Redbooks

ibm.com/redbooks

Implementation Guide for IBM Blockchain Platform for Multicloud

Austin Grice

Eric Everson Mendes Marins

Garrett Lee Woodworth

Juliana Medeiros Destro

Rahul Gupta

Vasfi Gucer

Cloud

LinuxONE

In partnership with
IBM **Academy of Technology**

IBM

Redbooks

IBM Redbooks

Implementation Guide for IBM Blockchain Platform for Multicloud

October 2019

SG24-8458-00

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

First Edition (October 2019)

This edition applies to IBM Blockchain Platform for Multicloud Version 2.0.

Contents

Figures

Examples

Tables

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

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

Db2®	IBM z13®	UrbanCode®
developerWorks®	IBM z13s®	WebSphere®
FICON®	IBM z14®	z/OS®
IBM®	IBM z15™	z/VM®
IBM Cloud™	Passport Advantage®	z13®
IBM Spectrum®	Redbooks®	z13s®
IBM Z®	Redbooks (logo) ®	z15™

The following terms are trademarks of other companies:

ITIL is a Registered Trade Mark of AXELOS Limited.

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

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

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

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

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

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

Preface

IBM® Blockchain Platform for Multicloud enables users to deploy the blockchain across public and private clouds, such as the IBM Cloud™, their own data center, and third-party public clouds, such as AWS and Microsoft Azure. It provides a blockchain console user interface that you can use to deploy and manage blockchain components on an IBM Cloud Private cluster.

This IBM Redbooks® publication discusses the major features, use case scenarios, deployment options, configuration details, performance, and scalability considerations of IBM Blockchain Platform for Multicloud. It also covers step-by-step implementation details for both Secure Service Container and non-Secure Service Container environments. You also learn about the benefits of deploying and using a blockchain environment on LinuxONE.

The target audience for this book is blockchain deployment specialists, developers, and solution architects.

Authors

This book was produced by a team of specialists from around the world who worked together at the at IBM Redbooks Austin Center.

Austin Grice is a Blockchain Technical Leader for IBM out of the Washington Systems Center and Federal market. Since joining IBM in 2016, Austin has focused on blockchain running on the IBM Z® platform. He helps North American clients, IBMers, and everyone in-between with blockchain. He specializes on creating innovative hands-on lab material that focuses on Hyperledger Fabric and the IBM Blockchain Platform. He firmly believes that blockchain is a revolutionary technology that will impact businesses from various industries, not only transforming their daily processes, but also transforming their customers' experience.

Eric Everson Mendes Marins is a Senior IT Architect for IBM Global Technology and Services in Brazil, focused on hybrid cloud solutions, Infrastructure and Platform solutions and competencies, including High Availability, Disaster Recovery, networking, Linux, and Cloud. Eric works with IGA (IBM Global Account), designing and implementing complex hybrid cloud solutions that involve Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) capabilities. He has more than 17 years of experience in designing and implementing Linux on IBM Z solutions. He is IBM L3 IT Specialist certified and an IBM L3 IT Architect certified. Also, he holds a degree in Computer Science from Faculdade Ruy Barbosa and a post-graduate degree in Computer Information Systems (Database Management). His areas of expertise include Linux, IBM z/VM®, Docker, Kubernetes, high availability, IP networking, and server management. Eric has co-authored several IBM Redbooks publications, including *Advanced Networking Concepts Applied Using Linux on IBM System z*, SG24-7995, *Security for Linux on System z,* SG24-7728, *Scale up for Linux on IBM Z,* REDP-5461 and *Getting Started with Docker Enterprise Edition on IBM Z,* SG24-8429.

Garrett Lee Woodworth is a blockchain and Kubernetes Specialist for IBM covering North America for the Washington Systems Center. Since joining IBM in 2017, Garrett has been traveling across North America enabling everyone from developers to executives on combining emerging technologies such as blockchain and Kubernetes with the security and reliability of the IBM LinuxONE and IBM Z family of systems. He leverages his education and degree in Computer Engineering from the University of California, Davis to engage with both the business and technical considerations around these innovative technologies. Working with open source technology, he believes in publicly sharing his work with the world whenever possible through GitHub (siler23) [such as the blockchain workshop available at ibm.biz/bc-immersion] and various other sites such as IBM developerWorks®. Garrett insists that people must constantly evaluate their practices:

- How they do business.
- What they accept as "normal."

In this way, they can develop the most effective solutions by incorporating innovative technologies like blockchain (which can instill trust in transactions) and Kubernetes (which can eliminate the boundary from one machine to another).

Juliana Medeiros Destro is a Developer in IBM Brazil, and joined IBM in 2003. She has a strong background in security and accumulated extensive expertise in infrastructure, operating systems, and several programming languages. She has been a technical speaker in internal and external events, and has delivered several security solutions to key external customers in Finance Industry. Currently, she is working in CIO on business transformation and blockchain deployment. She recently obtained her PhD degree in Computer Science from UNICAMP.

Rahul Gupta is a Cloud Native Solutions Architect in IBM Cloud Solutioning Centre in the US. Rahul is an IBM Certified Cloud Architect with 14 years of professional experience in IBM Cloud technologies, such as Internet of Things, blockchain, and Container Orchestration Platforms. Rahul has been a technical speaker in various conferences worldwide. Rahul has authored several IBM Redbooks publications about messaging, mobile, and cloud computing. Rahul is an IBM Master Inventor and also works on MQTT protocol in the OASIS board for open source specifications.

Vasfi Gucer is an IBM Redbooks Project Leader with the IBM International Technical Support Organization. He has more than 20 years of experience in the areas of systems management, networking hardware, and software. He writes extensively and teaches IBM classes worldwide about IBM products. His focus has been on cloud computing for the last 5 years. Vasfi is also an IBM Certified Senior IT Specialist, Project Management Professional (PMP), IT Infrastructure Library (ITIL) V2 Manager, and ITIL V3 Expert.

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

Rebecca Gott, Shalawn King, Matt Lesher, Ann Lund, Fehmina Merchant, Seongwook Park, Lukas A. Staniszewski, Jin VanStee, Paul Tippett, Henry Welborn
IBM USA

Claudio Boscaino, Joao Eduardo Proenca Pascoa, Diego Jose Basso Pigossi, Leandro Raphael Pinto, Gustavo Henrique Da Silva Sommaggio, Carlos Roberto Visconde
IBM Brazil

Ming Zhe Jiang
IBM China

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

1

Introduction

This chapter introduces IBM Blockchain Platform for IBM Cloud Private on a Secure Service Container and non-Secure Service Container technologies. It also covers blockchain basics, benefits, and differentiators for deploying and using a blockchain environment on LinuxONE.

