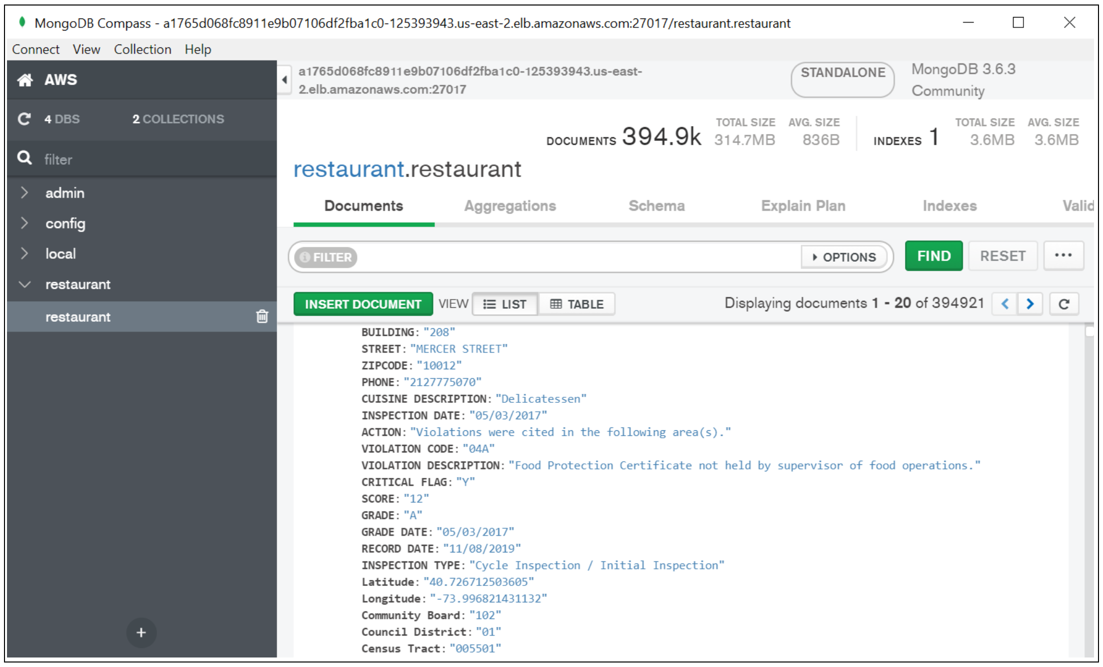
```
[root@ip-10-0-2-217 ~]# oc apply -f csipv.yaml
persistentvolume/appmongo-ibm-mongodb-dev-datavolume created
[root@ip-10-0-2-217 ~]# oc apply -f csipvc.yaml
persistentvolumeclaim/appmongo-ibm-mongodb-dev-datavolume created
[root@ip-10-0-2-217 ~]# oc apply -f secret.yaml
secret/appmongo-ibm-mongodb-dev created
[root@ip-10-0-2-217 ~]# oc apply -f awsservice.yaml
service/appmongo-ibm-mongodb-dev created
[root@ip-10-0-2-217 ~]# oc apply -f mongo_pod.yaml
pod/appmongo-ibm-mongodb-dev created
[root@ip-10-0-2-217 ~]#
```

The service is created by AWS. To get the address, check the service, as shown in Example 7-27.

Example 7-27 Checking service created

```
[root@ip-10-0-2-217 ~]# oc get service appmongo-ibm-mongodb-dev
NAME            TYPE      CLUSTER-IP   EXTERNAL-IP
PORT(S)           AGE
appmongo-ibm-mongodb-dev   LoadBalancer   172.30.43.162
a1765d068fc8911e9b07106df2fba1c0-125393943.us-east-2.elb.amazonaws.com
27017:31725/TCP   9d
[root@ip-10-0-2-217 ~]#
```

Now you can access your MongoDB on AWS. Point MongoDB to the load balancer IP address on the service and enter the admin user name and password that defined in the secret file, as shown in Figure 7-10.



```
BUILDING: "208"
STREET: "MERCER STREET"
ZIPCODE: "10012"
PHONE: "2127775070"
CUISINE DESCRIPTION: "Delicatessen"
INSPECTION DATE: "05/03/2017"
ACTION: "Violations were cited in the following area(s)."
VIOLATION CODE: "04A"
VIOLATION DESCRIPTION: "Food Protection Certificate not held by supervisor of food operations."
CRITICAL FLAG: "Y"
SCORE: "12"
GRADE: "A"
GRADE DATE: "05/03/2017"
RECORD DATE: "11/08/2019"
INSPECTION TYPE: "Cycle Inspection / Initial Inspection"
Latitude: "40.726712503605"
Longitude: "-73.996821431132"
Community Board: "102"
Council District: "01"
Census Tract: "005501"
```

Figure 7-10 Opening MongoDB on the AWS Cluster

All of the data that was inserted in the cluster that was installed on the Power Systems servers in Mexico is shown transparently on the x86 cluster at AWS. The multicloud environment is ready to be used.

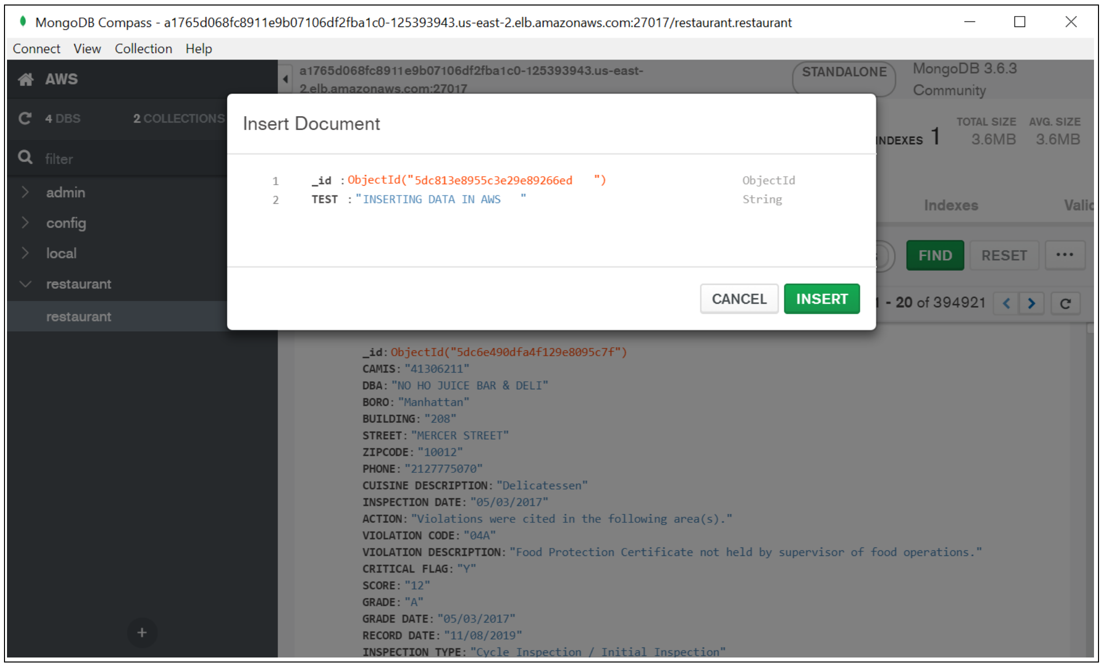
7.2.4 Testing the hybrid multicloud

With both OpenShift Container Platform clusters working, test the movement of the MongoDB pod, at the same time writing on both clouds. Start from the public cloud, inserting a test document to show the behavior and easiness of migration back to the on-premises cluster. Then, insert another document, move back to the cloud, and check both documents. Example 7-28 shows that the pod is running in the public cloud.

Example 7-28 Confirm Pod is running in the public cloud

```
[root@ip-10-0-2-217 ~]# oc get pod appmongo-ibm-mongodb-dev
NAME                                 READY    STATUS     RESTARTS    AGE
appmongo-ibm-mongodb-dev             1/1      Running    1           14m
[root@ip-10-0-2-217 ~]#
```

Click **Insert** to add the test document on the window that appears, as shown in Figure 7-11.

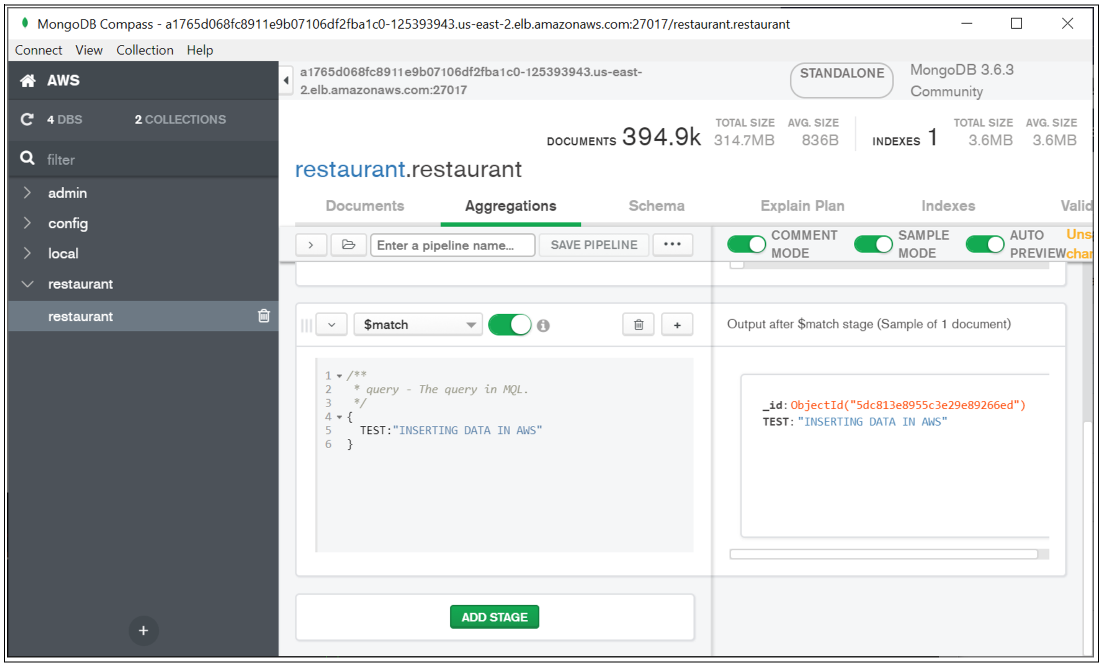

MongoDB Compass - a1765d068fc8911e9b07106df2fba1c0-125393943.us-east-2.elb.amazonaws.com:27017/restaurant.restaurant
Connect View Collection Help
AWS
4 DBS 2 COLLECTIONS
filter
admin
config
local
restaurant
restaurant
Insert Document
CANCEL
INSERT
FIND
RESET

```
_id : ObjectId("5dc813e8955c3e29e89266ed   ")        ObjectId
TEST : "INSERTING DATA IN AWS   "                    String
```

```
_id: ObjectId("5dc6e490dfa4f129e8095c7f")
CAMIS: "41306211"
DBA: "NO HO JUICE BAR & DELI"
BORO: "Manhattan"
BUILDING: "208"
STREET: "MERCER STREET"
ZIPCODE: "10012"
PHONE: "2127775070"
CUISINE DESCRIPTION: "Delicatessen"
INSPECTION DATE: "05/03/2017"
ACTION: "Violations were cited in the following area(s)."
VIOLATION CODE: "04A"
VIOLATION DESCRIPTION: "Food Protection Certificate not held by supervisor of food operations."
CRITICAL FLAG: "Y"
SCORE: "12"
GRADE: "A"
GRADE DATE: "05/03/2017"
RECORD DATE: "11/08/2019"
INSPECTION TYPE: "Cycle Inspection / Initial Inspection"
```

Figure 7-11 Inserting data (test document) into AWS running container

Check that the document was correctly inserted, Click **Aggregations**, and filter to match the document that was just created, as shown in Figure 7-12.


MongoDB Compass - a1765d068fc8911e9b07106df2fba1c0-125393943.us-east-2.elb.amazonaws.com:27017/restaurant.restaurant
Connect View Collection Help
AWS
4 DBS 2 COLLECTIONS
filter
admin
config
local
restaurant
restaurant
a1765d068fc8911e9b07106df2fba1c0-125393943.us-east-2.elb.amazonaws.com:27017
STANDALONE
MongoDB 3.6.3 Community
DOCUMENTS 394.9k TOTAL SIZE 314.7MB AVG. SIZE 836B
INDEXES 1 TOTAL SIZE 3.6MB AVG. SIZE 3.6MB
restaurant.restaurant
Documents Aggregations Schema Explain Plan Indexes
Enter a pipeline name...
SAVE PIPELINE
COMMENT MODE
SAMPLE MODE
AUTO PREVIEW
$match
Output after $match stage (Sample of 1 document)
ADD STAGE

```
/**
 * query - The query in MQL.
 */
{
  TEST:"INSERTING DATA IN AWS"
}
```

```
_id: ObjectId("5dc813e8955c3e29e89266ed")
TEST: "INSERTING DATA IN AWS"
```

Figure 7-12 Checking data inserted in the public cloud

Delete the pod that is running in the public cloud, as shown in Example 7-29.

Example 7-29   Deleting the Pod from the public cloud

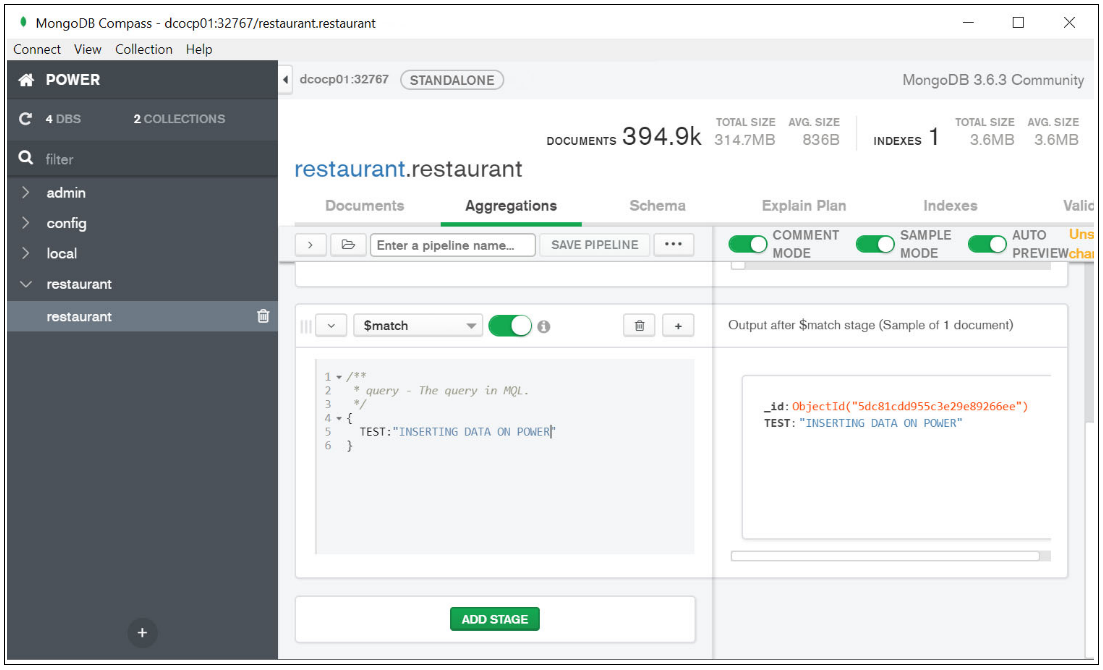
```
[root@ip-10-0-2-217 ~]# oc delete pod appmongo-ibm-mongodb-dev
pod "appmongo-ibm-mongodb-dev" deleted
[root@ip-10-0-2-217 ~]# oc get pod appmongo-ibm-mongodb-dev
Error from server (NotFound): pods "appmongo-ibm-mongodb-dev" not found
root@ip-10-0-2-217 ~]#
```

Bring the pod up on-premises, as shown in Example 7-30.

Example 7-30   Bring Pod up at the on-premises cloud

```
[root@dcocp01 ~]# oc apply -f mongo_pod.yaml
pod/appmongo-ibm-mongodb-dev created
[root@dcocp01 ~]# oc get pod appmongo-ibm-mongodb-dev
NAME                        READY     STATUS     RESTARTS    AGE
appmongo-ibm-mongodb-dev    1/1       Running    0           1m
```

Now, open the restaurant database at the on-premises cluster, and again on the Aggregation tab. Look for the entry that was inserted in the public cloud. You can see the match, as shown in Figure 7-13.

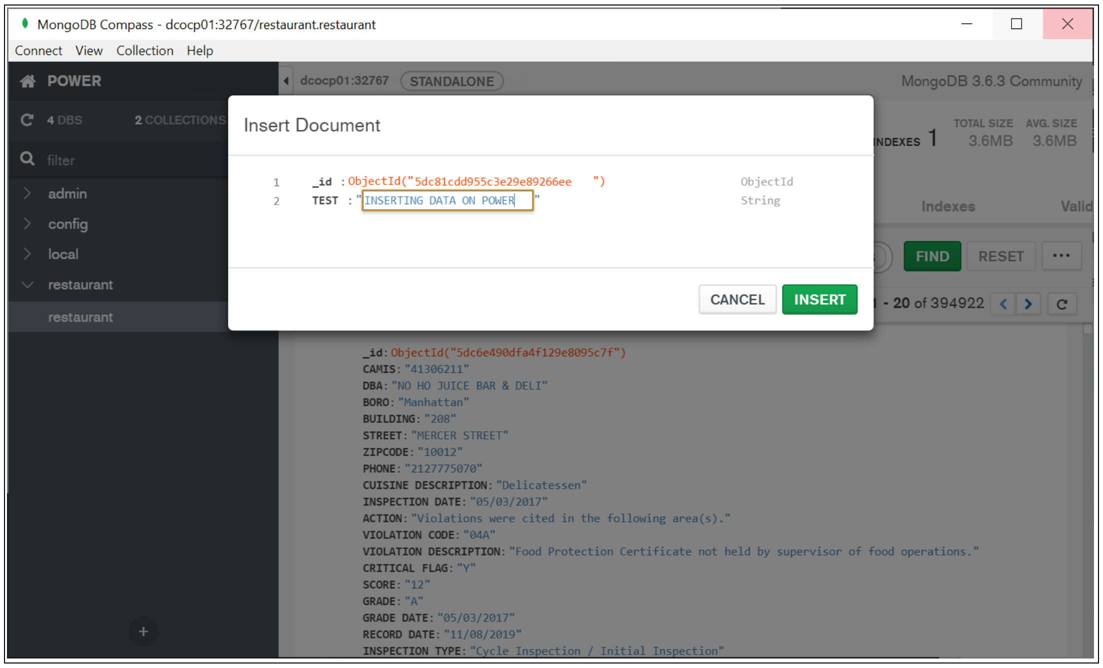

MongoDB Compass - dcocp01:32767/restaurant.restaurant
Connect View Collection Help
POWER
4 DBS 2 COLLECTIONS
filter
admin
config
local
restaurant
restaurant
dcocp01:32767 STANDALONE
MongoDB 3.6.3 Community
DOCUMENTS 394.9k TOTAL SIZE 314.7MB AVG. SIZE 836B
INDEXES 1 TOTAL SIZE 3.6MB AVG. SIZE 3.6MB
restaurant.restaurant
Documents Aggregations Schema Explain Plan Indexes
Enter a pipeline name...
SAVE PIPELINE
COMMENT MODE
SAMPLE MODE
AUTO PREVIEW
$match
Output after $match stage (Sample of 1 document)
ADD STAGE

```
/**
 * query - The query in MQL.
 */
{
  TEST:"INSERTING DATA ON POWER"
}
```

```
_id: ObjectId("5dc81cdd955c3e29e89266ee")
TEST: "INSERTING DATA ON POWER"
```

Figure 7-13   Matching document created at the public cloud on-premises

To test the migration from on-premises to the cloud, create another document. Click the **Documents** tab (see Figure 7-13 on page 187). Then, click **Insert** to add the document, as shown in Figure 7-14.



```
_id : ObjectId("5dc81cdd955c3e29e89266ee   ")      ObjectId
TEST : "INSERTING DATA ON POWER"                   String
```

```
_id: ObjectId("5dc6e490dfa4f129e8095c7f")
CAMIS: "41306211"
DBA: "NO HO JUICE BAR & DELI"
BORO: "Manhattan"
BUILDING: "208"
STREET: "MERCER STREET"
ZIPCODE: "10012"
PHONE: "2127775070"
CUISINE DESCRIPTION: "Delicatessen"
INSPECTION DATE: "05/03/2017"
ACTION: "Violations were cited in the following area(s)."
VIOLATION CODE: "04A"
VIOLATION DESCRIPTION: "Food Protection Certificate not held by supervisor of food operations."
CRITICAL FLAG: "Y"
SCORE: "12"
GRADE: "A"
GRADE DATE: "05/03/2017"
RECORD DATE: "11/08/2019"
INSPECTION TYPE: "Cycle Inspection / Initial Inspection"
```

Figure 7-14 Inserting data in the Power Systems Cluster

Delete the container from the on-premises cluster, as shown in Example 7-31.

Example 7-31 Delete MongoDB Pod on-premises cloud

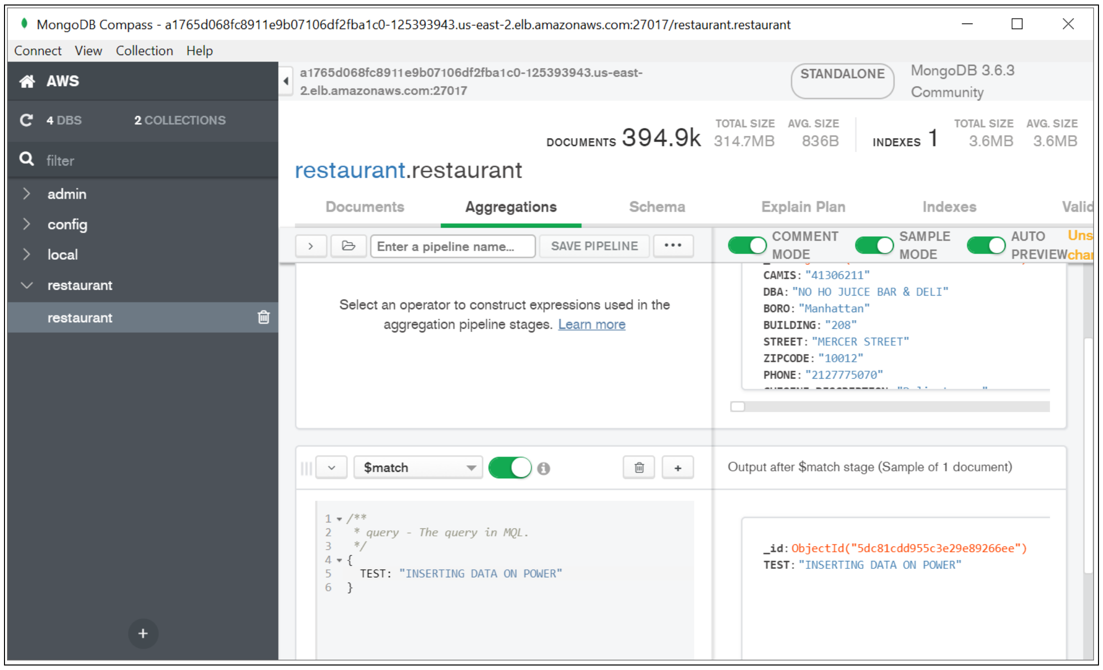
```
[root@dcocp01 ~]# oc delete pod appmongo-ibm-mongodb-dev
pod "appmongo-ibm-mongodb-dev" deleted
[root@dcocp01 ~]#
```

Bring the container up in the public cloud, as shown in Example 7-32.