This chapter has the following sections:

1.1 Introduction

Blockchain is a technology for a new generation of transactional applications that fundamentally changes the way businesses create and capture value.

This section provides an overview of blockchain technology and how developers can build solutions and use IBM Blockchain Platform to launch, test, and move applications into production.

We describe the benefits of running blockchain applications on a powerful platform, LinuxONE.

1.1.1 What does blockchain do for a business network?

Blockchain is a shared, distributed ledger that facilitates the recording of transactions and tracking of assets in a *business network*. Virtually anything of value can be tracked and traded on a blockchain network, thereby reducing risks and costs for everyone involved.

A business network describes any group of organizations or individuals that connect with a desire to transfer or share assets. Those assets can be tangible, such as food or manufactured goods, or digital, such as music or data. Items are tracked in a common, shared ledger that is distributed across the business network. As a result, assets can be transferred between members. Each member has a record of the transaction and access to the latest version of the ledger.

Blockchains establish trust across a business network through the combination of a distributed ledger, smart contracts, and consensus. The ledger contains the current state of assets and the history of all transactions. Transactions can only be added to the ledger, not removed. Past transactions are protected with cryptography so that they cannot be successfully tampered with. This way, changes to the ledger are final and immutable, allowing the ledger to be the source of truth within the network.

Table 1-1describes the set of benefits that blockchain brings to the use cases for business networks. In essence, blockchain adds several types of trust to asset transfers.

Table 1-1 Benefits of blockchain

Benefit	Meaning
Immutability	The historical record of transactions cannot be altered.
Provenance	The origin of any assets that are contained in the ledger is known.
Consensus	Changes to the ledger require approval by participants according to an agreed upon endorsement policy. This goal is achieved through consensus, in which participants of the network endorse that the transaction is valid and come to agreement on the updated state of the ledger.
Finality	Each participant is assured that their copy of the ledger matches all other copies, and that transactions that are contained in the blockchain are committed faithfully.

A smart contract is an application that contains business logic. Smart contracts define all access to the ledger and are invoked by participants to query or update values of assets.

1.1.2 Why blockchain?

Ledgers are at the core of every business. Ledgers store the flow of assets, such as payments to suppliers, taxes owed, and goods delivered. They also store the firm's current balances. A firm's revenue, relationships, and assets are all recorded in a core system of ledgers. For millennia, such records were stone, clay, or paper-based. However, these records were among the first systems to be automated and moved to computing systems.

Even with the digitization of ledgers, many transaction-related tasks remain manual and outdated. Each shipping container that is moved by freight requires a stack of paperwork that adds significant processing time. Each time that you open a new bank account or visit a new doctor, you must share personal information using redundant and insecure forms. The transfer of financial assets between parties often takes days to settle.

If ledgers, financial records, personal records, and inventory are now digitized, why do these inefficiencies remain? The reason is that the move to digitization has generally only occurred within an organization, rather than between organizations. In addition, the systems that were created often cannot communicate with each other. The result is that modern business-to-business processes perpetuate inefficiencies that date from when ledgers were on paper. Blockchain is here to change how business-to-business processes work.

1.1.3 IBM Blockchain Platform introduction

IBM Blockchain Platform (IBP) is an IBM Cloud offering that is built on Fabric, a blockchain infrastructure that is provided by the open source Hyperledger project. IBP simplifies development and management of a blockchain network. Often, you can accomplish the following tasks with just a few clicks in the easy-to-use interface:

- Automated deployment of Fabric
- Creation of custom governance policies
- Initial development
- Deployment of the application into production, including creation of channels and deployment of chaincode
- Inviting new members into the network, and managing identity credentials over time.

Additional IBM Blockchain Platform references:

- *Zero to Blockchain IBM Redbooks workshop*

https://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/crse0401.html?Open

- *Developing a Blockchain Business Network with Hyperledger Composer using the IBM Blockchain Platform Starter Plan,* REDP-5492

http://www.redbooks.ibm.com/abstracts/redp5492.html?Open

1.1.4 Benefits and differentiators of deploying and using a blockchain environment on LinuxONE

The LinuxONE platform combines the essential features for operation, deployment, and governance of blockchain workloads:

- Unique functionality and an ability to handle the most sensitive data and critical applications.
- Optimization for dealing with massive transactions and memory operations.
- Ability to scale for new work.
- Option to base security on the new cryptocard.

Indeed, blockchain can use special hardware in the LinuxONE to deploy at scale, with performance, availability, and security built in. Additionally, this platform can deliver the following benefits:

- Blockchain peer-to-peer nodes realize optimized communication with IBM z/OS®, speeding up access to colocated business data
- Isolated partitions in memory keep ledgers separate and secure
- Availability and scalability of the LinuxONE servers as an environment for both blockchain development/test and production
- Vertical (scale up) scalability offers unmatched processing power
- Reduced data center footprint, simplified management, and energy savings
- Hardware encryption with built-in accelerators for blockchain hashing, signing, and security
- Faster responses with IBM HiperSockets
- Global Security Standards compliant
- Tamper-proof crypto keys in firmware/crypto cards
- Unlimited random keys to encode transactions
- More efficient: Can scale up or scale out
- Fewer points of failure: Greater availability

Indeed, LinuxONE unique capabilities can vastly improve efficiency and performance of blockchain applications, without requiring hundreds of distributed servers to handle the application workload. LinuxONE users in large-scale operations naturally save on energy, cooling, and space when they consolidate x86 workloads onto a single LinuxONE machine.

Furthermore, users in the financial services industry require redundant processor execution steps and integrity checking. In fact, LinuxONE platform typically enables hot-swapping of hardware, such as processors and memory. This swapping is typically transparent to the operating system, enabling routine repairs to be performed without shutting down the system.

Note: The LinuxONE platform can deliver all the strength of the architecture in a single server or multiple servers, depending on the business requirements.

Cloud computing platforms and mobile applications grow more common in organizations around the world. Their success depends on the ability of the infrastructure that supports them to respond faster to demands for intelligent user experiences, high availability, and highly securable environments.

LinuxONE platform runs open source software with flexibility, security, and scalability. Your system thus gets the cost and availability benefits of new software applications like the cloud-based MongoDB, blockchain, and Docker. Imagine combining the best of both scaling worlds with the effective costs and availability advantages.

This platform architecture is flexible enough to allow applications that run on cloud services to easily scale up in any increments that you require. At the same time, processing power is available for high-performance applications. Also, it addresses application scale-out by adding new virtual servers within minutes and making them part of the cluster.

Scale out or scale up alone do not adequately serve today's applications. These applications must be responsive, always available, and handling many user types, including cloud, mobile, analytics, and big data.

These factors make LinuxONE the best platform to meet different demands, from small to complex. In summary, the platform remains the powerhouse for tasks that computers have performed for decades and technologically redesigned to attend emerging demands.

Performance

This section summarizes blockchain-related performance metrics for LinuxONE servers.

Scale out performance

The following scale-out information refers to performance tests that IBM performed on a LinuxONE server.

- Run 1344 concurrent databases that execute a total of 377 billion database transactions per day on a single LinuxONE Emperor II server.
- Run 25% more MongoDB guests with the same throughput under z/VM 6.4 on the LinuxONE Emperor II server compared to the LinuxONE Emperor server.
- Use up to 170 cores on the LinuxONE Emperor II server to scale out MongoDB databases under z/VM 6.4, each with a constant throughput and not more than 10 µs latency increase per additional MongoDB instance.
- Scale out to 2 million Docker containers in a single LinuxONE Emperor II system, no application server farms necessary.
- Run 41.8 billion web transactions per day on a single Emperor II server.

Scale up performance

The following scale-up information refers to performance tests that IBM performed on a LinuxONE server.

- Scale up a single MongoDB instance to 17 TB in a single system without database sharding and get 2.4 times more throughput and 2.3 times lower latency on a LinuxONE Emperor II server that leverages the additional memory that is available, compared to a LinuxONE Emperor server.
- Run MongoDB under z/VM 6.4 on a LinuxONE Emperor II server and get 4.8 times better performance leveraging additional memory available per z/VM instance compared to a LinuxONE Emperor server.

Blockchain performance

Benchmark tests put four 64-byte values in a set of blockchain transactions, with these results on a LinuxONE Emperor II server:

- 2.3 times more throughput per core than an x86 Skylake server.
- 82% lower latency than an x86 Skylake server. Figure 1-1 shows the lab environment that was used for the blockchain performance tests on LinuxONE.

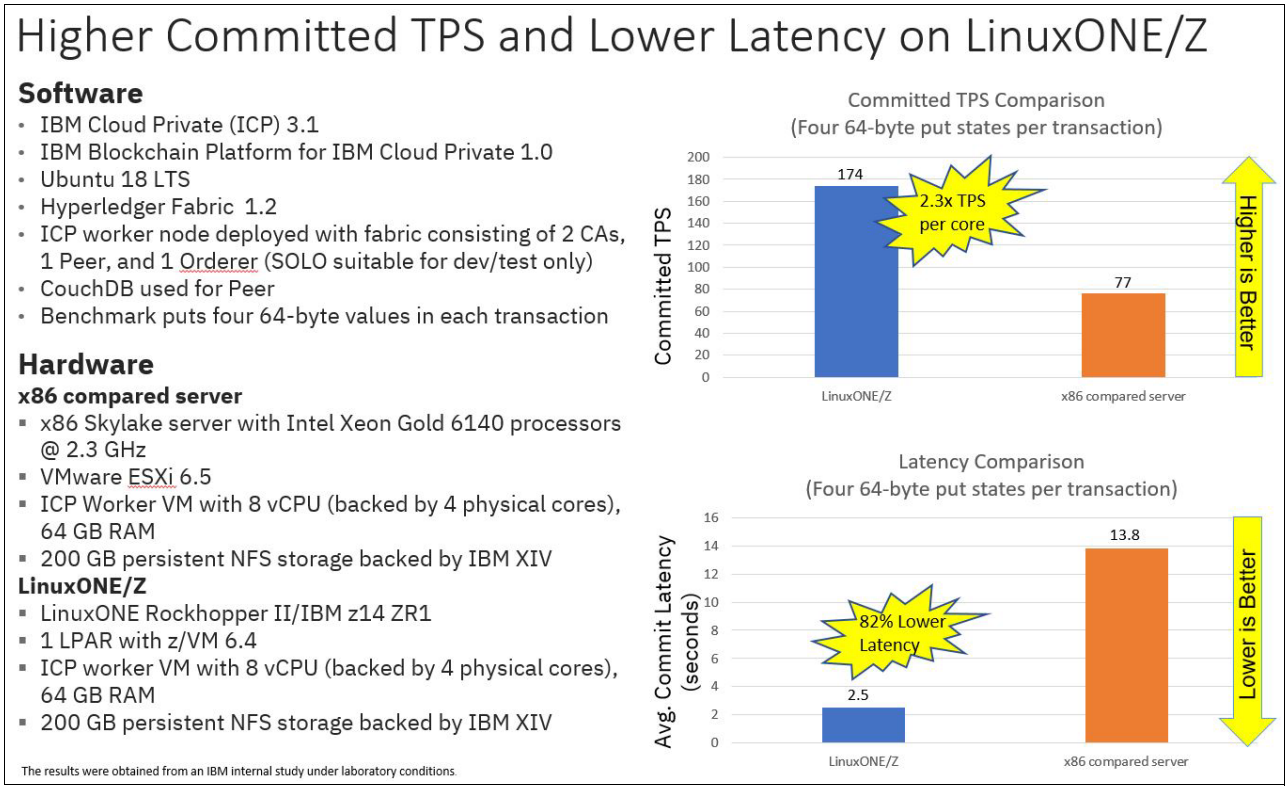


Figure 1-1 Lab environment that was used for blockchain performance tests on LinuxONE

The latest LinuxONE systems have a tried-and-true architecture to support digital transformation, create a strong cloud infrastructure, and make back-end services available through secure APIs. They can also streamline your ability to integrate disparate data center systems and create a single, cohesive IT shop.

Deployment of blockchain workloads on LinuxONE platform enhances security and allows quick scaling of the environment. A blockchain network must be protected and safe against insider attacks, data leaks, and hardware tampering through the use of encryption. So, if your organization is thinking about exploring blockchain technology, consider how the LinuxONE platform securely supports blockchain initiatives.