Example 7-32 Bringing MongoDB up in the public cloud

```
[root@ip-10-0-2-217 ~]# oc apply -f mongo_pod.yaml
pod/appmongo-ibm-mongodb-dev created
[root@ip-10-0-2-217 ~]#
```

As you notice, the migration steps are just two lines and can be automated in many ways. Open the database and on the Aggregation tab, look for the document that was inserted in the cluster on-premises. The result is shown in Figure 7-15.


MongoDB Compass - a1765d068fc8911e9b07106df2fba1c0-125393943.us-east-2.elb.amazonaws.com:27017/restaurant.restaurant
Connect View Collection Help
AWS
4 DBS 2 COLLECTIONS
filter
admin
config
local
restaurant
restaurant
a1765d068fc8911e9b07106df2fba1c0-125393943.us-east-2.elb.amazonaws.com:27017
STANDALONE
MongoDB 3.6.3 Community
DOCUMENTS 394.9k TOTAL SIZE 314.7MB AVG. SIZE 836B
INDEXES 1 TOTAL SIZE 3.6MB AVG. SIZE 3.6MB
restaurant.restaurant
Documents Aggregations Schema Explain Plan Indexes
Enter a pipeline name...
SAVE PIPELINE
COMMENT MODE
SAMPLE MODE
AUTO PREVIEW
$match
Output after $match stage (Sample of 1 document)

Select an operator to construct expressions used in the aggregation pipeline stages. Learn more

```
CAMIS: "41306211"
DBA: "NO HO JUICE BAR & DELI"
BORO: "Manhattan"
BUILDING: "208"
STREET: "MERCER STREET"
ZIPCODE: "10012"
PHONE: "2127775070"
```

```
/**
 * query - The query in MQL.
 */
{
  TEST: "INSERTING DATA ON POWER"
}
```

```
_id: ObjectId("5dc81cdd955c3e29e89266ee")
TEST: "INSERTING DATA ON POWER"
```

Figure 7-15 Data inserted on-premises seen at the public cloud MongoDB

For a seamless experience, a network solution that knows where the application is running must be in place. For more information, see Appendix C, “Seamless application movement across multicloud environments” on page 241.

8

Special topics

This chapter discusses some other topics that can be used to enhance the use of and manage Red Hat OpenShift and IBM Cloud Paks on Power Systems.

This chapter includes the following topics:

8.1 IBM Multicloud Manager: Container orchestration

IBM Multicloud Manager is used to manage OpenShift deployments across multicloud platforms.

8.1.1 IBM Multicloud Manager overview

There are many challenges that you will be faced with when trying to manage workloads across multiple cloud environments as you modernized your environment and adopted a hybrid and multicloud approach. You are looking at an enterprise out approach where you will take your on-premises environment that can be an private cloud and scale-out to a public cloud. You also might be looking at the cloud-in approach where you have a workload in a public cloud environment and want to connect it back to a datacenter.

Both of these approaches can be characterized as both hybrid and multicloud.

Whichever approach you take, the challenges remain the same:

- How can you develop modern cloud-native applications and integrate into your existing environments?
- How can you monitor data movement and comply with appropriate governance regulations?
- How can service management, security and compliance be maintained and monitored?

IBM Multicloud Manager is designed facilitate the journey to cloud. Helping organizations orchestrate, manage, and monitor their containerized workloads across multiple data centers and public or private clouds allows your services to be managed as though they were a single unified environment.

IBM Multicloud Manager supports multiple public cloud providers and private cloud environments:

- Public cloud providers:
  - IBM Cloud
  - Amazon Web Services
  - Google Cloud Platform
  - Microsoft Azure
- Private cloud:
  - Red Hat OpenShift
  - IBM Cloud Private

8.1.2 Key features and capabilities of IBM Multicloud Manager

In this section, we provide a high-level overview of some of the key features of IBM Multicloud Manager.

Multicluster management and visibility

Parallel queries can be performed against multiple clusters aggregate the information received based on certain criteria. You also can view real-time pod traffic to determine pod traffic flows.

Cluster health can be viewed by region or cloud provider. You also can easily provision, upgrade, and de-provision clusters across multiple hybrid and multicloud environments.

A complete view of your clusters is available in the Overview page, as shown in Figure 8-1.

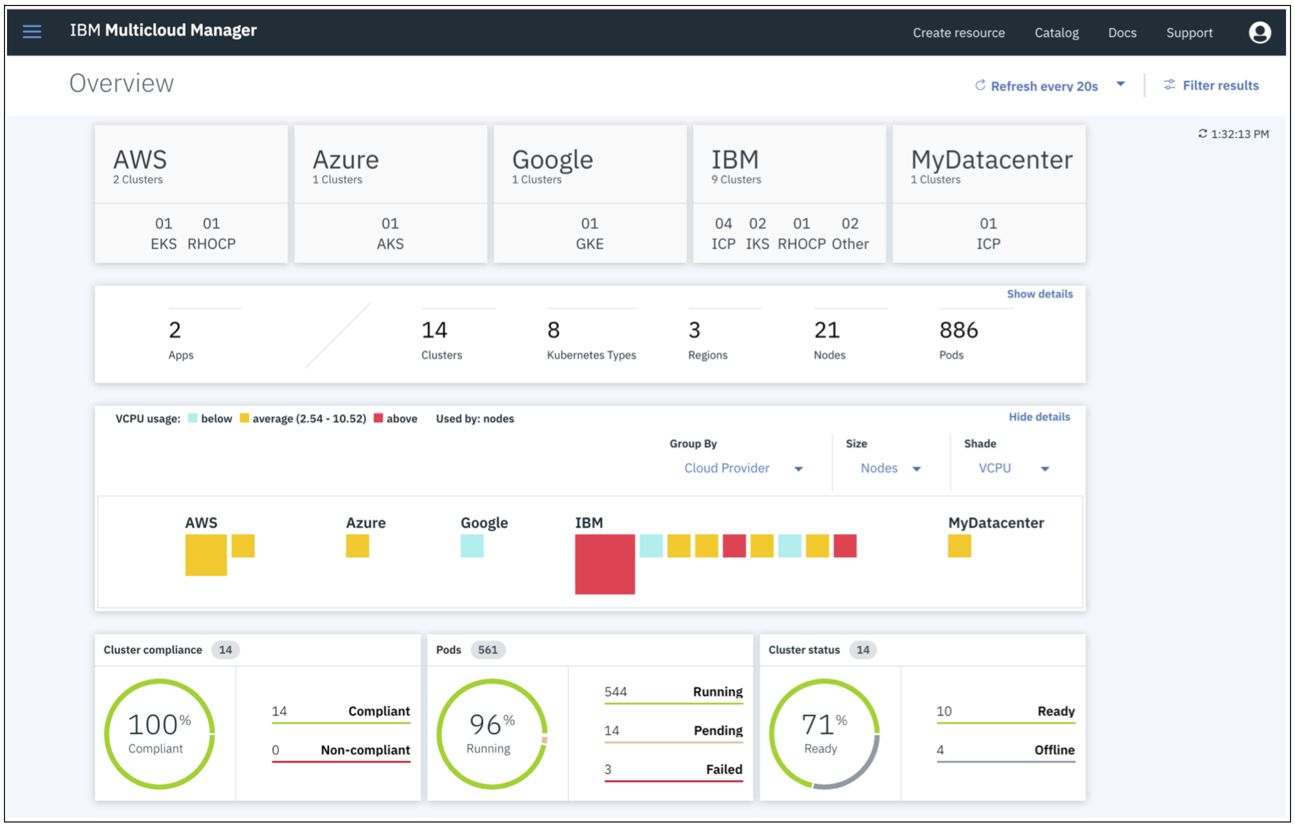


Figure 8-1 Multicloud Manager overview

Governance and risk management

You can set policies for applications, security, and infrastructure to maintain enforceable governance and compliance across all your clouds. You also can view and manage policy violations and security risks raised against clusters, which can be categorized by severity. Customizable views are also available so that you can see only the information that is relevant to your own clusters.

Monitoring, logging, and auditing

You can dynamically monitor and resolve problems by using opens source tools, such as Grafana, Prometheusm, and ELK stack. Predictive alert systems can be set up, including automatic backup and Disaster Recovery options and workload transfers.

Automated deployment of clusters

IBM Multicloud Manager leverages IBM Cloud Automation Manager services to provision, configure, and deliver individual Kubernetes clusters as a service in any cloud that Cloud Automation Manager supports.

8.2 Moving data across clouds

Sharing data between cloud environments can be a challenge, especially if your data sets are large or your cloud environments are geographically dispersed. IBM Aspera® on Cloud is an IBM hosted service for quick and reliable movement of data between cloud environments.

Traditional file transfer methods, such as FTP or HTTP, and other file transfer protocols can be inherently slow and unreliable for transferring large amounts of data. IBM Aspera is designed to move data files of any size of volume reliably, quickly and securely.

IBM Aspera uses patented Fast, Adaptive and Secure Protocol or FASP® technology to support thousands of concurrent transfer requests. This feature enables Aspera to deliver high throughput over networks with high latency, such as WAN. This allows high-performance secure transport of files, directories, and other large data sets to, from, and between cloud storage.

8.2.1 IBM Aspera key features and benefits

IBM Aspera includes the following key features and benefits:

- Easy and intuitive file sharing and content delivery across a hybrid-cloud environment
- High-speed data transfer at any distance
- Cloud-native technology with high availability and scalability
- Enterprise-grade security
- Real-time control over your transfers
- Central administration of hybrid environments
- Automated transfers based on schedule, file arrival event, or an API call

For more information, see this web page.

8.2.2 Using IBM Aspera in a Hybrid cloud environment

IBM Aspera gives you the ability to connect your on-premises storage with on-premises private cloud and off-premises public cloud to access files and folders in multiple environments. IBM Aspera supports all of the leading cloud platforms, such as IBM Cloud, AWS, Azure, and Google Cloud Platform.

You can configure access between the transfer nodes in your private and public cloud environments to ensure seamless data transfer. You can embed file and folder delivery within your applications by using the Aspera API. IBM Aspera's containerized, scalable software is optimized to run on and is certified on Red Hat OpenShift. It is included in the IBM Cloud Pak for Integration to give you an end-to-end solution for hybrid cloud integration.

For more information about integrating IBM Aspera in a hybrid cloud environment, see the data sheet *IBM Aspera on Cloud*.

IBM Aspera also supports the main cloud object storage services, including the following examples:

- IBM Cloud
- Amazon AWS S3
- OpenStack Swift version 1.12 and Up
- Microsoft Azure BLOB
- Akamai NetStorage
- Google Storage

- Limelight Object Storage
- HDFS
- HGST
- NetApp Object Storage

For more information about the benefits of using IBM Aspera with third-party cloud storage services, see the white paper *IBM Aspera Direct-to-Cloud Storage*.

8.3 Configuring a multicloud data lake

This section introduces the concept of a data lake and how it fits in a multicloud environment.

8.3.1 Data lake overview

A data lake is a next-generation hybrid data management solution that is designed to meet the challenge and need to deal with big data in a hybrid multicloud environment. It is not a product; rather, it is a hybrid data management reference architecture that is designed to form part of your data governance strategy.

The use of a data lake offers greater flexibility than a data warehouse. A data lake consolidates an organization’s data into a governed and well-managed environment that supports production workloads and analytics development.

The use of a data lake includes the following benefits:

- Storing data in its native format means less time spent on data preparation
- Simplified data access
- Enhanced agility for applications data users
- Improved decision-making
- Reduced costs

For more information about data lakes and their benefits, see the following resources:

- What is a Data Lake?
- *Build a better data lake*

The following data lake providers operate in a multicloud environment:

- IBM (through a partnership with Cloudera)
- Google
- Amazon AWS
- Microsoft Azure
- Hadoop
- Terradata
- Hortonworks

8.3.2 Using a data lake in a multicloud environment

Using a data lake in a multicloud environment enables secure integration and provides a means to achieve unified data governance in a hybrid environment. Your data lake repositories accept data from any data source in its native format, which provides a common platform for containerized applications to use in both on-premises data centers and in the cloud.

Connecting and integrating applications across traditional on-premises environments with private and public cloud services can be challenging. Using a data lakes as part of your strategy to move to the cloud enables secure multicloud integration by providing a means for an enterprise to achieve unified data governance in a hybrid environment.

For more information about integrating a data lake into a multicloud strategy, see this web page.

Part 4

Appendixes

This part details the creation and installation of the Lab environments that are used during the development of this publication.

The following appendixes are included in this part:

A

Sample lab: Deployment and Pod management

In this appendix, we describe how to deploy a single application, test the resiliency, and scale the deployment.

This appendix includes the following topics:

A.1 Connecting to the lab environment

To connect to the lab environment, you need access to the account credentials to your `admin` OpenShift web console and the terminal window with `root` privileges, as shown in Figure A-1.

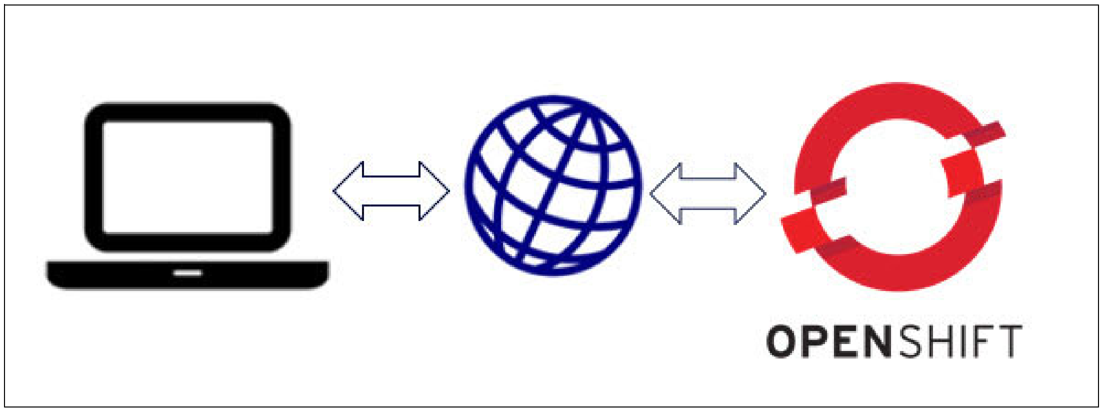


Figure A-1 Access OpenShift web console

Complete the following steps:

1. Modify the hosts file on your local computer.

   The containers that are deployed in OpenShift are on a private network within the OpenShift cluster. Accessing applications from an external connection requires network and deployment planning, which is beyond the scope of this lab exercise. In this step, you modify the hosts file on your local machine to access the deployments that were created for this lab, as shown in Example A-1.

   Example A-1 Modify the hosts file

   ```
   IP_ADDRESS console.router.default.svc.cluster.local app-http-git
   ```

2. Open a terminal window to the OCP machine with a user with `root` privileges. Run the **`ssh`** command or PuTTY from your local computer.
3. Verify the release of Red Hat and other pieces of information about your operating system, as shown in Example A-2.

   Example A-2 Verify the release of Red Hat

   ```
   $ uname -srm
   Linux 3.10.0-957.21.3.el7.ppc64le ppc64le
   ```

A.2 Creating a user and project by using the OpenShift command line

Complete the following steps:

1. Add a user called `user1` with password `cloudonpower`, as shown in Example A-3.

   Example A-3 Adding a user

   ```
   $ sudo adduser user1
   $ echo “cloudonpower” | sudo passwd user1 --stdin
   ```

2. Run the **htpasswd** command to update the flat files that are used to store user names and passwords for basic authentication of HTTP users. Use user `user1` with password `cloudonpower`, as shown in Example A-4.

   Example A-4 Updating password for an HTTP user

   ```
   $ sudo htpasswd -b /etc/origin/master/htpasswd user1 cloudonpower
   Adding password for user user1
   ```

3. Assign the new user cluster administration rights by logging in as cluster administrator (`root` privileges) into the default project by using the OpenShift CLI command (**oc**), as shown in Example A-5.

   Example A-5 Logging in to the default project

   ```
   $ oc login -u user615 -n default
   Server [https://localhost:8443]:
   The server is using a certificate that does not match its hostname: x509:
   certificate is valid for kubernetes, kubernetes.default,
   kubernetes.default.svc, kubernetes.default.svc.cluster.local, openshift,
   openshift.default, openshift.default.svc, openshift.default.svc.cluster.local,
   p615-kvm1.cecc.ihost.com, 129.40.252.236, 172.30.0.1, not localhost
   You can bypass the certificate check, but any data you send to the server could
   be intercepted by others.
   Use insecure connections? (y/n): y

   Authentication required for https://localhost:8443 (openshift)
   Username: user615
   Password: *********
   Login successful.

   You have access to the following projects and can switch between them with 'oc
   project <projectname>':

     * default
       kube-public
       kube-service-catalog
       kube-system
       management-infra
       openshift
       openshift-ansible-service-broker
       openshift-console
       openshift-infra
       openshift-logging
       openshift-monitoring
   ```

```
    openshift-node
    openshift-sdn
    openshift-template-service-broker
    openshift-web-console

Using project "default".
Welcome! See 'oc help' to get started.
```

4. List all currently defined OCP users by using the **`oc get`** command, as shown in Example A-6.

Example A-6 Listing all currently defined OCP users

```
$ oc get user
NAME      UID                                    FULL NAME   IDENTITIES
user615   3e8d8153-f5f8-11e9-b535-fa163e4226a4
htpasswd_auth:user615
```

5. Run the **`oc policy`** command to assign the `user1` user cluster admin rights, as shown in Example A-7.

Example A-7 Assigning user cluster admin rights

```
$ oc adm policy add-cluster-role-to-user cluster-admin user1
Warning: User 'user1' not found
cluster role "cluster-admin" added: "user1"
```

6. Log out by using the **`oc logout`** command, as shown in Example A-8.

Example A-8 Logging out from OpenShift

```
$ oc logout
Logged "user615" out on "https://localhost:8443"
```

7. Log back in to OpenShift CLI as `user1`, as shown in Example A-9.

Example A-9 Logging in to OpenShift

```
$ oc login -u user1
Authentication required for https://localhost:8443 (openshift)
Username: user1
Password: *********
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

  * default
    kube-public
    kube-service-catalog
    kube-system
    management-infra
    openshift
    openshift-ansible-service-broker
    openshift-console
    openshift-infra
    openshift-logging
    openshift-monitoring
    openshift-node
    openshift-sdn
    openshift-template-service-broker
    openshift-web-console

Using project "default".
```

8. List again all currently defined OCP users by using the **`oc get`** command. Now, you see that `user1` is included in the list, as shown in Example A-10.

Example A-10 List all currently defined OCP users

```
$ oc get user
NAME      UID                                    FULL NAME   IDENTITIES
user1     4da3954f-f77b-11e9-b535-fa163e4226a4
htpasswd_auth:user1
user615   3e8d8153-f5f8-11e9-b535-fa163e4226a4
htpasswd_auth:user615
```

9. Run the **`oc get project`** command to list all defined OpenShift projects, as shown in Example A-11.

Example A-11 Listing all defined OpenShift projects

```
$ oc get project
NAME                                DISPLAY NAME   STATUS
default                                            Active
kube-public                                        Active
kube-service-catalog                               Active
kube-system                                        Active
management-infra                                   Active
openshift                                          Active
openshift-ansible-service-broker                   Active
openshift-console                                  Active
openshift-infra                                    Active
openshift-logging                                  Active
openshift-monitoring                               Active
openshift-node                                     Active
openshift-sdn                                      Active
openshift-template-service-broker                  Active
openshift-web-console                              Active
```

10. Run the **`oc new-project`** command to create a project that is called `project1`, as shown in Example A-12.

Example A-12 Creating a project

```
$ oc new-project project1
Now using project "project1" on server "https://localhost:8443".

You can add applications to this project with the 'new-app' command. For
example, try:

    oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.
```

A.3 Logging in to the OpenShift web console

Complete the following steps:

1. From your local computer, open a web browser and browse to the master node by using port `8443`, as shown in Example A-13.

Example A-13 Logging in to the OpenShift web console

```
https://<master_ip>:8443
```

2. Log in as `user1`, as shown in Figure A-2.

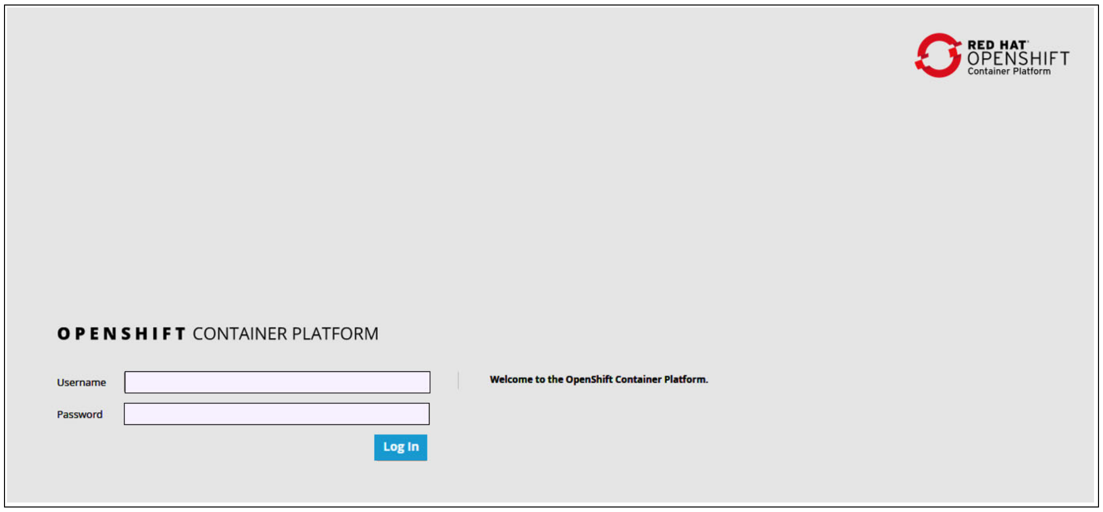

RED HAT OPENSHIFT Container Platform
OPENSHIFT CONTAINER PLATFORM
Username
Password
Log In
Welcome to the OpenShift Container Platform.