Finally, this platform offers a differential infrastructure and open standards for running blockchain platform to take advantage of the latest technology advancements and benefits of LinuxONE.

1.2 Typical use cases

Blockchain has limitless applicability across many, if not all, industries. The following three rules unlock the power of blockchain.

1. ***Have a business problem to solve.*** One question to ask yourself about business problems is, *What manual processes are increasing the time that it takes to react?*
2. ***Identify a network that solves the business problem.*** Blockchain must exist within a network. This network includes 3 or more organizations, or the network can consist of departments within a single organization.
3. ***Have an element of trust within your network.*** Blockchain inspires mutual trust between organizations, especially if there is a lack of trust in the existing network.

The following chart suggests use cases for specific industries:

Industry	Use case
Supply Chain	Current supply chains do what they are supposed to do, but there are vast areas in which the process can be improved upon. For example, many manual processes must be fulfilled to transfer a product from one organization to another. Also, this manual process lacks transparency across the broader network and the network organizations. Blockchain allows for a product to be tracked, in real time, from the origin all the way to final step in the supply chain. Due to the process being digital, there is now adequate transparency regarding where the product is in the journey. Additionally, regulatory compliance accelerates because customs and border agencies quickly identify what is coming into their respective countries.
Healthcare	Currently, healthcare data is siloed across various medical practices and large organizations, largely due to regulations and compliance requirements. For example, a patient fills out a form that summarizes their medical history at one medical office. Later, the patient fills out a form withthe same information at a different medical office. When blockchain can be deployed across the healthcare industry, the patient fills out the history form once. Then, that information can be shared with the various medical practices and organizations with the consent of the patient. This means that the patient has more control of their data and can provide a single source of truth regarding their medical history.
Digital Identity	A driver's license is a form of digital identity that permits an individual to operate a motorized vehicle on public roads. The license is granted to persons as proof of their ability to drive. Beyond that purpose, the license might serve as personal ID for security checkpoints in domestic air travel. In contrast, a passport is required for international air travel. Consider the changes that blockchain could bring to an individual's management of their personal identity. For example, the full digitization that blockchain provides might enable new degrees of privacy. You might be able to give only a small amount of personal information to successfully pass a security checkpoint. Researchers are investigating *zero knowledge proof* (ZKP), through which your identity is verified without revealing any of your personal data.
Finance	Sending money from one country to another takes 3 to 7 business days as the transaction passes through multiple intermediaries such as clearing houses. Such delays can be detrimental to citizens who depend on funds from relatives or friends who are located in a different country. Financial institutions transfer billions of dollars daily and experience similar delays. Blockchain permits money transfers in times frames that can be much closer to real time. All parties can enjoy more flexibility in money management. Furthermore, unbanked individuals might have more options to participate in the global economy thanks to the simpilfications and efficiencies that blockchain brings to digital identity and financial transactions.

Industry	Use case
Music	Today, music streaming is the typical medium for artists to sell their work. The fees that are associated with streaming services restrict the profit potential for these artists. With blockchairn, musicians could sell directly to their fans and enjoy greater revenue for the music that they write or produce. Doubtless, other benefits can emerge as musicians increase their direct business and artistic relationship with their fans.

Additionally, Figure 1-2 shows more use cases based on various industries:

Figure 1-2 More use cases based on various industries

As the preceding use cases show, blockchain can affect multiple industries in multiple ways. Also, each example illustrates the rules for blockchain:

- Have a business problem to solve.
- Identify a network that solves the business problem.
- Have an element of trust within your network.

To see more sample use cases and videos that showcase actual use cases in production, go to http://www.ibm.com/blockchain/use-cases.

1.3 Solution components

This section presents the components that are required for the IBM Blockchain Platform on LinuxONE, including an overview and key features of each component.

1.3.1 LinuxONE

LinuxONE is the premier Linux server hardware for highly secure data and cloud services. The option to have dedicated cryptographic processors means encryption for data at rest and for data in transit. These cryptographic processors supplement the standard processing units that process the applications. Partitions with Secure Service Container (SSC) technology help to protect data and applications from internal and external threats.

Highly engineered for high performance, large-scale data, and cloud services, a single LinuxONE platform consolidates hundreds of x86 cores. The platform's dedicated I/O processors allow you to move massive amounts of data while maintaining data integrity.

The first two LinuxONE products were named Emperor and Rockhopper. The second generation of LinuxONE, Emperor II and Rockhopper II, launched in 2017 and early 2018, features the following newest machines:

- **IBM LinuxONE Emperor II**

  This machine features up to 170 processor cores, running at 5.2 GHz, up to 32 TB of RAM, and 640 dedicated I/O processors, all housed in a dual-frame. It supports tens of thousands of sessions and millions of containers. It can run 8,000 virtual servers and over 30 billion RESTful web interactions per day. This server is dedicated to meeting the needs of Enterprise environments.

- **IBM LinuxONE Rockhopper II**

  The same technology as Emperor II, but at a lower price. It is housed in an industry-standard, 19-inch rack. Rockhopper II is available with up to 8 TB of memory and 30 processor cores, running at 4.5 GHz. It supports hundreds of production and development virtual machines (VMs) in a single footprint.

For more information, see this website: https://www.ibm.com/it-infrastructure/linuxone.

1.3.2 Kubernetes (K8s)

Kubernetes is an open source system for the automation of deployment, scaling, and management of containerized applications. It eliminates many of the manual processes involved in these tasks.

The Kubernetes framework runs distributed systems resiliently. It takes care of your scaling requirements, failover, deployment patterns, and more.

The key aspects of Kubernetes are as follows:

- Service discovery and load balancing

  Expose a container that uses the DNS name or that uses its own IP address. If traffic to a container is high, Kubernetes is able to load-balance and distribute the network traffic so that the deployment is stable.

- Storage orchestration

  Allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.

- Automated rollouts and rollbacks

  Just describe the desired state for a deployed container. The actual state changes to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers, and adopt all their resources to the new container.

- Automatic bin packing

  Allows to specify how much CPU and memory (RAM) that each container requires.

- Self-healing

  Kubernetes restarts containers that fail, replaces containers, kills containers that don’t respond to your user-defined health check, and doesn’t advertise them to be used until they are ready to serve.

- Secret and configuration management

  Safely store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys. You can deploy and update secrets and application configuration without rebuilding container images, and without exposing secrets in stack configuration.

Main Kubernetes components are as follows:

- **Master**: Controls Kubernetes nodes. All task assignments originate in a master.
- **Node**: Perform the requested, assigned tasks. Controlled by masters.
- **Pod**: Group of one or more containers that are deployed to a single node. All containers in a pod share an IP address, IPC, hostname, and other resources. Pods abstract the network and storage away from the underlying container.
- **Replication controller**: Controls how many identical copies of a pod can be running somewhere on the cluster.
- **Service**: Decouples work definitions from the pods. Proxies for the Kubernetes service automatically get service requests to the right pod.
- **Kubelet**: A service that runs on nodes, reads the container manifests, and ensures that the defined containers are started and running.
- **kubectl**: The command line configuration tool for Kubernetes.

1.3.3 IBM Cloud Private

IBM Cloud Private (ICP) is a private cloud platform for enterprises to develop and run their workloads locally. It consistes of Platform-as-a-Service (PaaS) and developer services that are needed to create, run, and manage cloud applications.

A platform that embraces open source, ICP delivers modernized IBM middleware and data services to enterprise customers.

As the only private cloud offering that can support LinuxONE, ICP offers the highest levels of security that are available through LinuxONE.

1.3.4 GlusterFS

A scalable network file system that is suitable for data-intensive tasks such as cloud storage and media streaming. GlusterFS aggregates various storage servers into one large parallel network file system.

GlusterFS exports an existing directory as-is, leaving it up to client-side translators to structure the store. The clients themselves are stateless, do not communicate with each other, and are expected to have translator configurations that are consistent with each other.

GlusterFS relies on an elastic hashing algorithm, rather than using either a centralized or distributed metadata model. The user can add, delete, or migrate volumes dynamically, which helps to avoid configuration coherency problems. This approach allows GlusterFS to scale up to several petabytes on commodity hardware by avoiding bottlenecks that normally affect distributed file systems that are more tightly coupled.

1.3.5 IBM Secure Service Container

IBM Secure Service Container (SSC) provides the base infrastructure on LinuxONE for container-based applications, either for hybrid or private cloud environments. This secure

computing environment delivers tamper-resistant installation and runtime operations. Consider these key management features of SSC:

- You can deploy the security capabilities of this environment for microservices-based applications without application code changes.
- SSC restricts administrator access to help prevent the misuse of privileged user credentials for cloud and on premises environments.
- Automatic application of pervasive encryption of data that is in transit and at rest to ensure protection to all application and database data without application changes.
- Additional security capabilities for Docker and other container environments.

Also, Linux provides a comprehensive set of security technologies, including firewalls, VPNs, auditing tools to support regulatory compliance, and SELinux, which is a kernel-based security system.

SSC technology builds on the workload isolation of the firmware that is based LPARs and is unique to IBM LinuxONE.

SSC for IBM Cloud Private consists of the following components:

- A software appliance that is based on the IBM Secure Service Container framework, which can host containerized workloads with a focus on superior data security in the cloud and on-premises.
- An isolated VM image that is used to host IBM Cloud Private proxy and worker nodes, and delivers VM-level isolation to containerized applications.
- A command line tool that is used to automate the base infrastructure of the IBM Cloud Private worker and proxy nodes by using the isolated VM image.

The following requirements apply for all supported appliances:

- Only one appliance can be installed and run in a Secure Service Container partition at any time. This type of partition does not support the simultaneous running of multiple appliances.
- You can define more than one Secure Service Container partition on the same system, and run instances of the same appliance in each one. In this case, each partition must use separate storage devices.
- You can reuse an existing Secure Service Container partition for a different appliance. After you stop the installed appliance and the partition, reboot the Secure Service Container installer and select a different appliance to install. Before you do so, we recommend that you check the storage and network connections for the partition to make sure that they are appropriate for the appliance that you are installing.

You can configure Secure Service Container partitions on the following IBM Z and LinuxONE systems by using HMC:

- IBM z14® TM (z14) (machine type 3906 or 3907)
- IBM z13® (z13) or IBM z13s® (z13s)
- IBM LinuxONE Emperor II TM (Emperor II), or IBM LinuxONE Rockhopper II (Rockhopper II)
- IBM LinuxONE Emperor TM (Emperor), or IBM LinuxONE Rockhopper TM (Rockhopper) machine type 2965

This list shows the SSC component versions for IBM Cloud Private:

- Software appliance 3.4.3
- Isolated VM 1.1.0.3
- Command line tool 1.1.0.3

1.3.6 IBM Blockchain Platform

With blockchain, you can build a decentralized business network. The network is built on business concepts such as assets, contracts, transactions, and so on. The IBM Blockchain Platform (IBP) is the IBM solution for blockchain in business. IBP is designed for hybrid and multi-cloud deployment capabilities, which are enabled by Kubernetes. IBP helps you to build, operate, and grow blockchain networks in heterogeneous environments.

The multicloud deployment is possible through IBM Cloud Private, IBM's Kubernetes-based container orchestration platform. This release provides a console user interface for blockchain, which you can use to deploy and manage blockchain components on an IBM Cloud Private cluster.

IBP is based around Hyperledger Fabric and provides an integrated developer experience with smart contracts that can be easily coded in Node.js, Golang, or Java. You can use the new IBM blockchain VS Code extension to write client applications, based on the IBP console's integration of the Fabric SDK. IBP offers the possibility of deploying only the necessary components to connect to multiple channels and networks, while you maintain control of identities in your environment. Flexible and scalable, IBP can be run in any environment that IBM Cloud Private supports, including LinuxONE.

IBM Blockchain Platform for IBM Cloud Private is delivered as a Helm chart that can be downloaded from IBM Passport Advantage® (PPA). For information on downloading the software, see the "*Installing the IBM Blockchain Platform Helm chart*" section in IBM Knowledge Center here:
https://cloud.ibm.com/docs/services/blockchain/howto?topic=blockchain-console-helm-install#helm-console-install.

1.4 Our lab environment

LinuxONE has been strong for decades in the areas of High Availability and Disaster Recovery. By design, storage and operating systems are implemented in a way to support enhanced availability requirements. The platform architecture allows you to define any type of blockchain infrastructure to accommodate any blockchain demand.

Sizing large-scale blockchain environments is complex. So, we recommend that customers seek assistance from IBM Blockchain specialists to help plan and size blockchain environments that have several characteristics and dependencies.