Figure A-2 Logging in to the OpenShift web console

The OpenShift web console is displayed, as shown in Figure A-3.

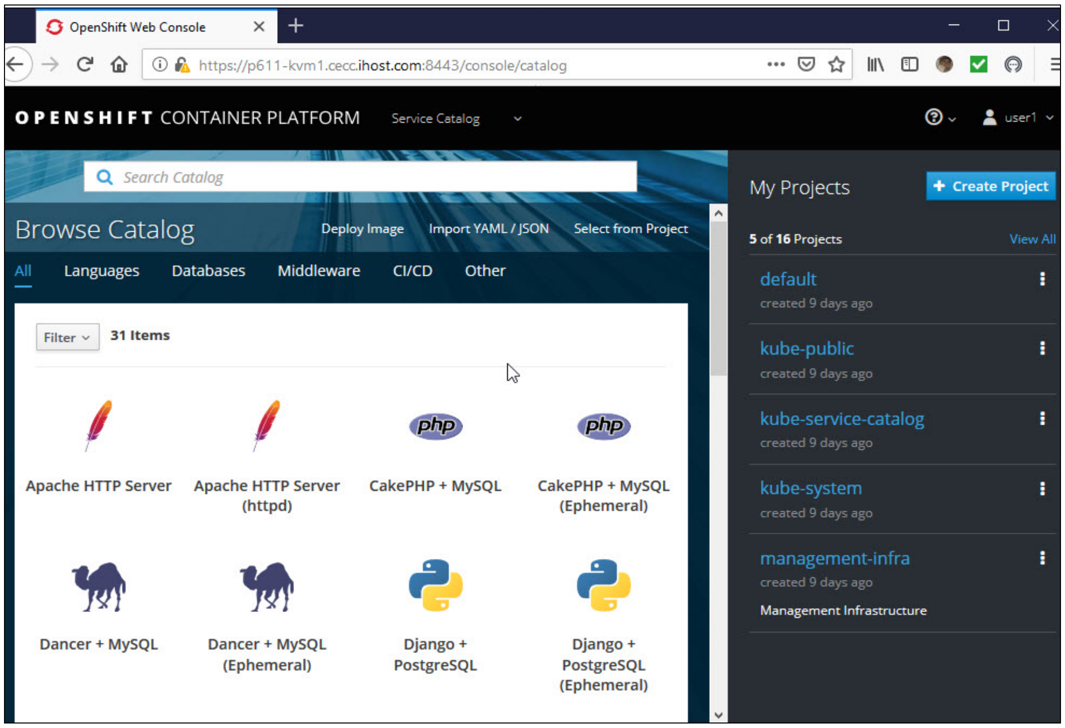


Figure A-3 Service Catalog View

A.4 Deploying an Apache server by using the OpenShift web console

Complete the following steps:

1. In the Service Catalog view, click **Apache HTTP Server,** as shown in Figure A-4.

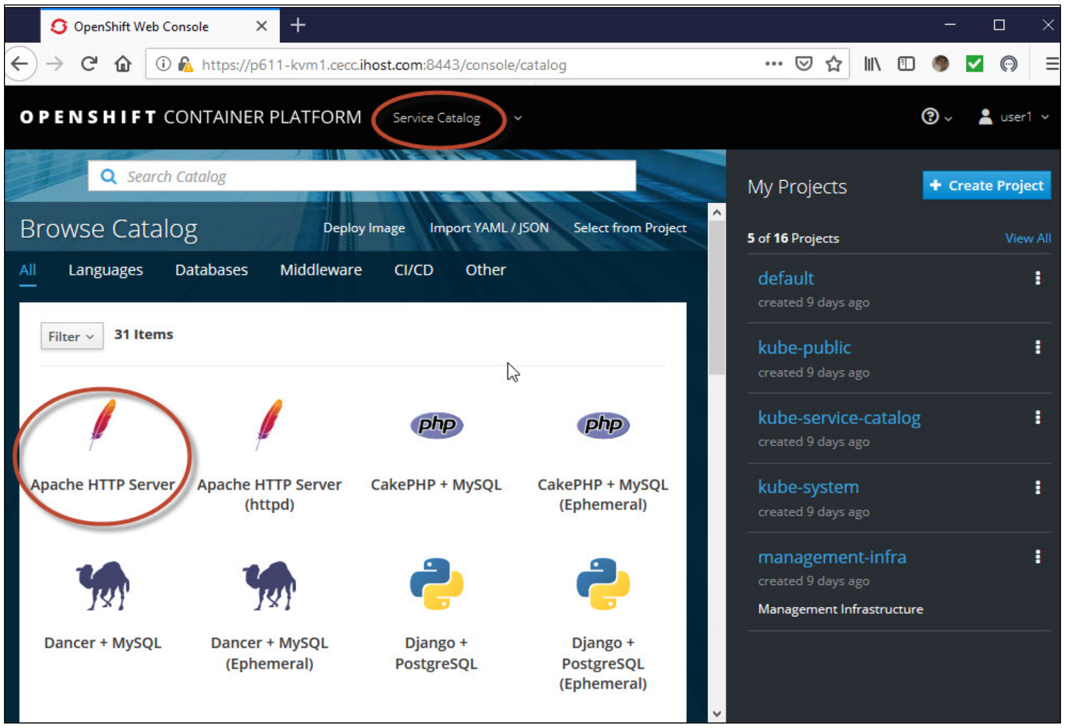
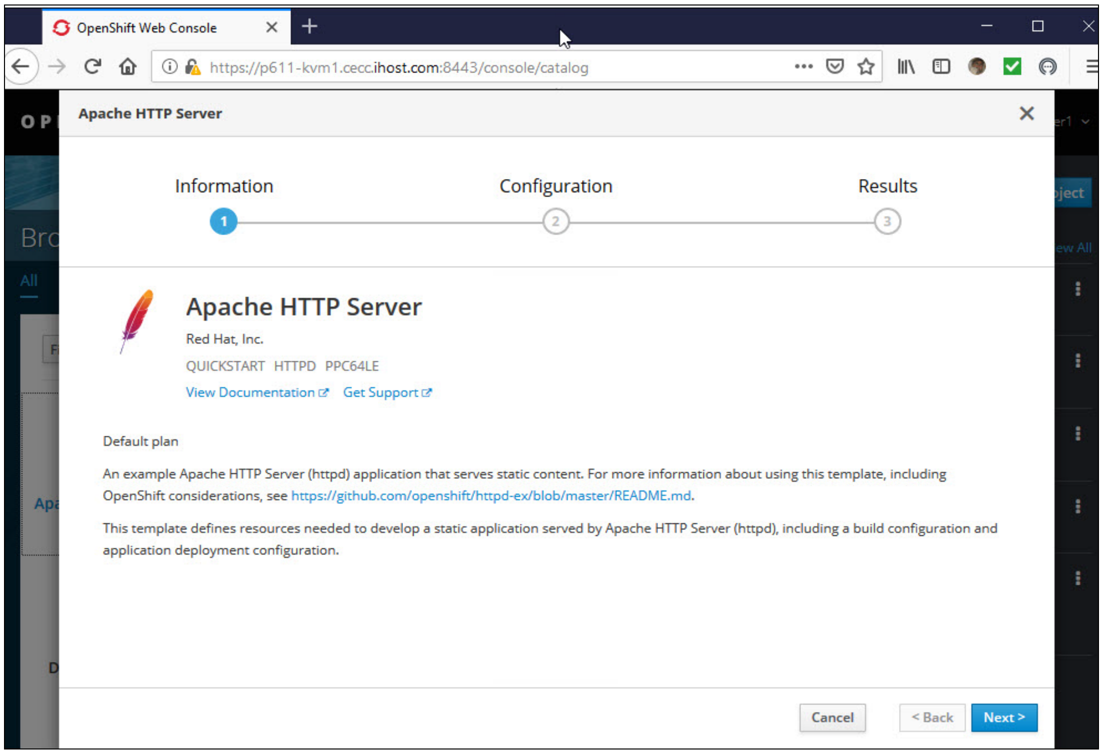


Figure A-4 Select Apache HTTP Server

2. In the Information window, click **Next**, as shown in Figure A-5.



Apache HTTP Server

Red Hat, Inc.

QUICKSTART HTTPD PPC64LE

View Documentation Get Support

Default plan

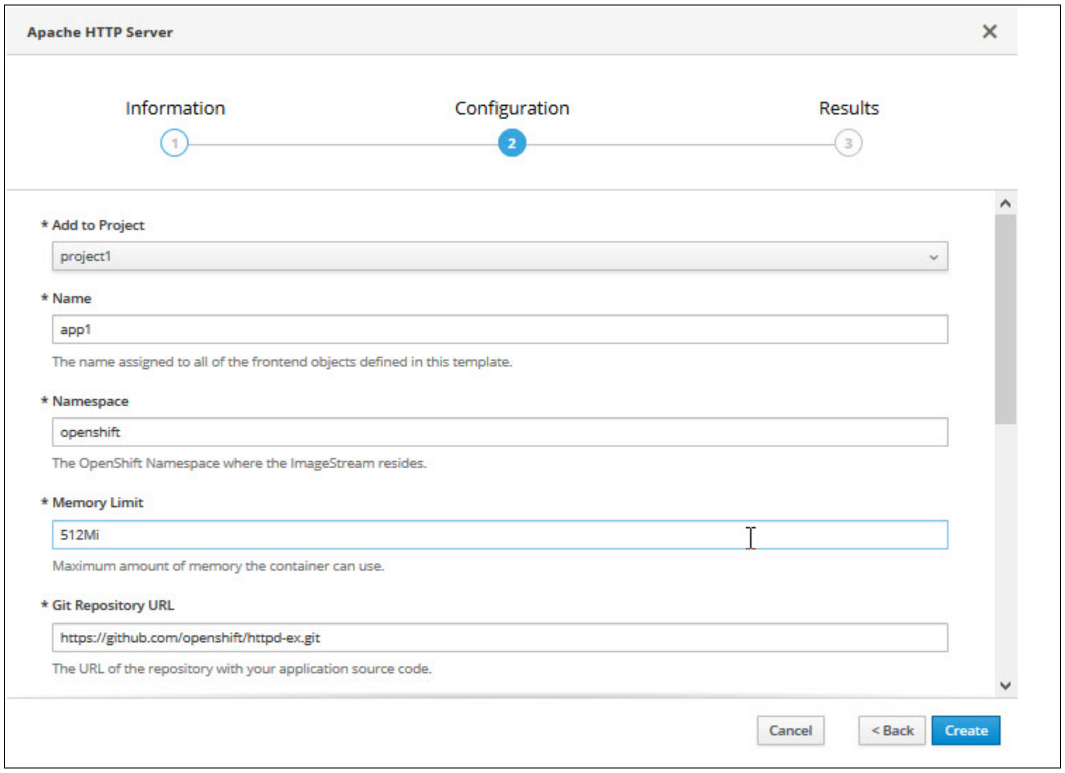
An example Apache HTTP Server (httpd) application that serves static content. For more information about using this template, including OpenShift considerations, see https://github.com/openshift/httpd-ex/blob/master/README.md.

This template defines resources needed to develop a static application served by Apache HTTP Server (httpd), including a build configuration and application deployment configuration.

Figure A-5 Information window

3. In the Configuration window, configure the following settings, as shown in Figure A-6:

   – Add to project: `project1`
   – Name: `app1`
   – Namespace: `openshift`
   – Memory limit: `512 Mi`
   – Git repository URL: `https://github.com/openshift/httpd-ex.git`
   – Application hostname: `app1-http-git`



* Add to Project: project1

* Name: app1

The name assigned to all of the frontend objects defined in this template.

* Namespace: openshift

The OpenShift Namespace where the ImageStream resides.

* Memory Limit: 512Mi

Maximum amount of memory the container can use.

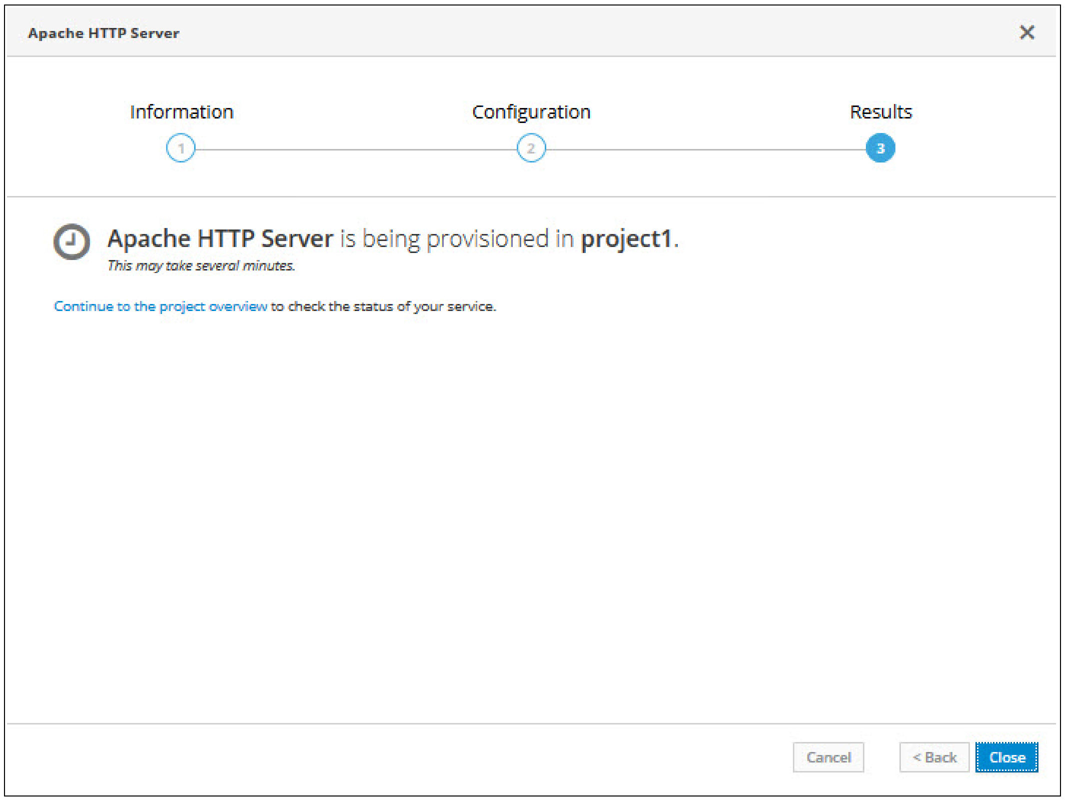
* Git Repository URL: https://github.com/openshift/httpd-ex.git

The URL of the repository with your application source code.

Figure A-6 Configuration window

4. To complete the deployment, click **Create**.

5. The Results window is displayed and shows the progress of the deployment (see Figure A-7). To continue, click **Close**.


Apache HTTP Server

Information 1 — Configuration 2 — Results 3

Apache HTTP Server is being provisioned in **project1**.

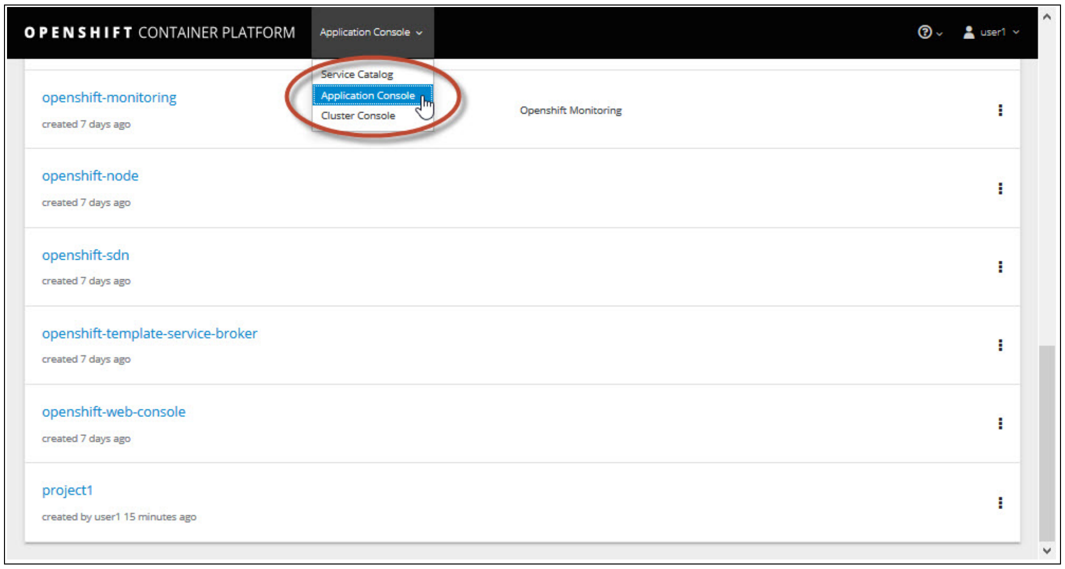
This may take several minutes.

Continue to the project overview to check the status of your service.

Cancel | < Back | Close

Figure A-7 Results window

6. In the Application Console view, browse to the `project1` project page by selecting the project from the project list, as shown in Figure A-8. Scroll down the list to find your project. Click **project1**.



openshift-monitoring created 7 days ago	Openshift Monitoring
openshift-node created 7 days ago	
openshift-sdn created 7 days ago	
openshift-template-service-broker created 7 days ago	
openshift-web-console created 7 days ago	
project1 created by user1 15 minutes ago	

Figure A-8 Application Console view

The deployments within the project are listed, as shown in Figure A-9.

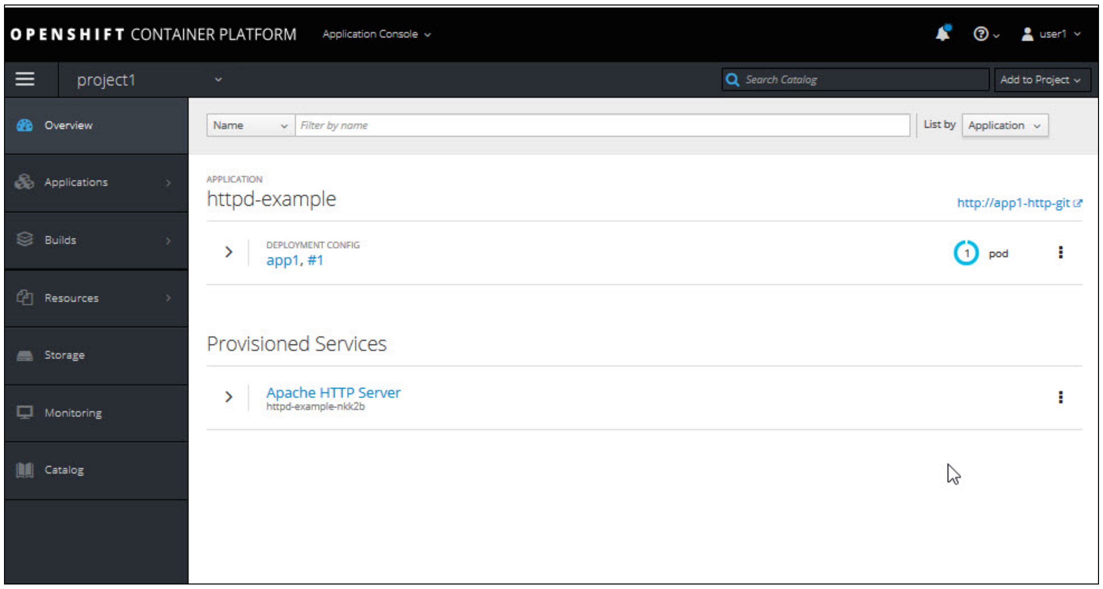


Figure A-9   Deployments within the project

7. Select the link for service on the `app1` that was deployed, as shown in Figure A-10. A new tab opens in the browser and displays the Welcome page of your application.

> **Note:** The service link relies on the hosts file entry for the name resolution that you made on your local computer.

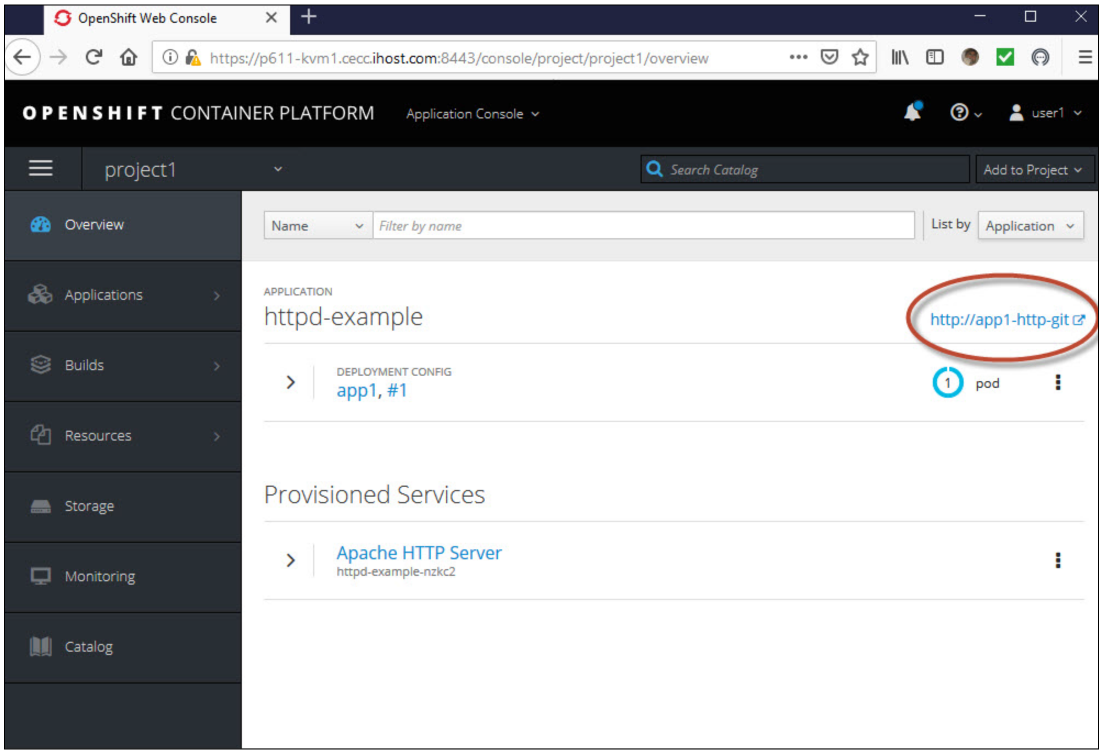
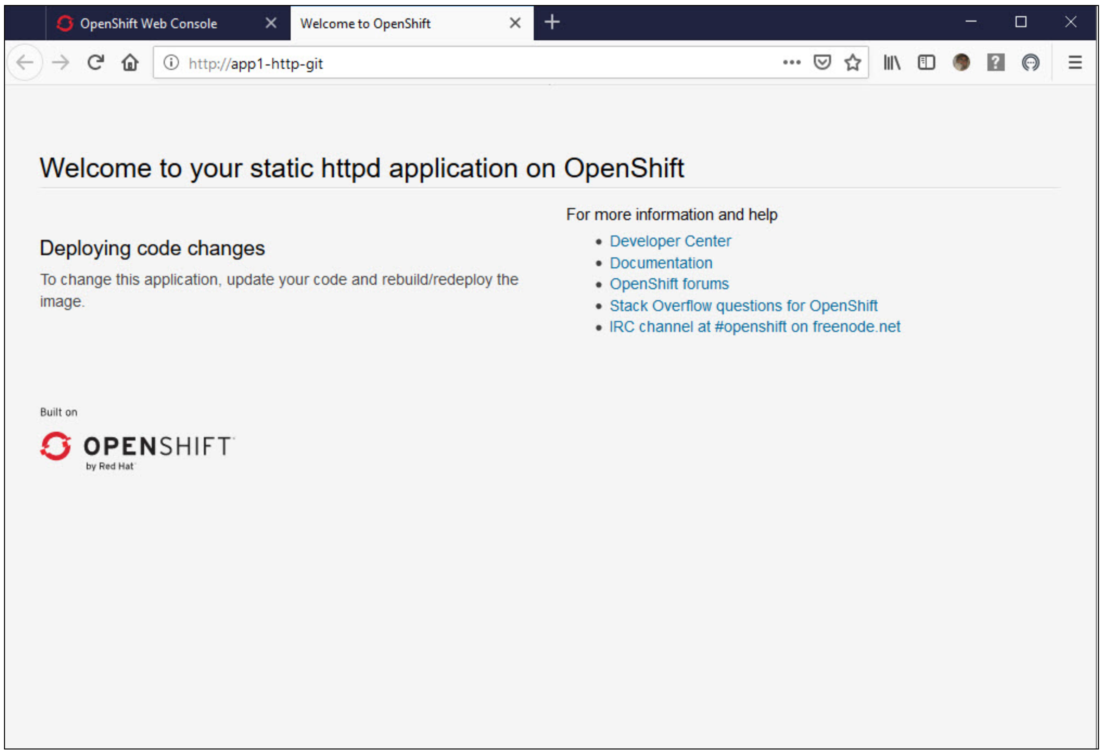


Figure A-10   Selecting the link for service

8. The Welcome to your static httpd application on OpenShift page is displayed in the new tab of your browser, as shown in Figure A-11.



Welcome to your static httpd application on OpenShift

Deploying code changes

To change this application, update your code and rebuild/redeploy the image.

For more information and help

- Developer Center
- Documentation
- OpenShift forums
- Stack Overflow questions for OpenShift
- IRC channel at #openshift on freenode.net

Figure A-11 Welcome page to your static httpd application

A.5 Verifying the status of the deployment

Complete the following steps:

1. From the Application Console view in the web console, select **Applications** → **Pods**, as shown in Figure A-12.

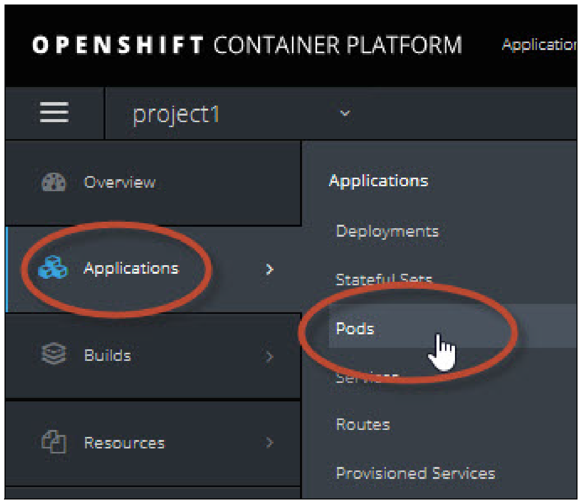
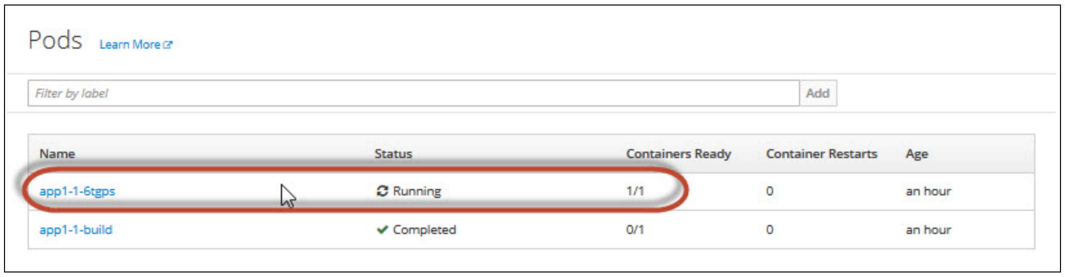


Figure A-12 Application Console view

2. From the displayed list of pods (see Figure A-13), confirm how many containers are running.



Pods Learn More

Filter by label Add

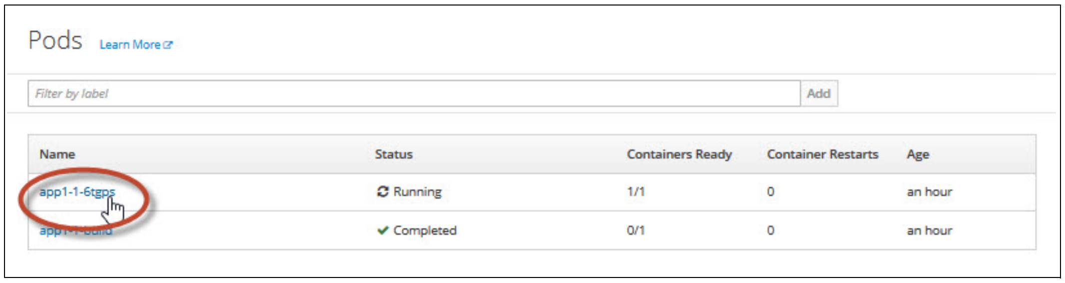
Name	Status	Containers Ready	Container Restarts	Age
app1-1-6tgps	Running	1/1	0	an hour
app1-1-build	✓ Completed	0/1	0	an hour

Figure A-13 List of pods table

A.6 Testing deployment resiliency

Complete the following steps:

1. Select the name of the running pod to display information about the pod, as shown in Figure A-14.



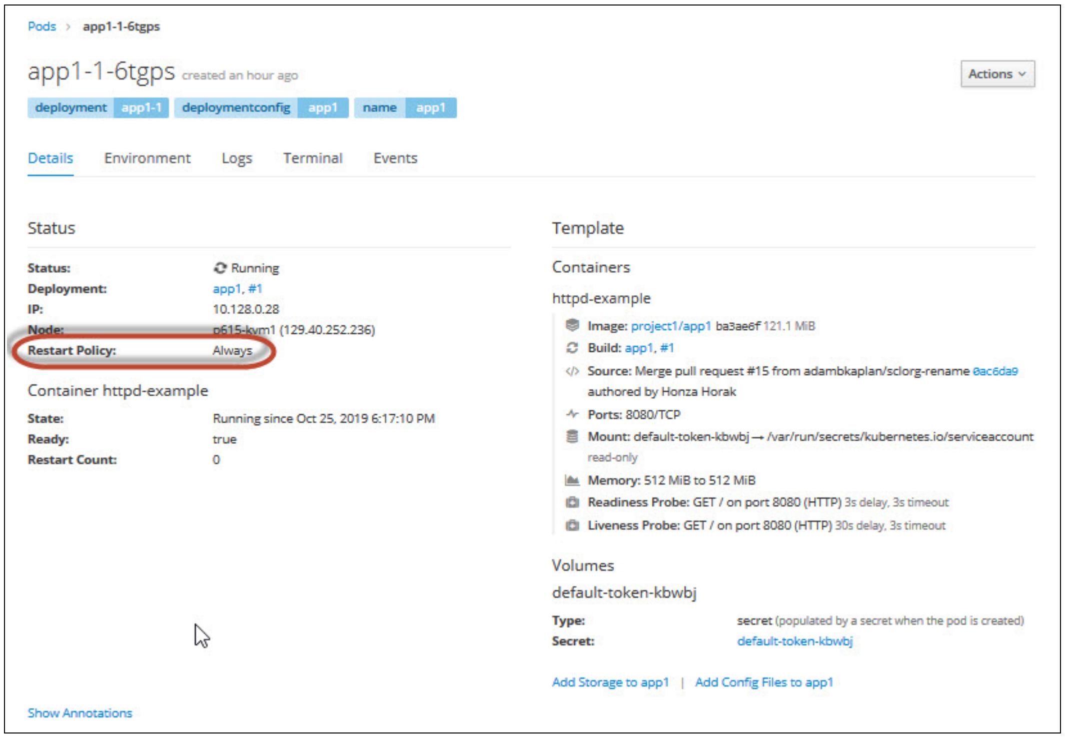
Pods Learn More

Filter by label Add

Name	Status	Containers Ready	Container Restarts	Age
app1-1-6tgps	Running	1/1	0	an hour
app1-1-build	✓ Completed	0/1	0	an hour

Figure A-14 Selecting a running pod

2. Verify the Pod restart policy of the deployment, as shown in Figure A-15, to determine how OpenShift responds when containers in that pod exit. A pod restart policy of Always attempts to restart a successfully exited container.


Pods > app1-1-6tgps

app1-1-6tgps created an hour ago

deployment app1-1 deploymentconfig app1 name app1

Status

Status: Running
Deployment: app1, #1
IP: 10.128.0.28
Node: p615-kvm1 (129.40.252.236)
Restart Policy: Always

Container httpd-example

State: Running since Oct 25, 2019 6:17:10 PM
Ready: true
Restart Count: 0

Template

Containers

httpd-example

Image: project1/app1 ba3ae6f 121.1 MiB
Build: app1, #1
Source: Merge pull request #15 from adambkaplan/sclorg-rename 0ac6da9 authored by Honza Horak
Ports: 8080/TCP
Mount: default-token-kbwbj → /var/run/secrets/kubernetes.io/serviceaccount read-only
Memory: 512 MiB to 512 MiB
Readiness Probe: GET / on port 8080 (HTTP) 3s delay, 3s timeout
Liveness Probe: GET / on port 8080 (HTTP) 30s delay, 3s timeout

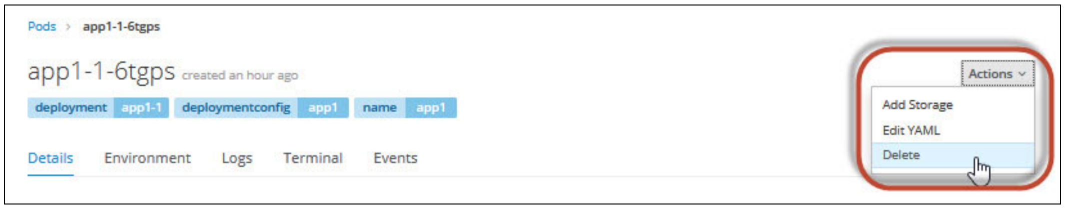
Volumes

default-token-kbwbj

Type: secret (populated by a secret when the pod is created)
Secret: default-token-kbwbj

Figure A-15 Verifying the Pod restart policy

3. To simulate a pod failure, stop the pod. In the Actions drop-down menu, select **Delete**, as shown in Figure A-16.

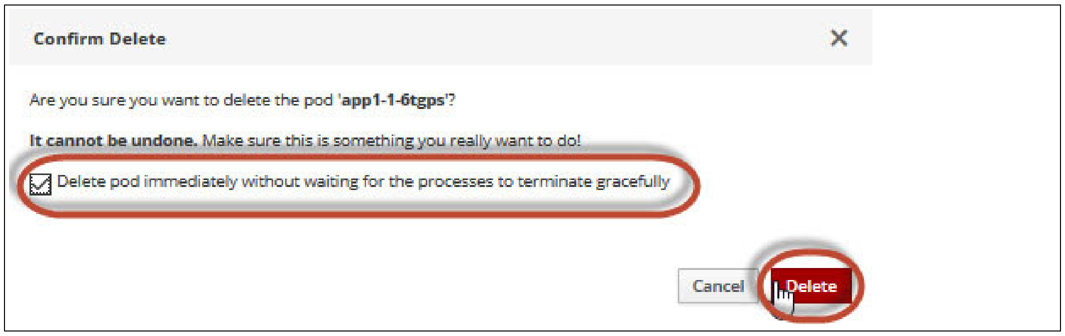

Pods > app1-1-6tgps

app1-1-6tgps created an hour ago

deployment app1-1 deploymentconfig app1 name app1

Figure A-16 Simulating a pod failure

4. Confirm the deletion of the pod, as shown in Figure A-17. Select the **Delete Pod immediately without waiting for the processes to terminate gracefully** option. Click **Delete**.



Confirm Delete

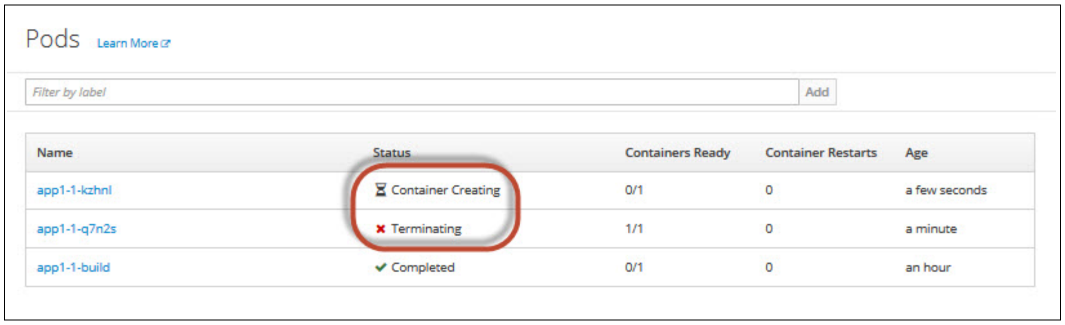
Are you sure you want to delete the pod '**app1-1-6tgps**'?

It cannot be undone. Make sure this is something you really want to do!

[✓] Delete pod immediately without waiting for the processes to terminate gracefully

Figure A-17 Confirming the deletion of the Pod

5. The pod is stopped immediately, as shown in Figure A-18. Then, the pod restarts.



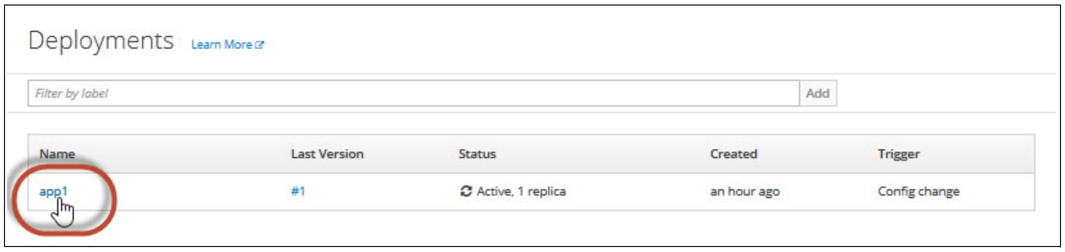
Name	Status	Containers Ready	Container Restarts	Age
app1-1-kzhnl	Container Creating	0/1	0	a few seconds
app1-1-q7n2s	Terminating	1/1	0	a minute
app1-1-build	Completed	0/1	0	an hour

Figure A-18 Pod restarting

A.7 Scaling the deployment

Complete the following steps:

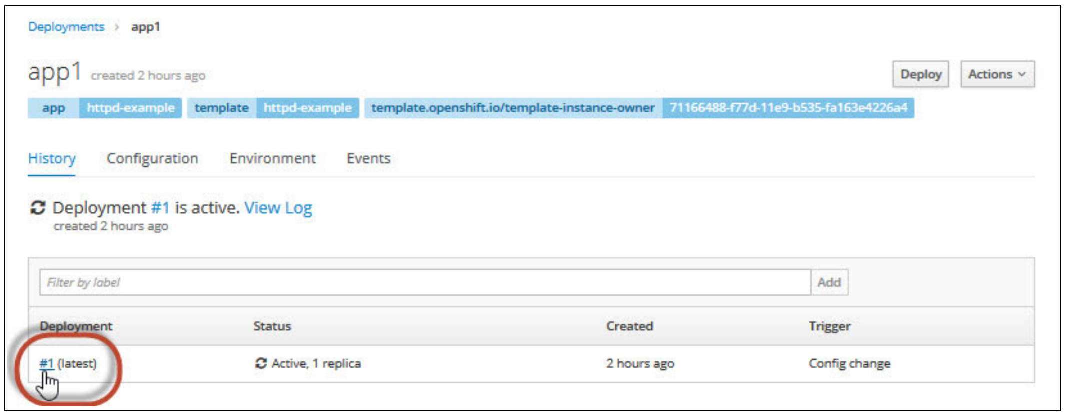
1. From the Application Console view, select **Applications** → **Deployments**.
2. Select **app1** from the list of deployments, as shown in Figure A-19.



Name	Last Version	Status	Created	Trigger
app1	#1	Active, 1 replica	an hour ago	Config change

Figure A-19 Selecting an application from the list of deployments

3. Select deployment **#1 (latest)** from the list of deployments, as shown in Figure A-20.



Deployments › app1

app1 created 2 hours ago

app | httpd-example | template | httpd-example | template.openshift.io/template-instance-owner | 71166488-f77d-11e9-b535-fa163e4226a4

Deploy | Actions

History | Configuration | Environment | Events

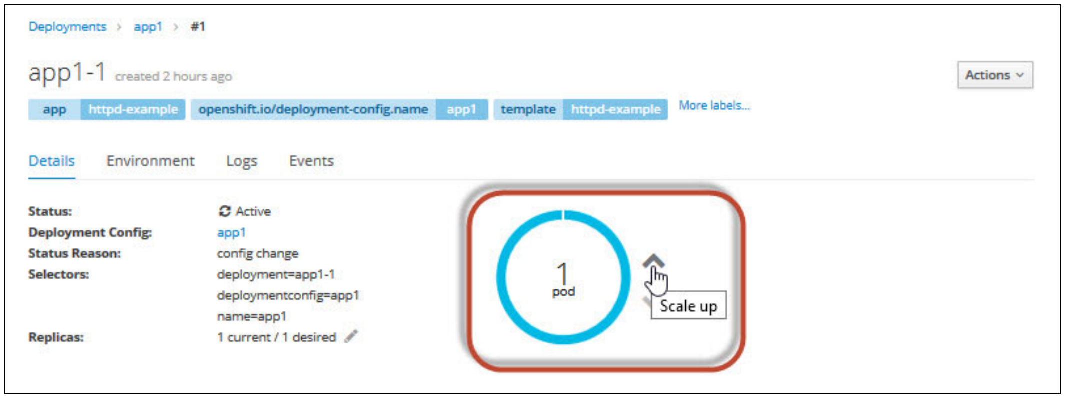
Deployment #1 is active. View Log
created 2 hours ago

Filter by label | Add

Deployment	Status	Created	Trigger
#1 (latest)	Active, 1 replica	2 hours ago	Config change

Figure A-20   Selecting a deployment

4. Scale up the pods from 1 to 2, by selecting the up arrow that is next to the pod count, as shown in Figure A-21.



Deployments › app1 › #1

app1-1 created 2 hours ago

app | httpd-example | openshift.io/deployment-config.name | app1 | template | httpd-example | More labels...

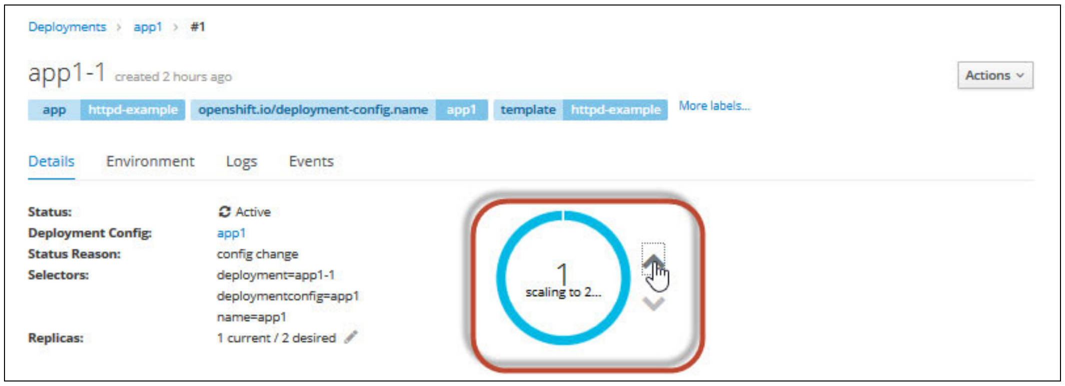
Actions

Details | Environment | Logs | Events

Status:	Active
Deployment Config:	app1
Status Reason:	config change
Selectors:	deployment=app1-1
	deploymentconfig=app1
	name=app1
Replicas:	1 current / 1 desired

Figure A-21   Selecting the up arrow to the Pod count

5. OpenShift starts another pod, as shown in Figure A-22. This process takes a moment.



Deployments › app1 › #1

app1-1 created 2 hours ago

app | httpd-example | openshift.io/deployment-config.name | app1 | template | httpd-example | More labels...

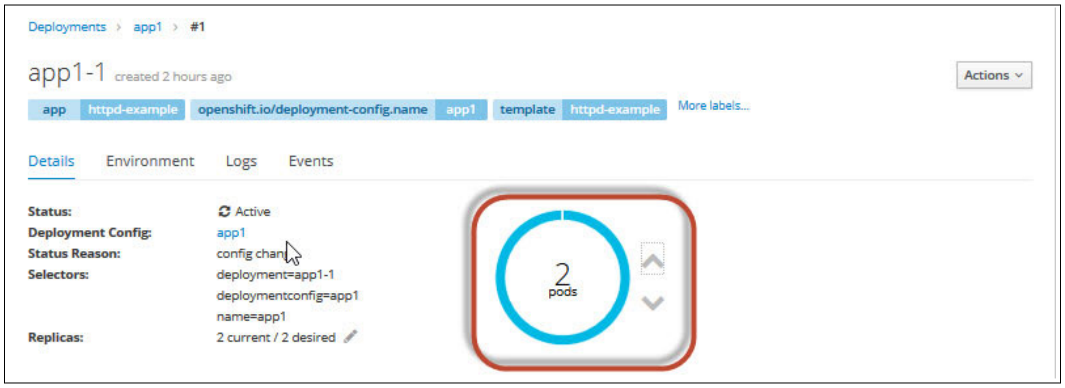
Actions

Details | Environment | Logs | Events

Status:	Active
Deployment Config:	app1
Status Reason:	config change
Selectors:	deployment=app1-1
	deploymentconfig=app1
	name=app1
Replicas:	1 current / 2 desired

Figure A-22   Scaling up a Pod

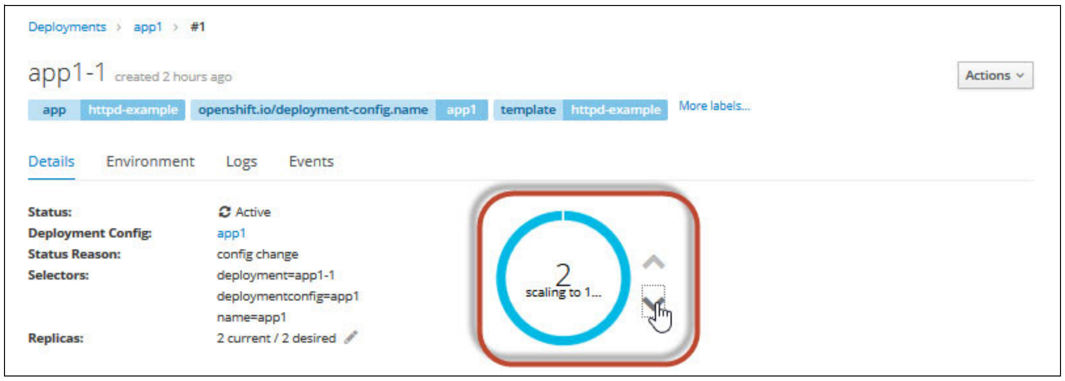
6. Verify that the second pod started, as shown in Figure A-23.