To show the easy integration into blockchain and LinuxONE platform, we built our lab environment so that IBM Cloud Private (ICP) can manage worker and proxy nodes that run on the SSC partition.

Our environment simulates a typical blockchain setup (Figure 1-3). We built two Z LPARs, each with its own resources. One LPAR is configured to load the SSC, and the second LPAR runs a z/VM that hosts the Redhat Linux server, which acts as the master node.

Having z/VM managing and controlling HW resources makes it easier to scale up or scale down resources. It brings flexibility, security, and manageability to Linux on Z servers.

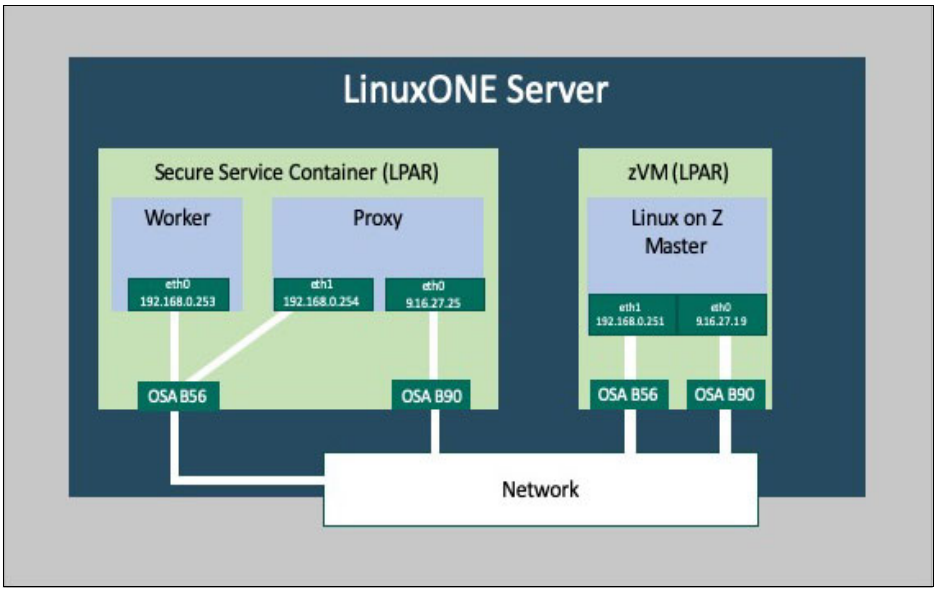


Figure 1-3 Our environment

Our physical lab environment is composed of the following components as shown in Figure 1-3:

- 1 LinuxONE server
  - 1 SSC partition
  - 1 z/VM partition (We built 1 Redhat Linux server on top of z/VM to be the master node)
- 2 network subnets
  - 192.168.0.0/24 (Internal network communication)
  - 9.16.27.0/24 (External network communication)

Before you start with the Secure Service Container for IBM Cloud Private, you can use a worksheet in Table 1-2. This exercise gives you an overall understanding on what information you need to run the Secure Service Container for IBM Cloud Private and where to get such information.

The example values in the worksheet reflect our environment topology, which was built to deploy the Secure Service Container for IBM Cloud Private. You can use different values in the checklist according to your actual network configuration or topology.

1.4.1 Secure Service Container partition

Table 1-2 shows the resources that we used to configure our Secure Service Container storage environment.

Table 1-2 Secure Service Container worksheet

Resource	Example
Partition IP address	9.16.17.18
Master ID	admin
Master password	sscpassword

Resource	Example
Storage disks for data pool resizing	IBM FICON® DASD (all model 54): 0.0.313F 0.0.3140 0.0.3141 0.0.3142 0.0.3143 0.0.3144 0.0.3145 0.0.3146 0.0.3147 0.0.3148 0.0.3149 0.0.3150 0.0.3151 0.0.3152
Notes: 1. You can also use SCSI SAN disks as storage devices for SSC appliances. 2. We used **0.0.313F** to install the SSC image.	

1.4.2 IBM Cloud Private cluster

An IBM Cloud Private cluster must have at least three types of cluster node: master, worker, and proxy. The master node can be hosted on the x86 or Linux on Z server, and the worker and proxy nodes are on the Secure Service Container partitions. In our lab environment, we preferred to install the master node on a Linux on Z server.

The cluster nodes communicate with each other by using internal IP addresses. Table 1-3 shows information about the ICP nodes in our environment.

Table 1-3 ICP Cluster nodes

Function	LPAR	IP addresses	OS version
Master Node	zVM partition	192.168.0.251 (Internal) 9.16.27.19 (External)	Redhat Linux 7.7
Worker Node	SSC partition	192.168.0.253 (Internal)	Ubuntu 18.04 Container
Proxy Node	SSC partition	192.168.0.253 (Internal) 9.16.27.25 (External)	Ubuntu 18.04 Container
Storage Node	SSC partition	192.168.0.245 (Internal)	Ubuntu 18.04 Container
Storage Node	SSC partition	192.168.0.246 (Internal)	Ubuntu 18.04 Container
Storage Node	SSC partition	192.168.0.247 (Internal)	Ubuntu 18.04 Container

Additional information

The following list shows additional information about our environment:

- All ICP functions (boot, master, management, proxy, and worker) running on LinuxONE.
- All management functions performed on the Linux on Z server.
- The Linux on Z master node is running under z/VM, which provides excellent resource management.

- Two 10 GB OSA cards.
  - `0B90` (chpid 00) - This interface is for external cluster communication. (9.16.27.0/24 network)
  - `0B50` (chpid E2) - This interface is for system control functions and internal cluster communication (192.168.0/24 network).
  - Both OSA devices above are shared between SSC and z/VM partition. Therefore, we do not need to assign four devices; only two.
  - Worker and Proxy nodes have direct connection to master node through the internal network by using the internal interface.

> **Important:** In our setup, the OSA cards are shared by both SSC and ZVM partitions (LPARs). For internal cluster communication, we recommend the use of an OSA card because hipersocket interfaces are currently not supported by SSC.

The following table shows information for the ICP nodes. This information is used in the *ssc4icp-config.yaml* file to define the SSC containers.