Deployments › app1 › #1
app1-1 created 2 hours ago
Actions
app httpd-example openshift.io/deployment-config.name app1 template httpd-example More labels...
Details Environment Logs Events
2 pods

Status: Active
Deployment Config: app1
Status Reason: config chan
Selectors: deployment=app1-1
deploymentconfig=app1
name=app1
Replicas: 2 current / 2 desired

Figure A-23 Verifying the number running of pods

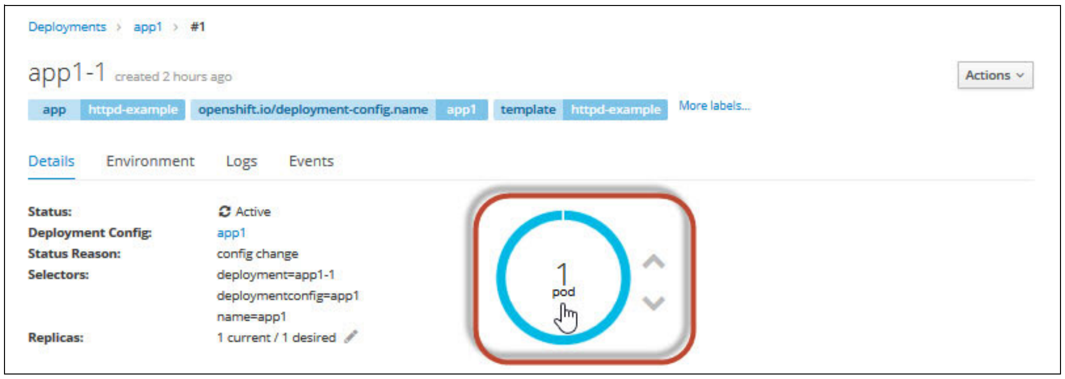
7. You can scale down the Pods from 2 to 1 by selecting the down arrow that is next to the pod count, as shown in Figure A-24.


Deployments › app1 › #1
app1-1 created 2 hours ago
Actions
app httpd-example openshift.io/deployment-config.name app1 template httpd-example More labels...
Details Environment Logs Events
2 scaling to 1...

Status: Active
Deployment Config: app1
Status Reason: config change
Selectors: deployment=app1-1
deploymentconfig=app1
name=app1
Replicas: 2 current / 2 desired

Figure A-24 Scaling down the number of pods

8. Verify that only one pod is running, as shown in Figure A-25.


Deployments › app1 › #1
app1-1 created 2 hours ago
Actions
app httpd-example openshift.io/deployment-config.name app1 template httpd-example More labels...
Details Environment Logs Events
1 pod

Status: Active
Deployment Config: app1
Status Reason: config change
Selectors: deployment=app1-1
deploymentconfig=app1
name=app1
Replicas: 1 current / 1 desired

Figure A-25 Verifying the number of Pods running

9. Log out of OpenShift, as shown in Figure A-26.

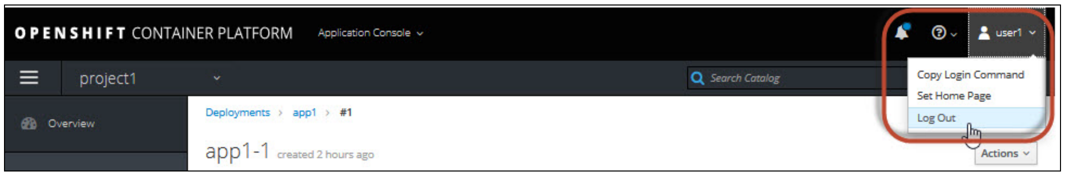


Figure A-26 Logging out from OpenShift

B

Sample lab: Deployments and workload balance

In this appendix, we describe how to deploy a single application and add a route for workload balance.

This appendix includes the following topics:

Connecting to the lab environment

To connect to the lab environment, you need access to the account credentials to your `admin` OpenShift web console and the terminal window with `root` privileges, as shown in Figure B-1.

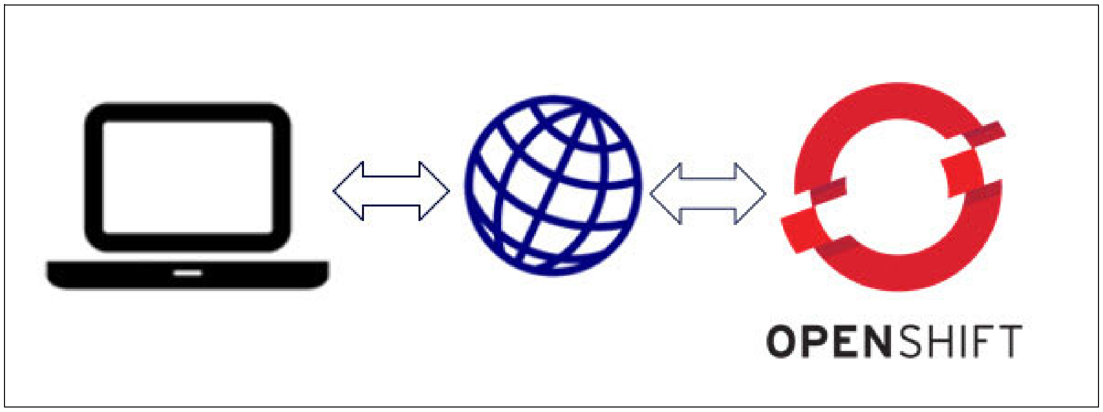


Figure B-1 Access OpenShift web console

Complete the following steps:

1. Modify the hosts file on your local computer.

   The containers that are deployed in OpenShift are on a private network within the OpenShift cluster. Accessing applications from an external connection requires network and deployment planning, which is beyond the scope of this lab exercise. In this step, you modify the hosts file on your local machine to access the deployments that were created for this lab, as shown in Example B-1.

   Example B-1 Modifying the hosts file

   ```
   IP_ADDRESS console.router.default.svc.cluster.local app2-http-git
   ```

2. Open a terminal window to the OCP machine by using a user with `root` privileges. Run the **`ssh`** command or PuTTY from your local computer.

3. Verify the release of Red Hat and other pieces of information about your operating system, as shown in Example B-2.

   Example B-2 Verifying the release of Red Hat

   ```
   $ uname -srm
   Linux 3.10.0-957.21.3.el7.ppc64le ppc64le
   ```

Creating a user and project by using the OpenShift command line

Complete the following steps:

1. Add a user that is called `user2` with password `cloudonpower`, as shown in Example B-3.

   Example B-3 Adding a user

   ```
   $ sudo adduser user2
   $ echo “cloudonpower” | sudo passwd user2 --stdin
   ```

2. Use the **htpasswd** command to update the flat files that are used to store user names and passwords for basic authentication of HTTP users. Use user `user2` with password `cloudonpower`, as shown in Example B-4.

   Example B-4 Updating password for an HTTP user

   ```
   $ sudo htpasswd -b /etc/origin/master/htpasswd user2 cloudonpower
   Adding password for user user2
   ```

3. Assign the new user cluster administration rights by logging in as the cluster administrator (`root` privileges). Then, into the `default` project by using the OpenShift CLI command (**oc**), as shown in Example B-5.

   Example B-5 Logging in to the default project

   ```
   $ oc login -u user618 -n default
   Server [https://localhost:8443]:
   The server is using a certificate that does not match its hostname: x509:
   certificate is valid for kubernetes, kubernetes.default,
   kubernetes.default.svc, kubernetes.default.svc.cluster.local, openshift,
   openshift.default, openshift.default.svc, openshift.default.svc.cluster.local,
   p618-kvm1.cecc.ihost.com, 129.40.253.20, 172.30.0.1, not localhost
   You can bypass the certificate check, but any data you send to the server could
   be intercepted by others.
   Use insecure connections? (y/n): y

   Authentication required for https://localhost:8443 (openshift)
   Username: user618
   Password: *********
   Login successful.

   You have access to the following projects and can switch between them with 'oc
   project <projectname>':

     * default
       kube-public
       kube-service-catalog
       kube-system
       management-infra
       openshift
       openshift-ansible-service-broker
       openshift-console
       openshift-infra
       openshift-logging
       openshift-monitoring
   ```

```
    openshift-node
    openshift-sdn
    openshift-template-service-broker
    openshift-web-console

Using project "default".
Welcome! See 'oc help' to get started.
```

4. List all defined OCP users by using the **`oc get`** command, as shown in Example B-6.

Example B-6 Listing all currently defined OCP users

```
$ oc get user
NAME      UID                                    FULL NAME   IDENTITIES
user618   18e66d09-fc00-11e9-a4e5-fa163e187e44
htpasswd_auth:user618
```

5. Use the **`oc policy`** command to assign the `user2` user cluster admin rights, as shown in Example B-7.

Example B-7 Assigning user cluster admin rights

```
$ oc adm policy add-cluster-role-to-user cluster-admin user2
Warning: User 'user2' not found
cluster role "cluster-admin" added: "user2"
```

6. Log out by using the **`oc logout`** command, as shown in Example B-8.

Example B-8 Logging out from OpenShift

```
$ oc logout
Logged "user618" out on "https://localhost:8443"
```

7. Log back into OpenShift CLI as user `user2`, as shown in Example B-9.

Example B-9 Logging in to OpenShift

```
$ oc login -u user2
Authentication required for https://localhost:8443 (openshift)
Username: user2
Password: *********
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

  * default
    kube-public
    kube-service-catalog
    kube-system
    management-infra
    openshift
    openshift-ansible-service-broker
    openshift-console
    openshift-infra
    openshift-logging
    openshift-monitoring
    openshift-node
    openshift-sdn
    openshift-template-service-broker
    openshift-web-console

Using project "default".
```

8. List all defined OCP users by using the **`oc get`** command. Now, you see that `user2` is included in the list, as shown in Example B-10.

Example B-10 Listing all defined OCP users

```
$ oc get user
NAME      UID                                    FULL NAME   IDENTITIES
user2     9099b168-fc00-11e9-a4e5-fa163e187e44 htpasswd_auth:user2
user618   18e66d09-fc00-11e9-a4e5-fa163e187e44 htpasswd_auth:user618
```

9. Use the **`oc get project`** command to list all defined OpenShift projects, as shown in Example B-11.

Example B-11 Listing all defined OpenShift projects

```
$ oc get project
NAME                                DISPLAY NAME   STATUS
default                                            Active
kube-public                                        Active
kube-service-catalog                               Active
kube-system                                        Active
management-infra                                   Active
openshift                                          Active
openshift-ansible-service-broker                   Active
openshift-console                                  Active
openshift-infra                                    Active
openshift-logging                                  Active
openshift-monitoring                               Active
openshift-node                                     Active
openshift-sdn                                      Active
openshift-template-service-broker                  Active
openshift-web-console                              Active
```

10. Use the **`oc new-project`** command to create a project that is called `project2`, as shown in Example B-12.

Example B-12 Creating a project

```
$ oc new-project project2
Now using project "project2" on server "https://localhost:8443".

You can add applications to this project with the 'new-app' command. For
example, try:

    oc new-app centos/ruby-25-centos7~https://github.com/sclorg/ruby-ex.git

to build a new example application in Ruby.
```

Logging in to the OpenShift web console

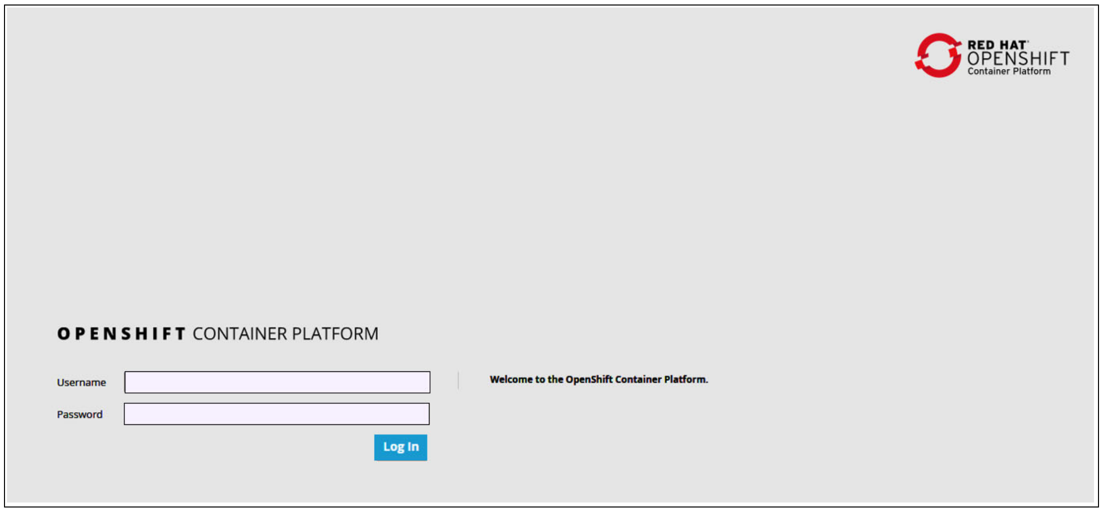
Complete the following steps:

1. From your local computer, open a web browser and browse to the master node by using port `8443`, as shown in Example B-13.

   Example B-13 Logging in to the OpenShift web console

   ```
   https://<master_ip>:8443
   ```

2. Log in as `user2`, as shown in Figure B-2.


RED HAT OPENSHIFT Container Platform
OPENSHIFT CONTAINER PLATFORM
Username
Password
Log In

Welcome to the OpenShift Container Platform.

Figure B-2 Logging in to the OpenShift web console

The OpenShift web console is displayed, as shown in Figure B-3.

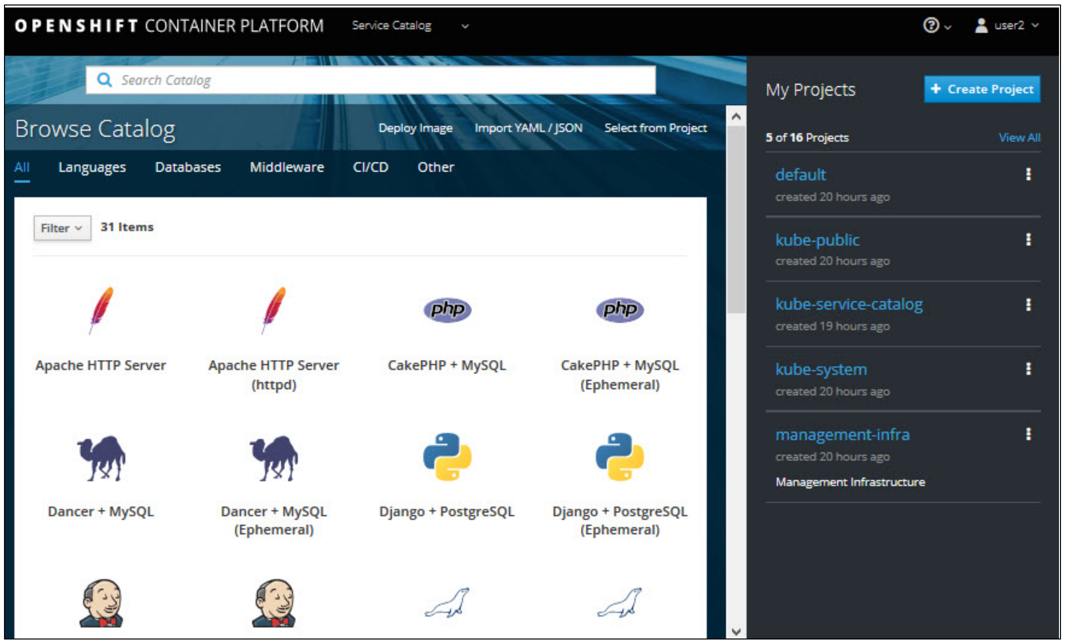


Figure B-3 Service Catalog View

Deploying an NGINX server by using the OpenShift web console

Complete the following steps:

1. In the Service Catalog view, click **Nginx HTTP server and reverse proxyB,** as shown in Figure B-4.

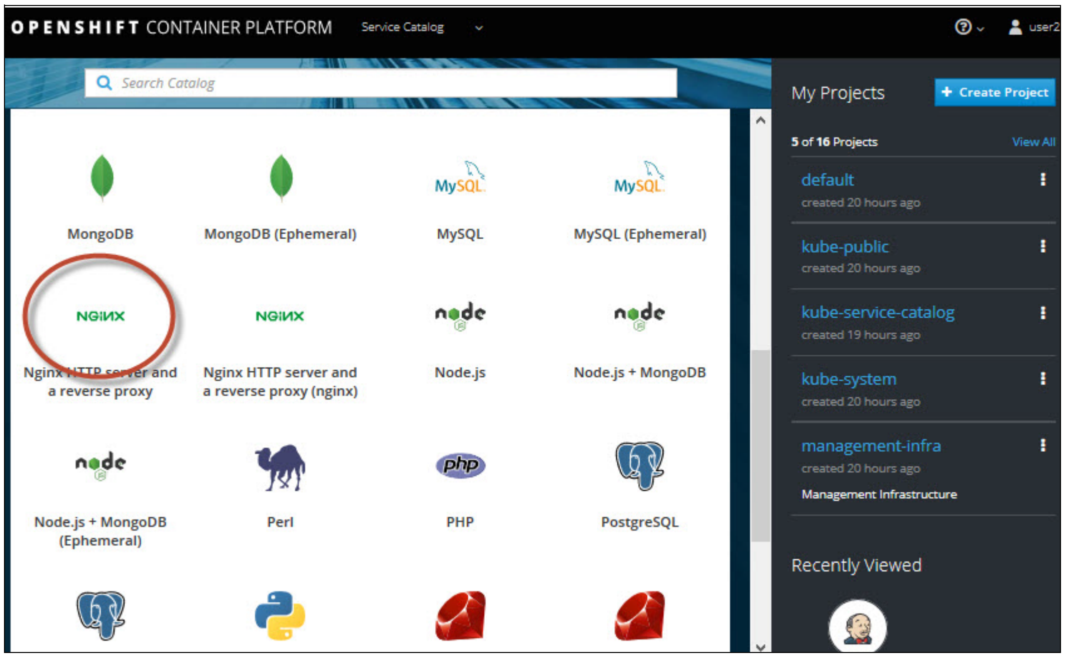
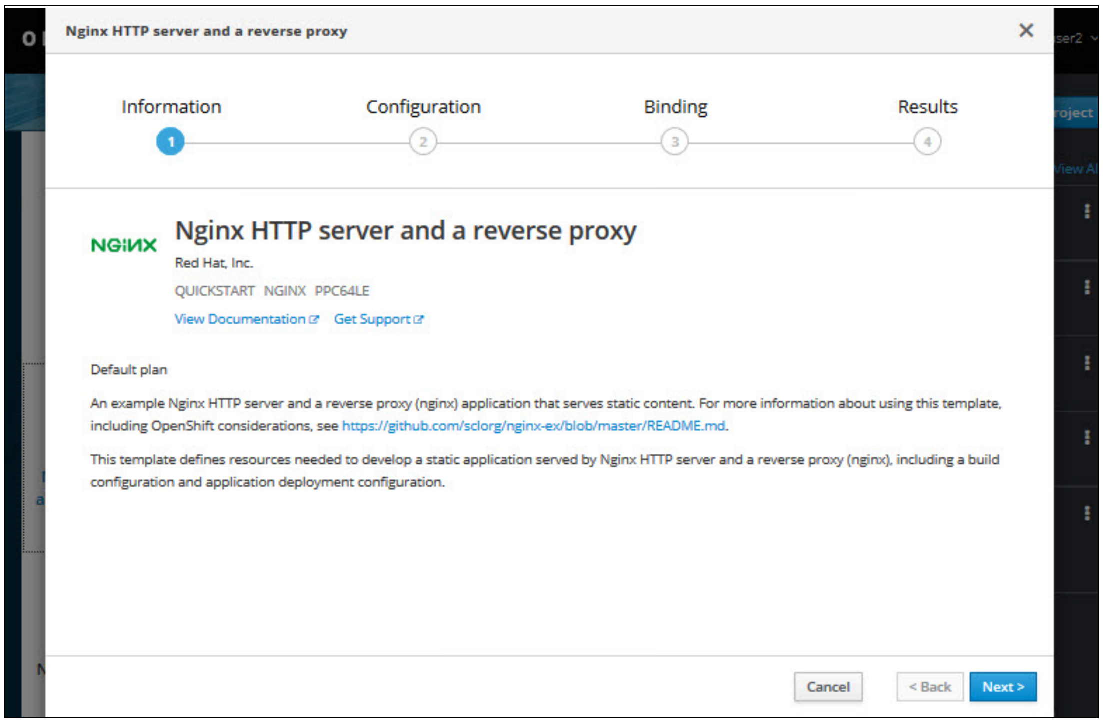


Figure B-4 Selecting the NGINX Server

2. In the Information window, click **Next**, as shown in Figure B-5.



Nginx HTTP server and a reverse proxy

Red Hat, Inc.

QUICKSTART NGINX PPC64LE

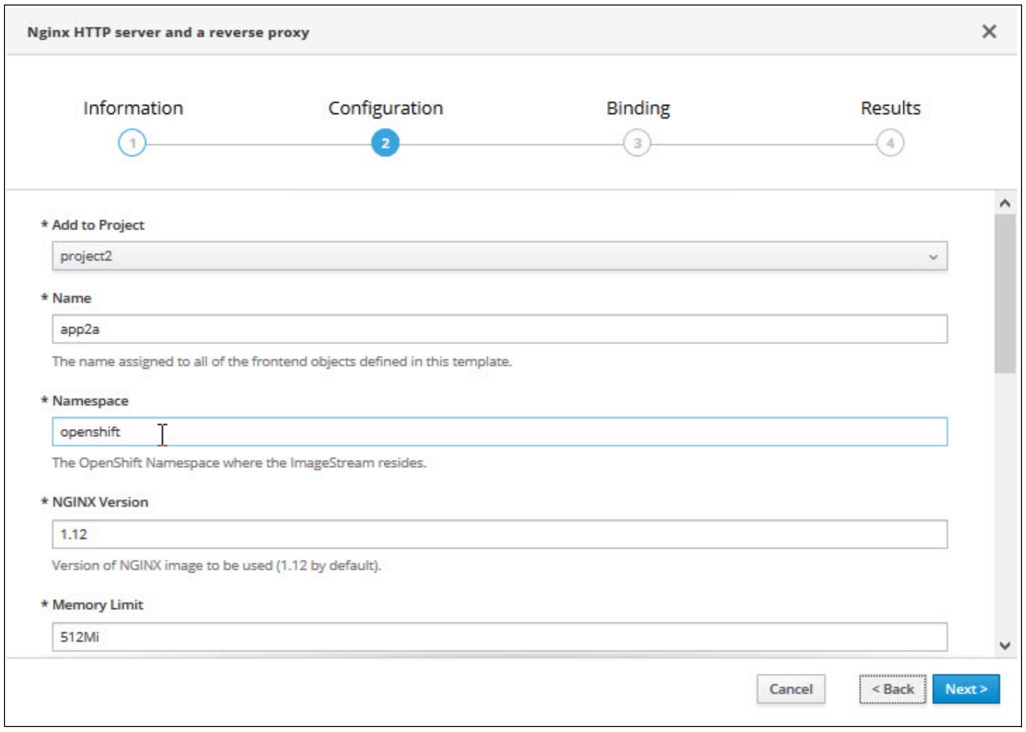
Default plan

An example Nginx HTTP server and a reverse proxy (nginx) application that serves static content. For more information about using this template, including OpenShift considerations, see https://github.com/sclorg/nginx-ex/blob/master/README.md.

This template defines resources needed to develop a static application served by Nginx HTTP server and a reverse proxy (nginx), including a build configuration and application deployment configuration.

Figure B-5 Information window

3. In the Configuration window, configure the following settings, as shown in Figure B-6:
   – Add to Project: `project2`
   – Name: `app2a`



* Add to Project: project2

* Name: app2a

The name assigned to all of the frontend objects defined in this template.

* Namespace: openshift

The OpenShift Namespace where the ImageStream resides.

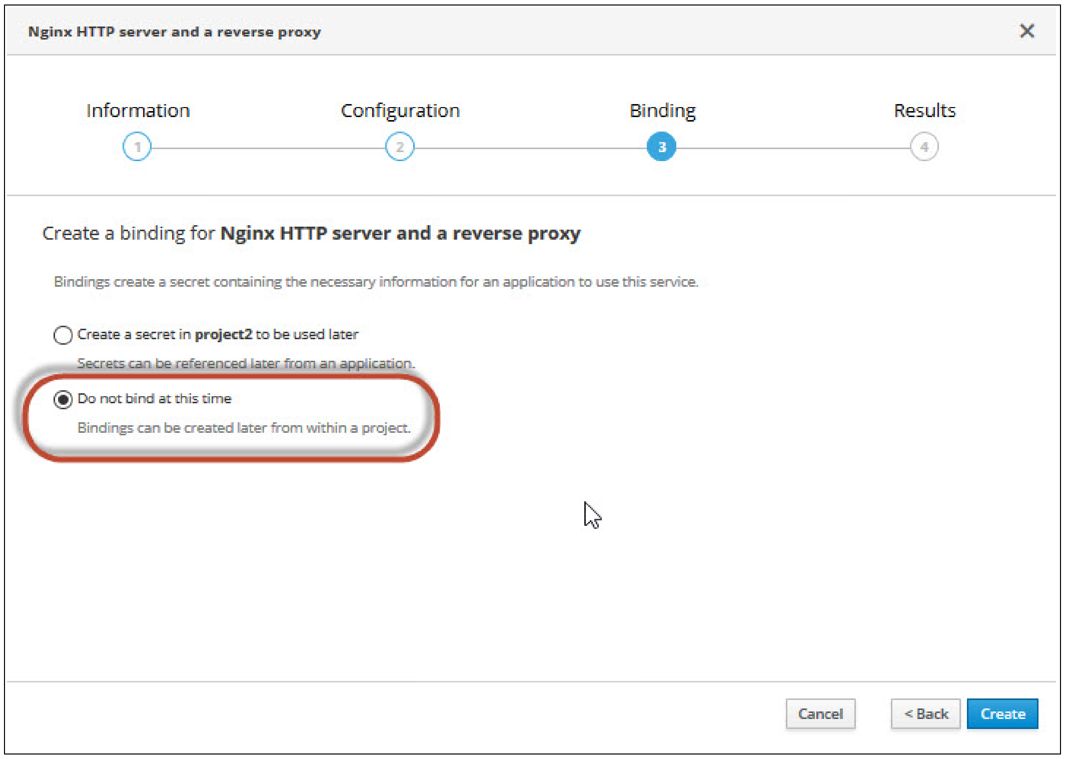
* NGINX Version: 1.12

Version of NGINX image to be used (1.12 by default).

* Memory Limit: 512Mi

Figure B-6 Configuration window

4. To complete the deployment, click **Next**. The Binding window is displayed, as shown in Figure B-7. Select the **Do not bind at this time** option. Click **Create** to continue the deployment.



Create a binding for **Nginx HTTP server and a reverse proxy**

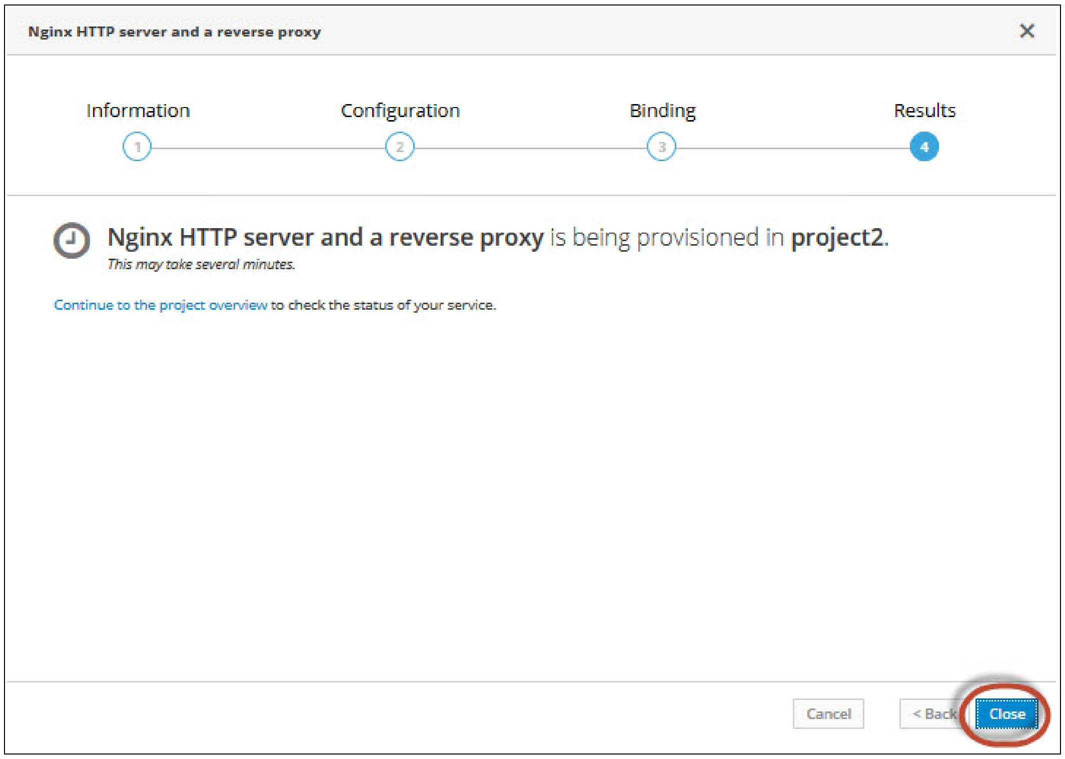
Bindings create a secret containing the necessary information for an application to use this service.

( ) Create a secret in **project2** to be used later
Secrets can be referenced later from an application.

[✓ Do not bind at this time]
Bindings can be created later from within a project.

Figure B-7 Binding window

5. The Results window is displayed and shows the progress of the deployment (see Figure B-8). To continue, click **Close**.



Nginx HTTP server and a reverse proxy is being provisioned in project2.

This may take several minutes.

Continue to the project overview to check the status of your service.

Figure B-8 Results window

Deploying a second NGINX server by using the OpenShift web console

1. In the Service Catalog view, click **Nginx HTTP server and reverse proxy**, as shown in Figure B-9.

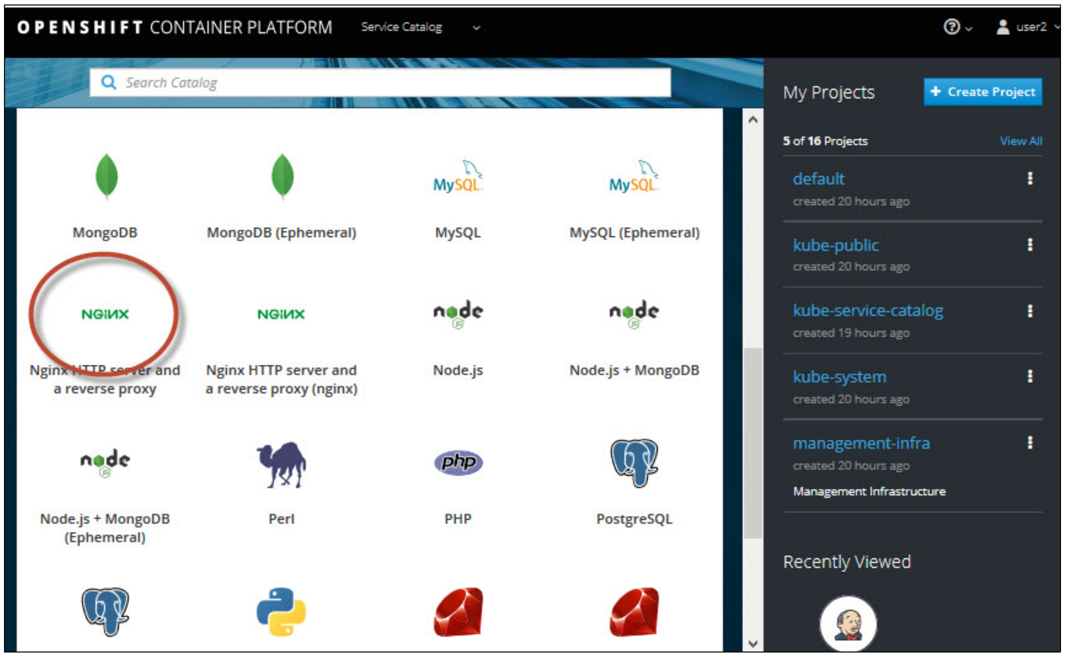
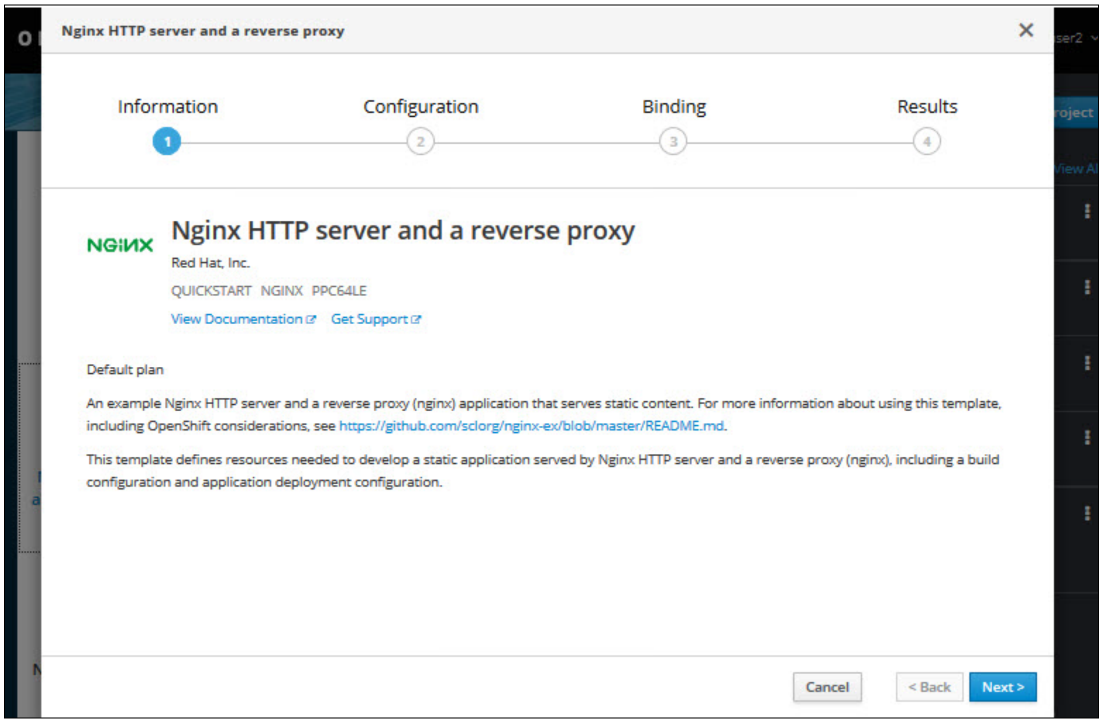


Figure B-9   Selecting NGINX Server

2. In the Information window (see Figure B-10), click **Next**.



Nginx HTTP server and a reverse proxy

Red Hat, Inc.

QUICKSTART NGINX PPC64LE

View Documentation Get Support

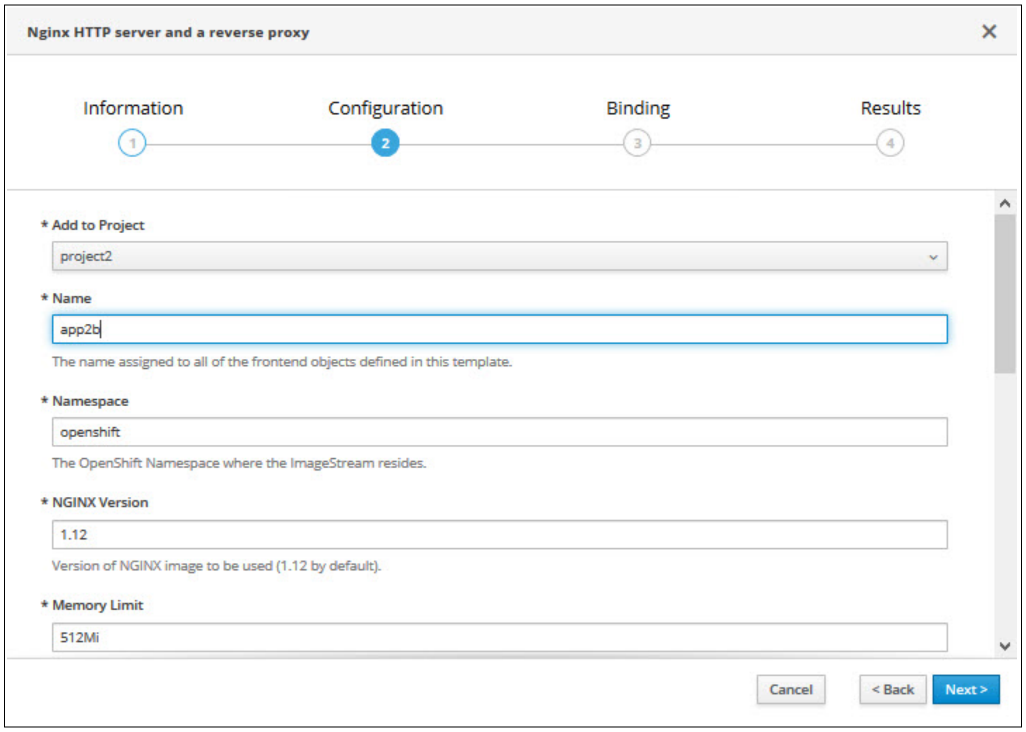
Default plan

An example Nginx HTTP server and a reverse proxy (nginx) application that serves static content. For more information about using this template, including OpenShift considerations, see https://github.com/sclorg/nginx-ex/blob/master/README.md.

This template defines resources needed to develop a static application served by Nginx HTTP server and a reverse proxy (nginx), including a build configuration and application deployment configuration.

Figure B-10   Information window

3. In the Configuration window (see Figure B-11), configure the following settings:
   - Add to Project: `project2`
   - Name: `app2b`



* Add to Project
project2

* Name
app2b
The name assigned to all of the frontend objects defined in this template.

* Namespace
openshift
The OpenShift Namespace where the ImageStream resides.

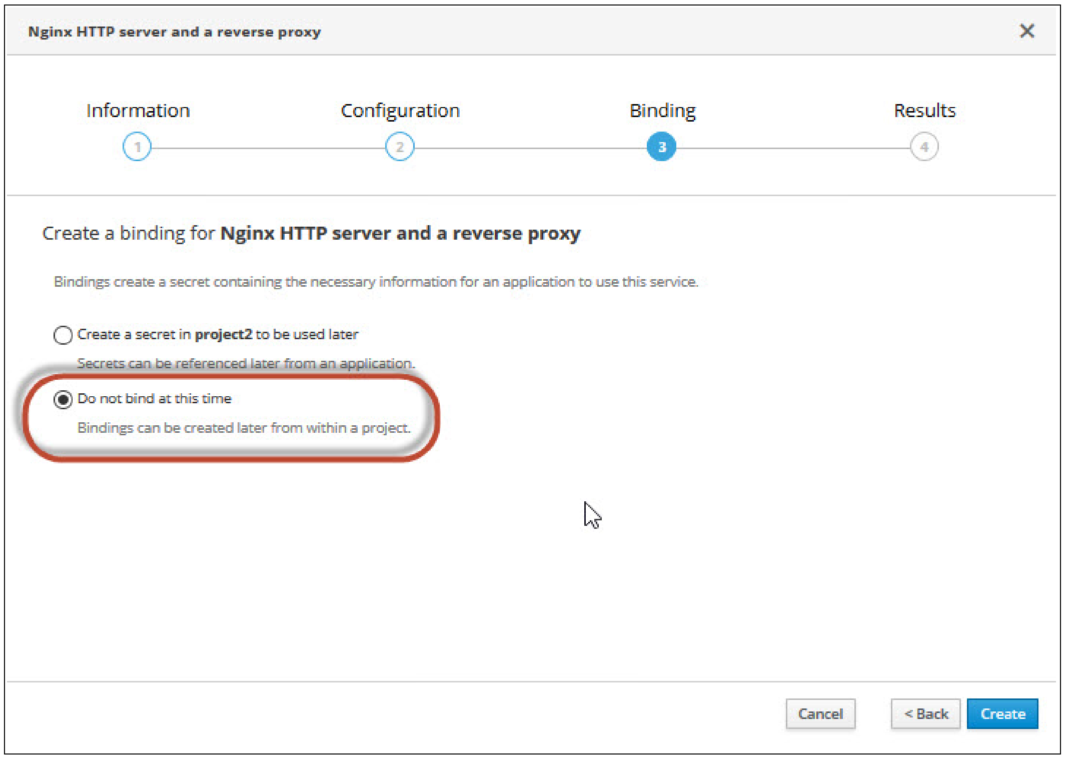
* NGINX Version
1.12
Version of NGINX image to be used (1.12 by default).

* Memory Limit
512Mi

Figure B-11 Configuration window

4. To complete the deployment, click **Next**.

5. The Binding window is displayed, as shown in Figure B-12. Select the **Do not bind at this time** option. Click **Create** to continue the deployment.


Nginx HTTP server and a reverse proxy

Create a binding for **Nginx HTTP server and a reverse proxy**

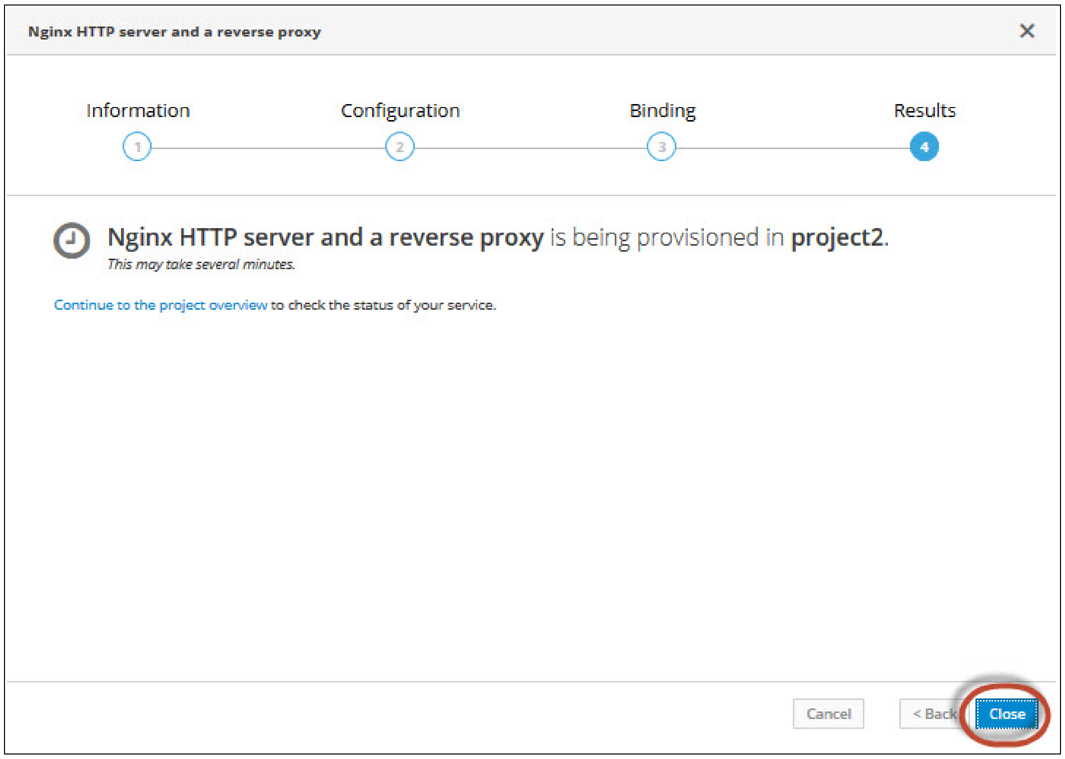
Bindings create a secret containing the necessary information for an application to use this service.

○ Create a secret in **project2** to be used later
Secrets can be referenced later from an application.

● Do not bind at this time
Bindings can be created later from within a project.

Figure B-12 Binding window

6. The Results window is displayed and shows the progress of the deployment. To continue, click **Close**, as shown in Figure B-13.