Table 1-4 ICP resources

Name	ICP function	Internal IP address	External IP address	Mem (GB)	Storage
ssc4icp-master	Master Node	**192.168.0.251**	**9.16.27.19**	32	root_storage: 40G icp_storage: 90G
worker1-15001	Worker Node	**192.168.0.253**	**9.16.27.25**[a]	4	root_storage: 40G icp_storage: 90G
proxy1-16001	Proxy Node	**192.168.0.254**		4	root_storage: 40G icp_storage: 90G glusterfs: 40G
storage-17001	Storage Node 1	**192.168.0.245**		4	root_storage: 40G icp_storage: 90G glusterfs: 40G
storage-18001	Storage Node 2	**192.168.0.246**		4	root_storage: 40G icp_storage: 90G glusterfs: 80G
storage-18002	Storage Node 3	**192.168.0.247**		4	root_storage: 40G icp_storage: 90G glusterfs: 80G
mycluster.icp	Cluster address	**192.168.0.251**[b]			\

a. Only Proxy node requires external IP address
b. Same IP address as the master node

We assume that the reader is familiar with networking and network configuration for Linux on Z. In this document, we do not discuss basic network concepts or how to set up network devices. These topics are discussed in the IBM documentation at the following website:
https://www.ibm.com/support/knowledgecenter/en/linuxonibm/com.ibm.linux.z.lgdd/lgdd_r_pt_dd_net.html

2

Planning for installation

This chapter describes how to plan for the implementation of IBM Blockchain Platform for IBM Cloud Private on a Secure Service Container (SSC) and a non-SSC environment. It also presents design considerations for your blockchain environment when ICP is used for provisioning and managing the cloud resources. In addition, the chapter provides typical blockchain use cases for your reference.

This chapter includes the following sections:

2.1 Why Secure Service Container?

Data breaches are being reported at an increased frequency by large and small organizations, which leads to lack of trust for the organization. Auditors and government regulators increasingly require that customer data is secured, especially sensitive customer data that might be used for fraud or identify theft. As result, security is a major requirement for applications today.

More and more organizations use containerization of workloads to simplify deploy, test, and migration of application workloads from one platform to another, without requiring code updates. Such organizations are required to provide a higher level of security for mission-critical and highly sensitive data.

When you build your container environment into LinuxONE servers, you can take advantage of the benefits from this hardware platform. Benefits of the platform include specialized processors, cryptographic cards, availability, flexibility, and scalability. These systems are not only prepared to handle blockchain applications, but also applications for cloud, mobile, big data, and analytics contexts.

LinuxONE servers deliver the highest standards of security and reliability, with millisecond response times, hybrid cloud deployment, and performance at scale. The world's top organizations rely on this platform to protect their data with impeccable security to run blockchain applications.

Although you can deploy your blockchain network in any server, IBM Secure Service Containers (SSC) brings more security to blockchain applications. SSC is a special logical partition that does not allow access through outside interfaces, such as SSH. The only access to an application that is running in SSC mode is through remote APIs in the application.

SSC is an appliance that provides the base infrastructure for an integration of operating system, middleware, and software components. It works anonymously and provides core services and infrastructure with a focus on security. The blockchain application that is deployed under this appliance is certified, encrypted, and very secure because it does not allow you to install malware. In fact, the SSC blockchain appliance adds a layer of security to your blockchain environment and protects your data against both internal and external threats.

This technology also delivers the best firmware protection to ensure the integrity of a blockchain environment. Even privileged system administrators cannot access the blockchain data, software, and applications. The inherent features for data security, including SSC, clearly differentiate LinuxONE servers from other servers.

SSC contains its own embedded operating system, security mechanisms, and other features to prevent unauthorized access. The result is a highly secure infrastructure to run blockchain workloads with these benefits:

- Simplified, fast deployment and management of packaged solutions
- Tamper protection during appliance installation and runtime
- Confidentiality of data and code that runs in an appliance for data in transit and data at rest
- Management through Remote APIs (RESTful) and web interfaces

Figure 2-1 provides a high-level view of the Secure Service Container components.

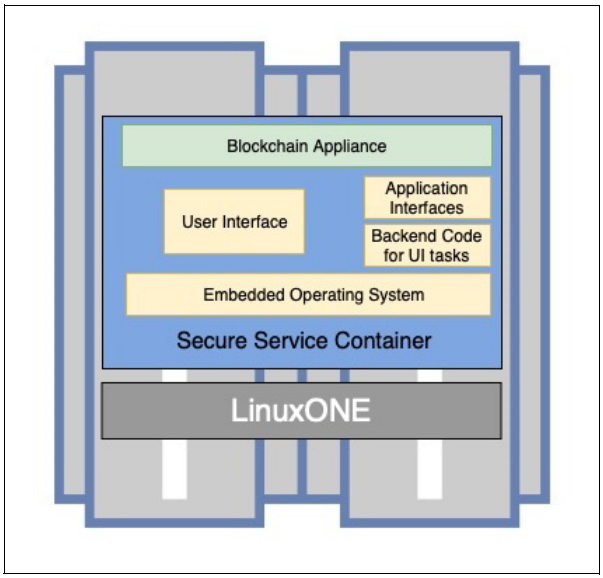


Figure 2-1 Security Service Container components

The premise for running IBM Blockchain Platform on SSC is security. Here is a summary of the main advantages of this appliance:

1. Protection from misuse of privileged user credentials.
2. Leveraging of security features without code changes.
3. SSC boots only untampered appliances, which protects your application from malware or malicious code.
4. Data and code are encrypted in transit and at rest.
5. System administrators cannot access the memory or processor state.
6. There is no direct interaction at the host or OS levels. Only well-defined interfaces give access into and out of the appliance.
7. Peer isolation that is based on LinuxONE's EAL5+ certified LPAR isolation for near 'air-gap' separation of appliance environments, on a single footprint. This feature blocks malware from getting on, or even detecting the appliance.
8. Debug Data (memory dumps) are encrypted.

SSC is designed for isolation. Isolation is helpful for security and it helps to protect workloads from adverse effects of other workloads. For example, another workload might fail or compete for resources during heavy-load periods. LinuxONE partitions provide EAL5+ isolation and ensure that the blockchain applications you deploy to different partitions do not interfere with each other. Additionally, SSC protects your environment from being injected by a malware or a malicious code. The Figure 2-2 illustrates the level of protection your application can have if it runs with an SSC container on LinuxONE server.