Nginx HTTP server and a reverse proxy is being provisioned in **project2**.

This may take several minutes.

Continue to the project overview to check the status of your service.

Figure B-13 Results window

7. Verify that two Nginx deployments are available that are in the `Active` state. From the Application Console view (see Figure B-14), browse to the `project2` application deployments.

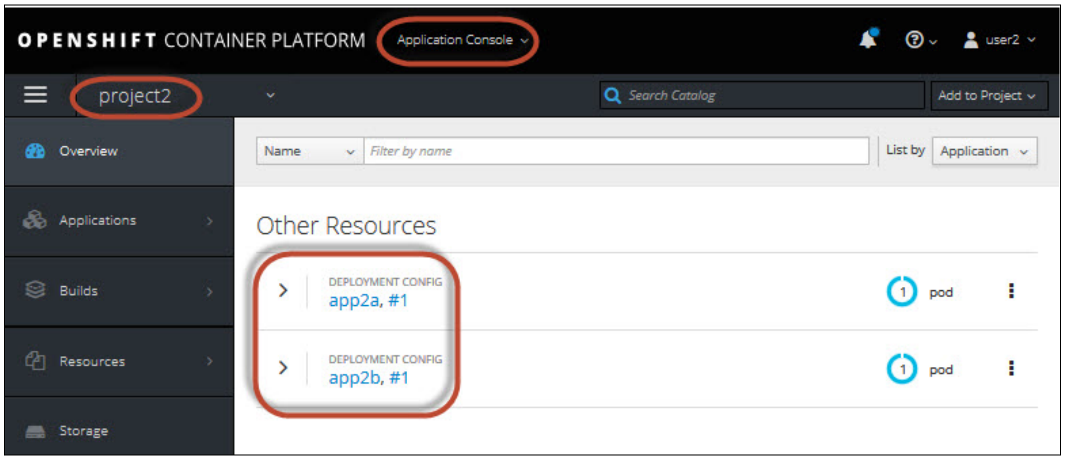


Figure B-14 Verifying application deployments

Customizing the index.test file of the NGINX instances

Complete the following steps:

1. From the web console, Application Console view (see Figure B-15), select **Applications** → **Pods**.

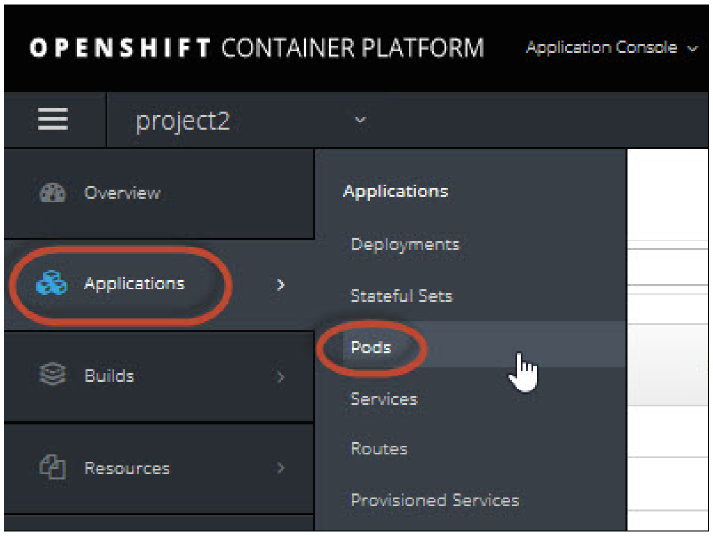
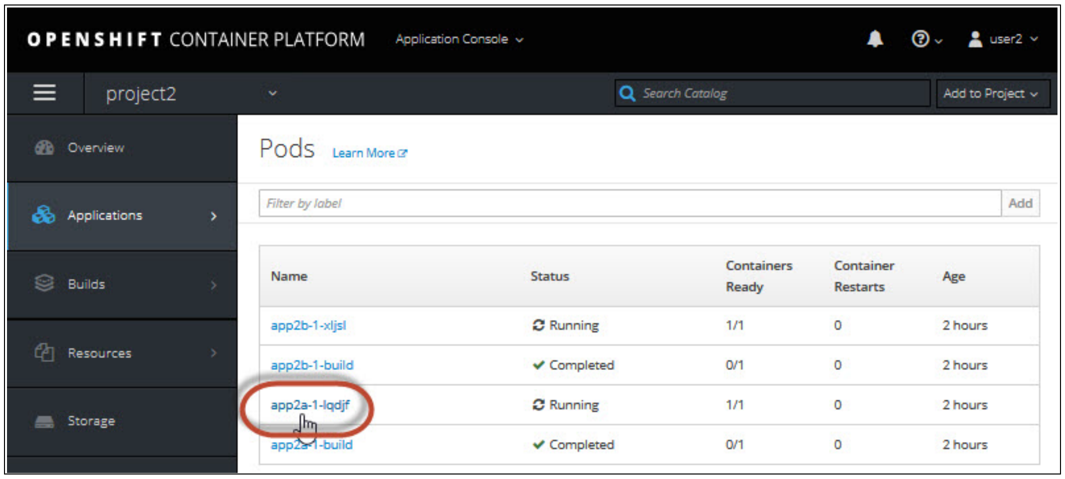


Figure B-15 Application Console view

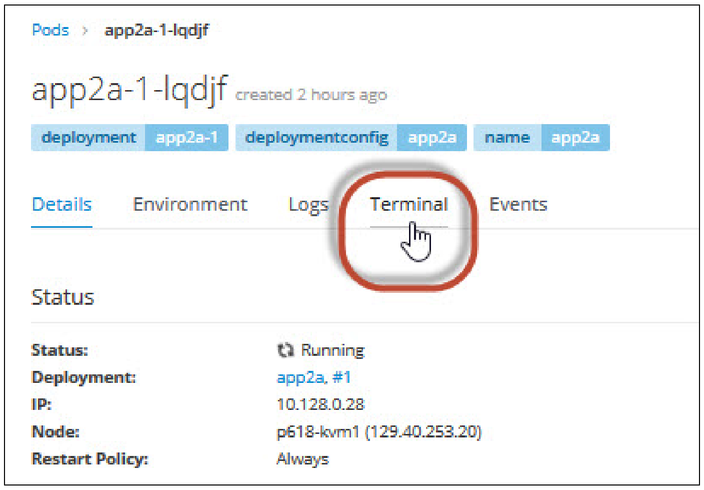
2. Select the name of the running Pod for `app2a-1-`*yyyy* (as shown in Figure B-16) where *yyyy* is the unique identifier for the running pod.



Name	Status	Containers Ready	Container Restarts	Age
app2b-1-xljsl	Running	1/1	0	2 hours
app2b-1-build	✓ Completed	0/1	0	2 hours
app2a-1-lqdjf	Running	1/1	0	2 hours
app2a-1-build	✓ Completed	0/1	0	2 hours

Figure B-16 Selecting the running Pod

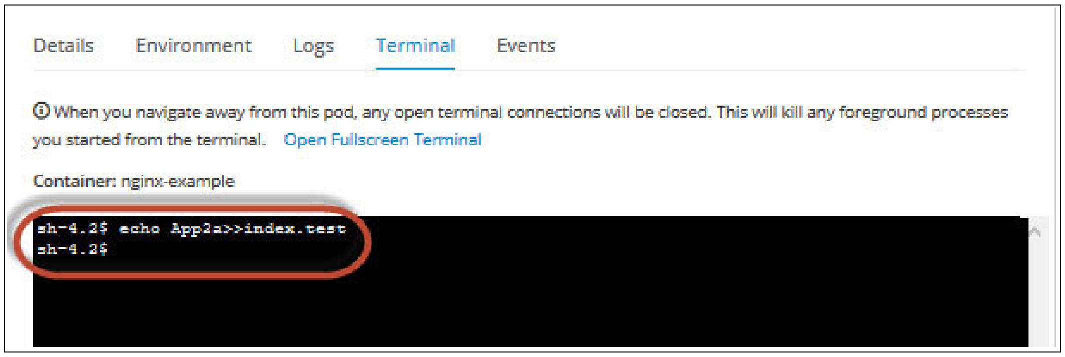
3. Select the **Terminal** tab, as shown in Figure B-17.



Status: Running
Deployment: app2a, #1
IP: 10.128.0.28
Node: p618-kvm1 (129.40.253.20)
Restart Policy: Always

Figure B-17 Selecting the terminal tab

4. In the Terminal window, run the **`echo App2a>>index.test`** command, as shown in Figure B-18.



When you navigate away from this pod, any open terminal connections will be closed. This will kill any foreground processes you started from the terminal.

Container: nginx-example

```
sh-4.2$ echo App2a>>index.test
sh-4.2$
```

Figure B-18 Terminal window view

5. From the web console, Application Console view (as shown in Figure B-19), select **Applications** → **Pods**.

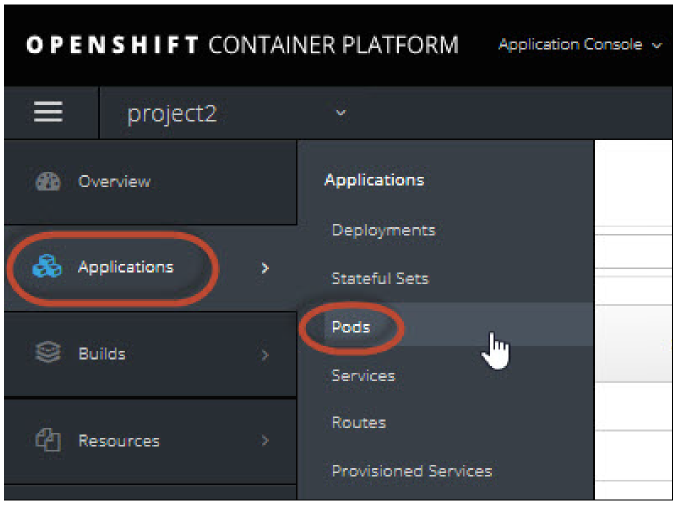
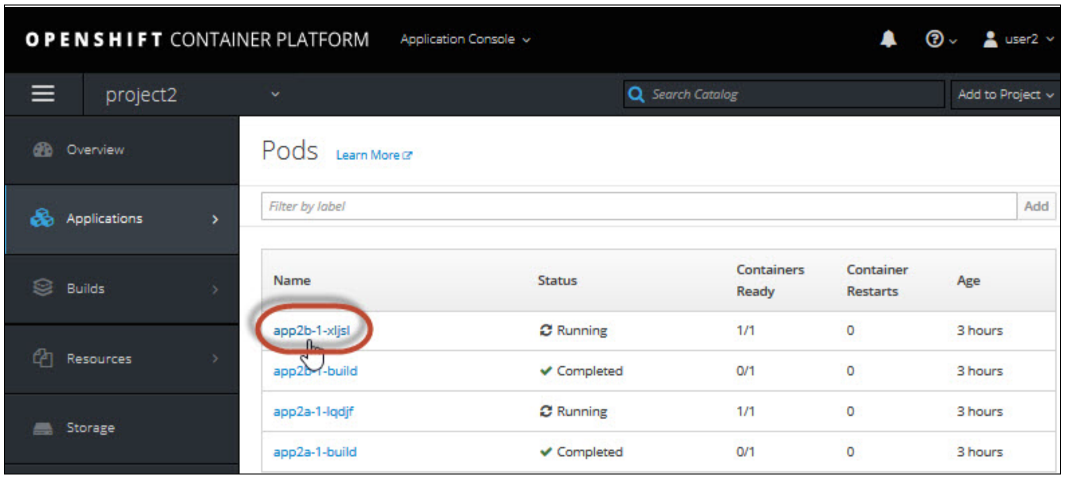


Figure B-19 Application Console view

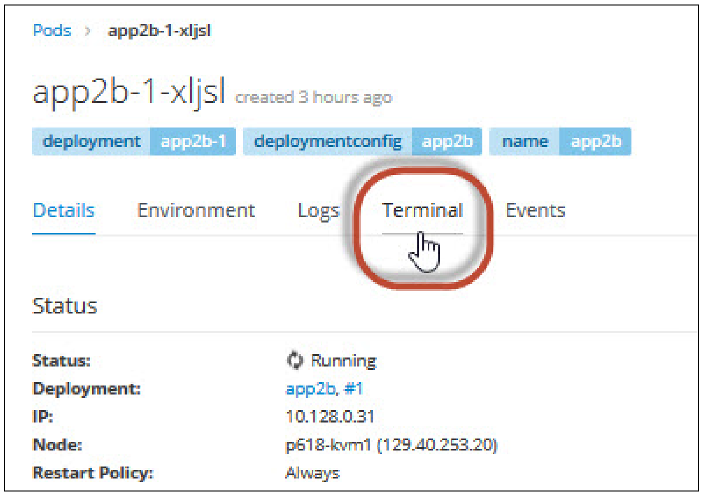
6. Select the name of the running Pod for `app2b-1-yyyy` (as shown in Figure B-20) where *yyyy* is the unique identifier for the running pod.



Name	Status	Containers Ready	Container Restarts	Age
app2b-1-xljsl	Running	1/1	0	3 hours
app2b-1-build	✓ Completed	0/1	0	3 hours
app2a-1-lqdjf	Running	1/1	0	3 hours
app2a-1-build	✓ Completed	0/1	0	3 hours

Figure B-20 Selecting the running pod

7. Select the **Terminal** tab, as shown in Figure B-21.


Pods > app2b-1-xljsl

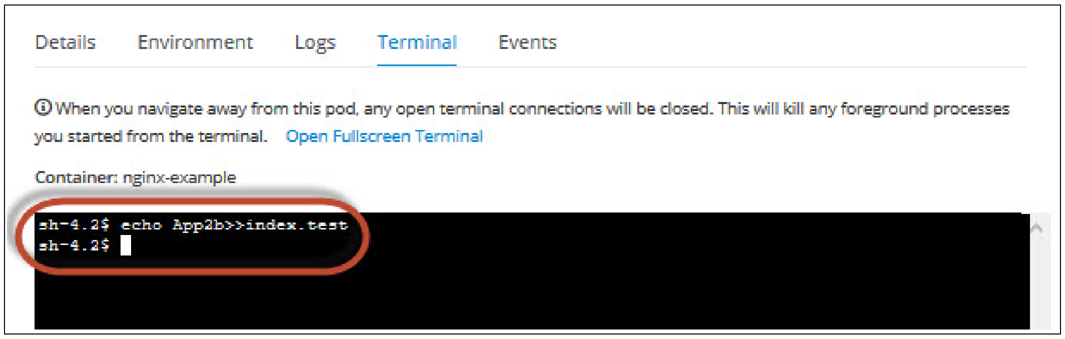
app2b-1-xljsl created 3 hours ago

Status

Status: Running
Deployment: app2b, #1
IP: 10.128.0.31
Node: p618-kvm1 (129.40.253.20)
Restart Policy: Always

Figure B-21 Selecting the terminal tab

8. In the Terminal window, run the `echo App2b>>index.test` command, as shown in Figure B-22.



When you navigate away from this pod, any open terminal connections will be closed. This will kill any foreground processes you started from the terminal. Open Fullscreen Terminal

Container: nginx-example

```
sh-4.2$ echo App2b>>index.test
sh-4.2$
```

Figure B-22 Terminal window view

Creating a route to balance the network traffic between the two NGINX instances

Complete the following steps:

1. Modify the `/etc/hosts` file of your OpenShift system to include the entry `IP_ADDRESS` with an alias of `app2-http-git` (the hostname of the service), as shown in Example B-14.

   Example B-14 Modifying the /etc/hosts file

   ```
   $ tail /etc/hosts
   129.40.253.20 app2-http-git
   ```

2. From the web console, Application Console view (as shown in Figure B-23), select **Applications** → **Routes**.

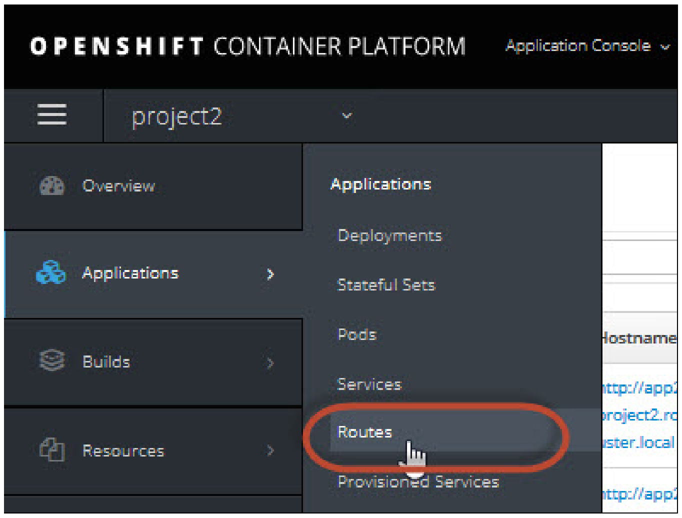
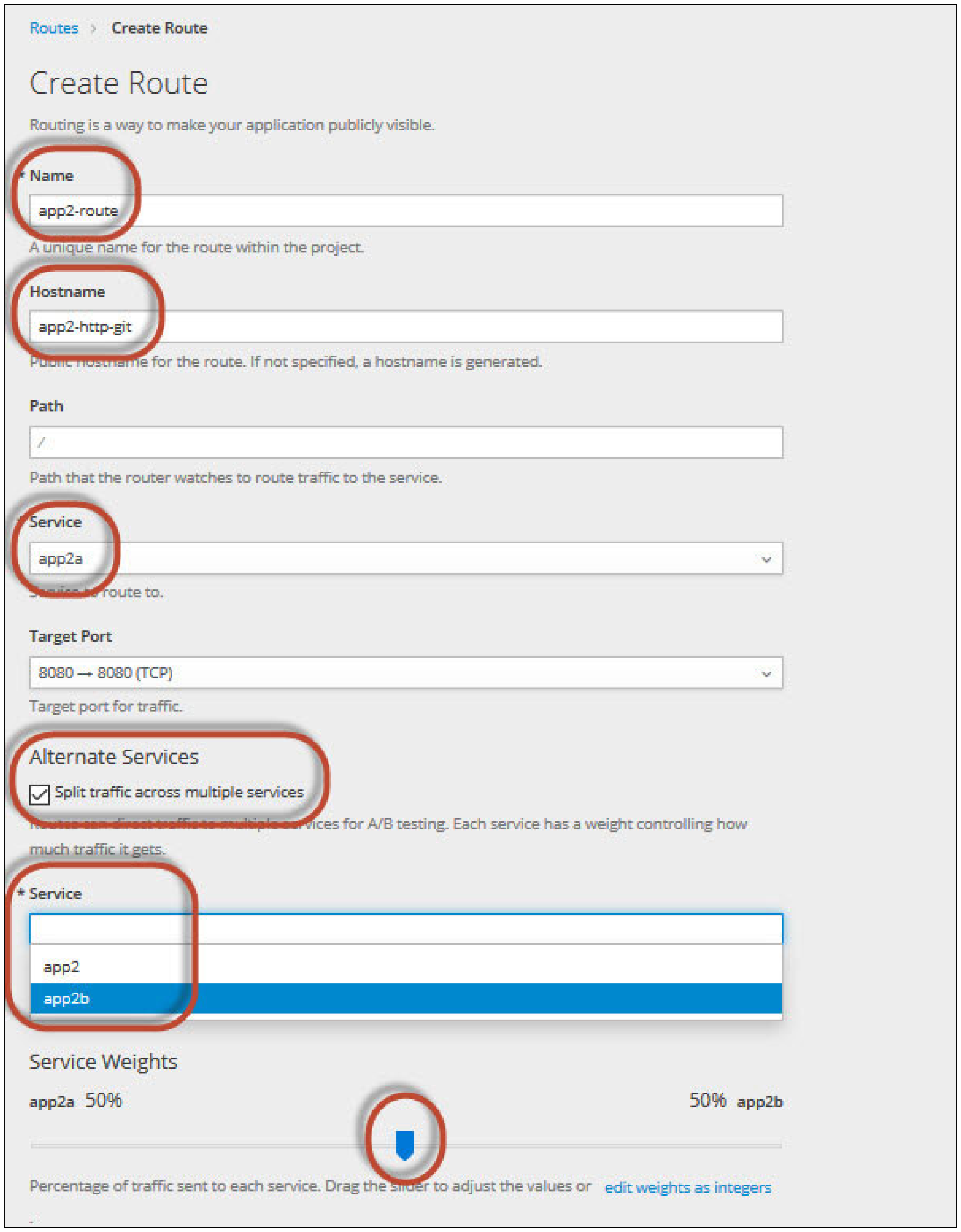


Figure B-23 Application Console view: Routes

3. In the Create Route content pane (as shown in Figure B-24), configure the following settings:
   - Name: `app2-route`
   - Hostname: `app2-http-git`
   - Service: `app2a`
   - Alternative Services: `Divide traffic across multiple servers`
   - Service: `app2b`
   - Service Weights: `50%/50%`



Create Route

Routing is a way to make your application publicly visible.

* Name
app2-route
A unique name for the route within the project.

Hostname
app2-http-git
Public hostname for the route. If not specified, a hostname is generated.

Path
/
Path that the router watches to route traffic to the service.

* Service
app2a
Service to route to.

Target Port
8080 → 8080 (TCP)
Target port for traffic.

Alternate Services
☑ Split traffic across multiple services
Routes can direct traffic to multiple services for A/B testing. Each service has a weight controlling how much traffic it gets.

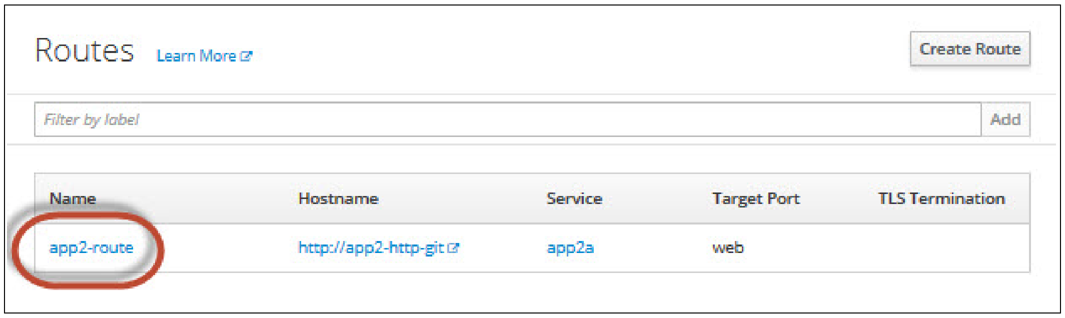
* Service
app2
app2b

Service Weights
app2a 50% 50% app2b

Percentage of traffic sent to each service. Drag the slider to adjust the values or edit weights as integers

Figure B-24 Configuring a route

4. Click **Create** to create the route, as shown in Figure B-25.



Name	Hostname	Service	Target Port	TLS Termination
app2-route	http://app2-http-git	app2a	web	

Figure B-25 Creating a route

Testing load balancing across NGINX instances

Complete the following steps:

1. Open a terminal window to the OCP machine with a user with `root` privileges. Use the **`ssh`** command or PuTTY from your local computer.
2. Run the script as shown in Example B-15 to test out the load balancing. The **`wget`** command uses a capital letter O:

```
$ for i in 1 2 3 4 5 6 7 8 9 10
do
wget -q http://app2-http-git/ -O index.test
grep App index.test
done
```

Example B-15 Running a script to test load balancing

```
$ for i in 1 2 3 4 5 6 7 8 9 10; do   wget -q http://app2-http-git/ -O
index.test;  grep App2 index.test; done
App2a
App2b
App2a
App2b
App2a
App2b
App2a
App2b
App2a
App2b
```

From the command output, you can see that the route alternated between `App2a` and `App2b` at a rate of 50%.

3. Close the terminal window and log out of the OpenShift console.

C

Seamless application movement across multicloud environments

This appendix provides an overview of a homegrown network solution. This solution is used to transparently link the running service from any cloud to the name that is configured to use for the multicloud case.

Many other industry solutions (proxies, load balancers, or traffic managers) are available and can be used, considering they also connect to the Kubernetes APIs.

This scenario shows that seamless multicloud can be achieved, even with a simple manual solution.

This appendix includes the following topics:

Network tunneling for MongoDB

A common solution that is present (even on the MongoDB Compass client) is the SSH tunneling as a means of accessing the server. The intention is that the script takes the SSH tunnel and points to the correct cloud.

This example uses the CLI commands, although Kubernetes APIs also can be used. Example C-1 shows the script that was used to complete the seamless application connection to the MongoDB database.

Example C-1 Script that looks for running MongoDB Pod and tunneling to correct cloud

```
AWSMASTERIP=3.15.11.140
POWERMASTERIP=10.108.98.213
ssh -M -S mongoconnect -fNT -L 27017:$POWERMASTERIP:32767 localhost

while true
do
TESTVAR=$(ssh root@$POWERMASTERIP -i /home/danielsc/.ssh/id_aws kubectl get pod
appmongo-ibm-mongodb-dev -o custom-columns=:{status.phase} -n default
2>>/dev/null)
if [ $? -eq 0 ]
   then
        if [ $(cat /proc/"$(ssh -S /home/danielsc/mongo/mongoconnect -O check
localhost 2>&1 |cut -d= -f2 | sed 's/)//')"/cmdline|cut -d: -f2) ==
"$POWERMASTERIP" ]
                then
                        TESTVAR=$(echo -n $TESTVAR)
                        echo "Pod status is $TESTVAR"
                        echo "Tunnel already connected to IBM correctly"
                else
                        ssh -S mongoconnect -O exit -p 443 localhost 2>>/dev/null
                        TESTVAR=$(echo -n $TESTVAR)
                        echo "Pod status is $TESTVAR"
                        ssh -M -S mongoconnect -fNT -L 27017:$POWERMASTERIP:32767
localhost
                        echo "Tunneled to MongoDB running on IBM"
        fi
fi


TESTVAR=$(ssh root@$AWSMASTERIP -i /home/danielsc/.ssh/id_aws kubectl get pod
appmongo-ibm-mongodb-dev -o custom-columns=:{.status.phase} -n default
2>>/dev/null)
if [ $? -eq 0 ]
   then
        if [ $(cat /proc/"$(ssh -S /home/danielsc/mongo/mongoconnect -O check
localhost 2>&1 |cut -d= -f2 | sed 's/)//')"/cmdline|cut -d: -f2) ==
"a1765d068fc8911e9b07106df2fba1c0-125393943.us-east-2.elb.amazonaws.com" ]
                then
                        TESTVAR=$(echo -n $TESTVAR)
                        echo "Pod status is $TESTVAR"
                        echo "Tunnel already connected to AWS correctly"
                else
                        ssh -S mongoconnect -O exit  localhost 2>>/dev/null
                        TESTVAR=$(echo -n $TESTVAR)
                        echo "Pod status is $TESTVAR"
                        ssh -M -S mongoconnect -fNT -L
27017:a1765d068fc8911e9b07106df2fba1c0-125393943.us-east-2.elb.amazonaws.com:27017
localhost
                        echo "Tunneled to MongoDB running on AWS"
        fi
fi
sleep 30
done
```

Every 30 seconds, the script looks for the pod that is running in the public cloud or on-premises cloud. The script returns the status of the pod if the need exists to tunnel to another location, or the tunnel is correctly mapped to the location where the pod is running. The tunnel is opened to the localhost IP `127.0.0.1`. Then, an alias is added on the local hosts file as `yournetworksolution`, which points to the localhost IP to make it transparent to the client.

Moving the application across clouds

The application movement is performed manually to make it easier to be seen. An automation script can be in place to decide where the database must be located. Also, IBM Cloud Pak for Multicloud Management is an option for controlling the APIs on any OpenShift clusters.

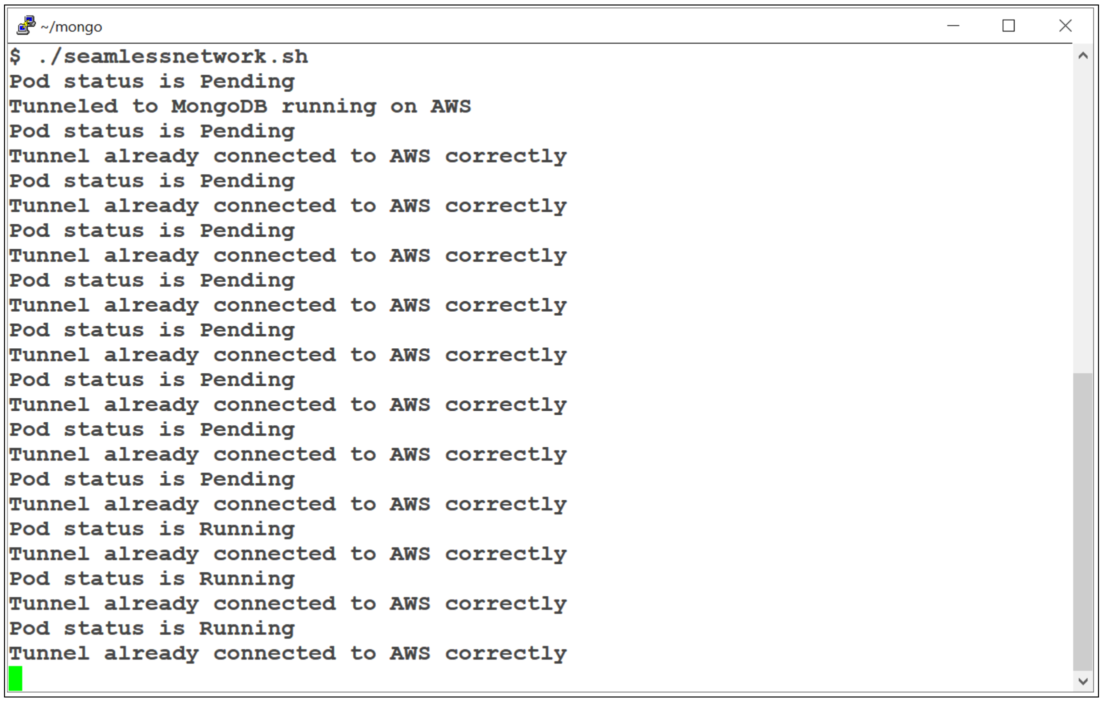
Starting the pod at Amazon Web Services

Run the example script as shown in Example C-1 on page 242. At the point when the pods are not running on any cloud, start them on the Amazon Web Services (AWS) cloud, as shown in Example C-2.

Example C-2 Starting MongoDB Cloud in AWS

```
[root@ip-10-0-2-217 ~]# oc apply -f mongo_pod.yaml
pod/appmongo-ibm-mongodb-dev created
[root@ip-10-0-2-217 ~]# oc get pod appmongo-ibm-mongodb-dev
NAME                           READY   STATUS              RESTARTS   AGE
appmongo-ibm-mongodb-dev       0/1     ContainerCreating   0          3s
```

The script queries both clouds and understands that the pod is running in the public cloud cluster, as shown in Figure C-1.

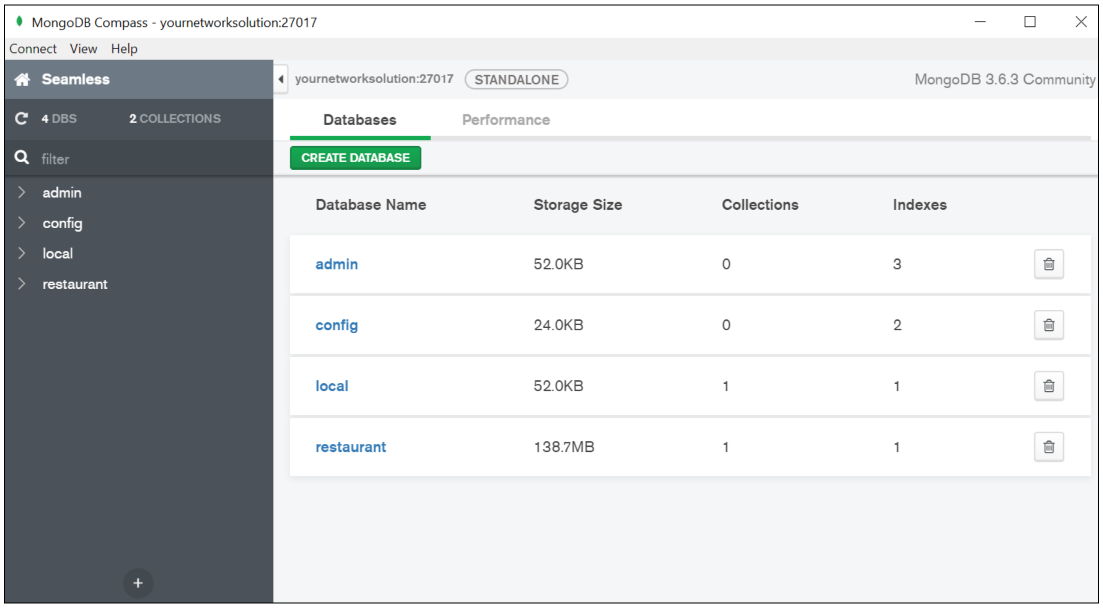


```
~/mongo
$ ./seamlessnetwork.sh
Pod status is Pending
Tunneled to MongoDB running on AWS
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Running
Tunnel already connected to AWS correctly
Pod status is Running
Tunnel already connected to AWS correctly
Pod status is Running
Tunnel already connected to AWS correctly
```

Figure C-1 Tunnel that is connected to the public cloud

Accessing MongoDB by using the tunneled connection

Now that the solution is running, point the MongoDB compass to the hostname that was created. As expected, you can see it is a seamless connection, as shown in Figure C-2.



Database Name	Storage Size	Collections	Indexes
admin	52.0KB	0	3
config	24.0KB	0	2
local	52.0KB	1	1
restaurant	138.7MB	1	1

Figure C-2 Connecting by using the tunneled connection

Moving to the on-premises Power Systems cloud

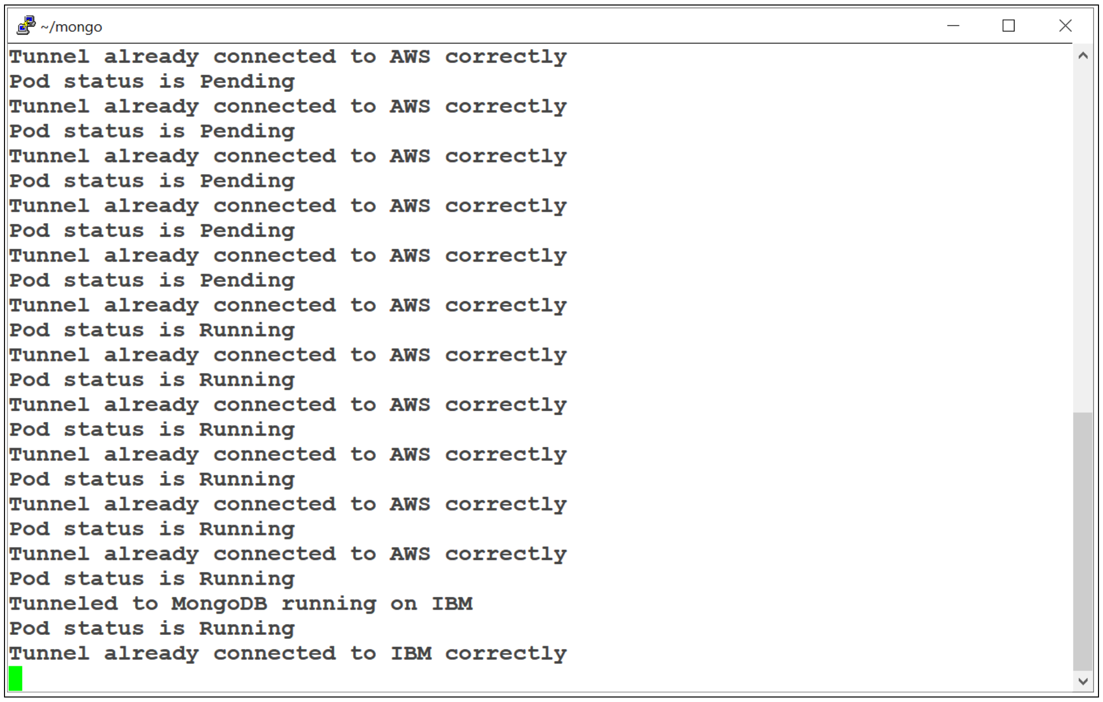
To show the seamless movement, delete the pod in the public cloud and bring it up in the on-premises cloud (Power Systems cluster), as shown in Example C-3.

Example C-3 Deleting the pod in the public cloud and bringing it up in the Power Systems cloud

```
[root@ip-10-0-2-217 ~]# oc delete pod appmongo-ibm-mongodb-dev
pod "appmongo-ibm-mongodb-dev" deleted
[root@ip-10-0-2-217 ~]# oc get pod appmongo-ibm-mongodb-dev
Error from server (NotFound): pods "appmongo-ibm-mongodb-dev" not found
[root@ip-10-0-2-217 ~]#

[root@dcocp01 ~]# oc apply -f mongo_pod.yaml
pod/appmongo-ibm-mongodb-dev created
[root@dcocp01 ~]# oc get pod appmongo-ibm-mongodb-dev
NAME                       READY     STATUS    RESTARTS   AGE
appmongo-ibm-mongodb-dev   1/1       Running   0          9s
[root@dcocp01 ~]#
```

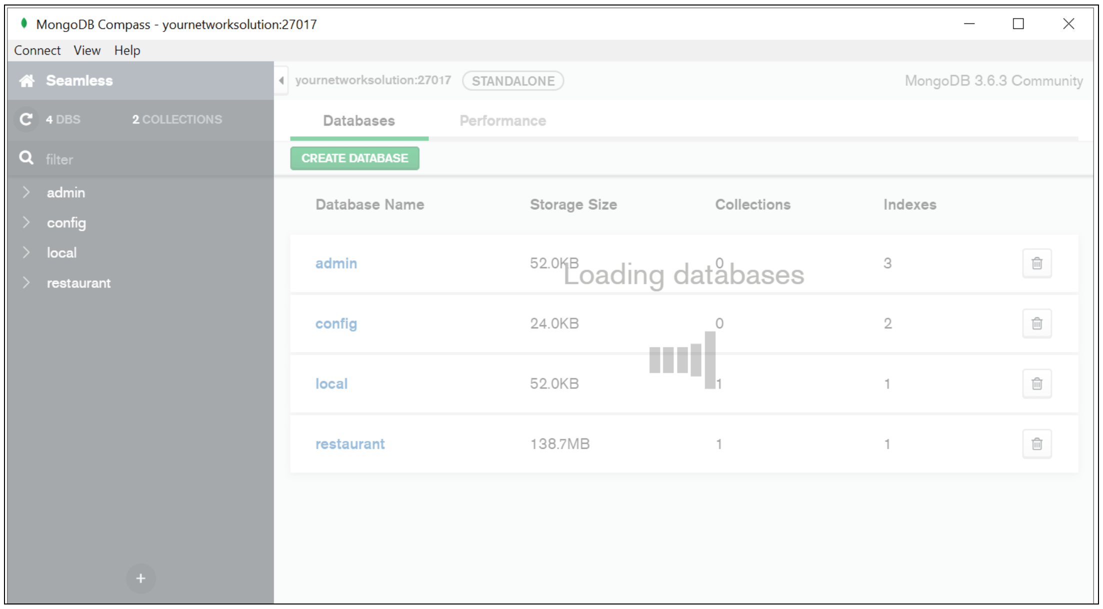
The script checks again, validates that the pod is now running in the on-premises cloud, and points the tunnel to it, as shown in Figure C-3.



```
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Pending
Tunnel already connected to AWS correctly
Pod status is Running
Tunnel already connected to AWS correctly
Pod status is Running
Tunnel already connected to AWS correctly
Pod status is Running
Tunnel already connected to AWS correctly
Pod status is Running
Tunnel already connected to AWS correctly
Pod status is Running
Tunnel already connected to AWS correctly
Pod status is Running
Tunneled to MongoDB running on IBM
Pod status is Running
Tunnel already connected to IBM correctly
```

Figure C-3 Changing the tunnel to the on-premises cloud

After the compass connection is reloaded (as shown in Figure C-4), you can access the MongoDB instance again without disconnecting.



Database Name	Storage Size	Collections	Indexes
admin	52.0KB	0	3
config	24.0KB	0	2
local	52.0KB	1	1
restaurant	138.7MB	1	1

Figure C-4 Reloading the connection on the new MongoDB instance without disconnecting

Glossary

Application Programming Interface (API)
An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

Bare metal machine
A dedicated, fully-customizable physical server that can be used for virtualization or web hosting.

CI/CD
Continuous Delivery/Continuous Integration

Continuous Delivery (CD)
A practice by which you build and deploy your software so that it can be released into production at any time.

Continuous Integration (CI)
A process where developers integrate their code more frequently to identify integration issues earlier when they are easier to fix.

Command Line Interface (CLI)
A computer interface in which the input and output are text-based.

Cloud computing
A computing platform where users can access applications or computing resources as services from anywhere through their connected devices.

Cloud Native Computing Foundation (CNCF)
A consortium to promote cloud-native technologies and principles, and provide guidance to the cloud community. Although CNCF is not intended to be a standards body, they do have criteria and a review board before a project can join.

Cloud native
How an application is built and deployed. A cloud native application consists of discrete, reusable components that are known as microservices that integrate into any cloud environment.

Cloud provider
An organization that provides cloud computing resources.

Container
A system construct that allows users to simultaneously run separate logical operating system instances. Containers use layers of file systems to minimize image sizes and promote reuse.

Containerization
A practice of encapsulating or packaging up software code (containers) and all its dependencies so that it can run uniformly and consistently on any infrastructure.

Container Orchestration
Manages the deployment, placement, and lifecycle of containers. See Kubernetes.

CRI-O
An integration point between Kubernetes and container runtimes that makes pods (groups of containers) work in Kubernetes clusters.

Dashboard
A user interface component that provides a comprehensive summary of pertinent information from various sources to the user.

DevOps
A software methodology that integrates application development and IT operations so that teams can deliver code faster to production and iterate continuously based on market feedback.

Dockerfile
A text file that contains instructions to build a Docker image.

Hybrid cloud
A cloud computing environment that consists of multiple public and private resources.

Infrastructure as a Service (IaaS)
The delivery of a computer infrastructure, including server functionality, networking functionality, data center functionality, and storage functionality, as an outsourced service.

Identity and Access Management (IAM)
The process of controlling access of authorized users to data and applications, at the same time helping companies comply with various regulatory requirements.

Image
A file and its execution parameters that are used within a container runtime to create a container. The file consists of a series of layers, combined at runtime, that are created as the image is built by successive updates.

Instance
An entity that consists of resources that are reserved for a particular application or a service.

Internet of Things (IoT)
A global network of endpoints that can capture or generate data. For example, a smartphone, smart watch and back-end server might all communicate with each other, sending data back and forth, or even to other devices within the network.

Kubernetes (also known as k8s or kube)
A container orchestration platform for scheduling and automating the deployment, management, and scaling of containerized applications.

Load Balancer as a Service (LBaaS)
A service that provides the ability to distribute traffic among instances in a virtual private cloud.

Local cloud
A cloud computing environment within the client's data center. The local cloud is on-premises, which provides improved latency and security.

Mobile Backend as a Service (MBaaS)
A computing model that connects mobile applications to cloud computing services and provides features, such as user management, push notifications, and integration with social networks.

Mobile cloud
An infrastructure in which the storage and processing of data for applications is offloaded from a mobile device into the cloud.

Multicloud
A cloud adoption strategy that embraces a mix of cloud models (public, dedicated, private, and managed) to best meet unique business, application, and workload requirements.

Microservices
An application architectural style in which an application is composed of many discrete, network-connected components that are called microservices.

OCI container image
A container image that is compliant with the OCI Image Format Specification.

On-premises (on-prem)
Software that is installed and run on the local computers of a user or organization.

Platform as a Service (PaaS)
The delivery of a computing platform, including applications, optimized middleware, development tools, and runtime environments, in a cloud-based environment.

Pod
A group of containers that are running on a Kubernetes cluster. A pod is a unit of work that can be run, which can be a a stand-alone application or a set of microservices.

Private cloud
A cloud computing environment on which access is limited to members of an enterprise and partner networks.

Public cloud
A cloud computing environment on which access to standardized resources, such as infrastructure, multi-tenant hardware, and services, is available to subscribers on a pay-per-use basis.

Registry
A public or private repository that contains images that are used to create containers.

Software as a Service (SaaS)
A model of software deployment whereby software, including business processes, enterprise applications, and collaboration tools, are provided as a service to customers through the cloud.

Related publications

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

IBM Redbooks

The IBM Redbooks publication *IBM PowerVM Best Practices*, SG24-8062*,* provides more information about the topic in this document. Note that this publication might be available in softcopy only.

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

ibm.com/redbooks

Online resources

The following websites are also relevant as further information sources:

- Deploying Red Hat OpenShift Container Platform 3.11 on Red Hat OpenStack Platform 13

  https://red.ht/2pEFNpV
- OpenShift on POWER

  https://red.ht/337zOIT
- Kubernetes concepts

  https://kubernetes.io/docs/concepts/services-networking/service/
- IBM PowerVC

  https://www.ibm.com/us-en/marketplace/powervc
- Using PowerVC storage

  https://ibm.co/34Cko06
- Red Hat OpenShift Container Platform 3.11 CLI Reference

  https://red.ht/2XZGBmz

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

Red Hat OpenShift and IBM Cloud Paks on IBM Power Systems: Volume 1

(0.2”spine)
0.17”<->0.473”
90<->249 pages

SG24-8459-00

ISBN 0738458376