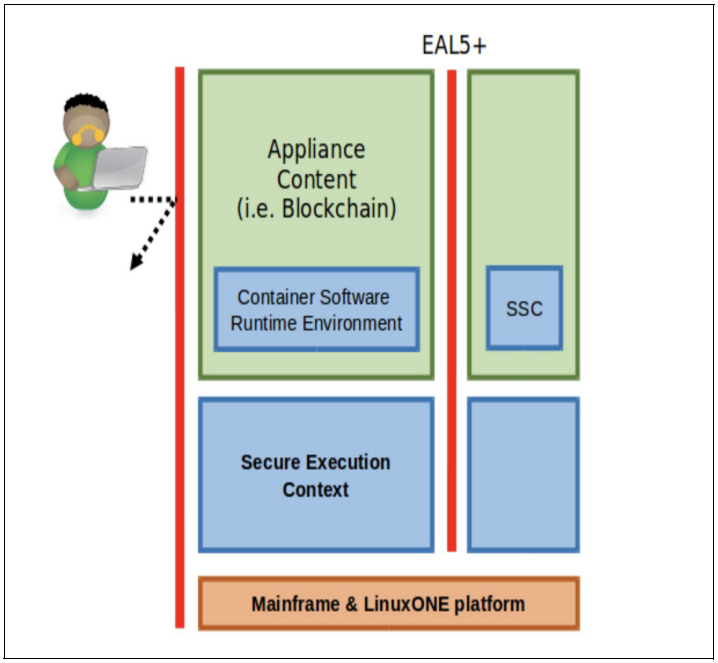


Figure 2-2 SSC security protection

Depending on your workload security needs, SSC containers can be the best choice.

In the past, the manual way of installing applications and its dependencies individually might satisfy the system administrators and application developers' needs. However, the new business model (mobile, social, big data, and cloud) requires an agile development environment, speed to build, and consistency to quickly deploy complex applications. So, SSC comes with a preinstalled and certified operating system and application codes to provide a secure environment for deployment of your blockchain workloads.

So, you can run IBM Cloud Private and IBM Secure Service Container for IBM Cloud Private workloads on a secure platform on LinuxONE.

Secure Service Container for IBM Cloud Private provides an encrypted environment (data at rest, data in transit), with peer-to-peer and peer-to-host isolation. This approach protects container applications from access through hardware and operating system admin credentials, whether access is accidental or malicious, internal or external to an organization.

Secure Service Container for IBM Cloud Private provides these protections while it integrates with IBM Cloud Private. IBM Cloud Private is a Platform as a Service (PaaS) management stack that delivers rapid innovation and application modernization, investment leverage, enterprise integration. It also brings management and compliance features to containerized applications. Figure 2-3 shows different types of clouds that ICP and SSC are part of.

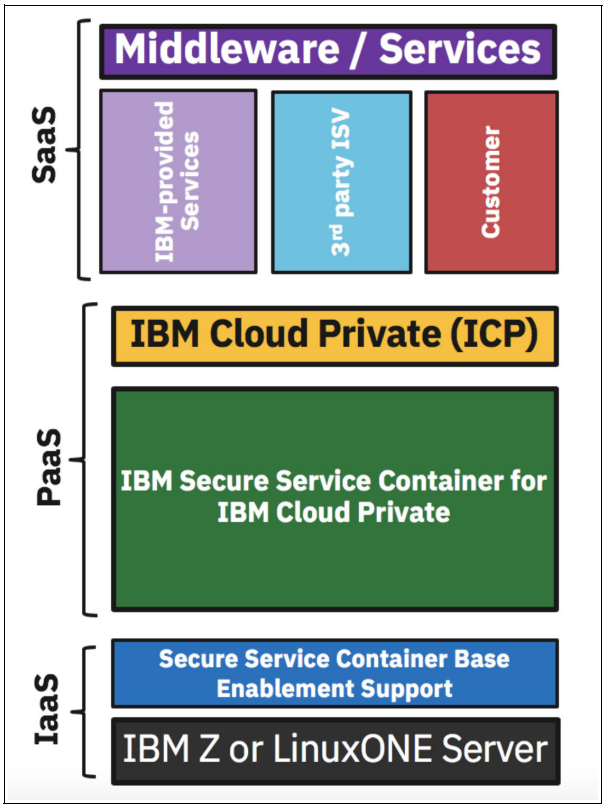


Figure 2-3 IBM Secure Service Container for IBM Cloud Private - Full-stack solution

To repeat, SSC improves on the traditional method of deploying applications. Most importantly, it prevents tampering with the appliance image and prevents load of the appliance at arbitrary times. Furthermore, the appliance code and data is protected/confidential while in transit and while at rest.

In addition, memory access is disabled and disk access is encrypted by default and these configurations cannot be disabled. The debugging data that system memory dumps provide is also encrypted. So, support groups can view only the application instructions in the dump, not any data that is included in the dump.

Finally, SSC makes your cloud infrastructure more secure and efficient for your blockchain deployments. This publication will consider use cases and implementations in the next sections.

2.2 Persistent Storage providers

For deployment of IBM Blockchain Platform (IBP), you must have mechanisms to persistently store data. This requirement applies to both an IBM Secure Service Container (SSC) and a non-SSC environment.

IBM Cloud Private offers many options for managing persistent storage within the cluster. IBM Cloud Private includes the following features:

- GlusterFS enterprise grade of storage to Kubernetes (K8s) pods. They offer ease of configuration, scaling, encryption support, replication, striping, and dynamic provisioning.

- vSphere Cloud Provider (vSphereVolume Plug-in) gives access to enterprise grade storage (vSAN, VMFS, vVol) that is native to and already supported by the VMware infrastructure.
- IBM Spectrum® Scale for solutions that are not hosted in VMware provides direct access to IBM block storage through dynamic provisioning.
- NFS provides a versatile and easy to use method of getting persistent storage to pods that is already available in most customer environments.
- HostPath is ideal for testing persistence in non-production environments. It isn't typically considered for shared or production environments.
- Ceph (Rook) is an industry proven option that can provide several storage options along with persistent volumes for Kubernetes.
- Minio is a lightweight, Amazon S3-compatible object storage server. Minio is best suited for storing unstructured data such as photos, videos, log files, backups, VMs, and container images.

IBM Cloud Private cluster needs to be prepared for data persistence. In the next sections, we discuss NFS and GlusterFS options that are available for running containerized blockchain applications.

2.3 Setting up file storage

The IBM Blockchain Platform for Multicloud blockchain console saves information such as logins and blockchain network management information that requires persistency. In contrast, a container's file system is ephemeral, meaning that after a container dies all of its memory goes with it.

To keep the information that they require, the applications need to write to persistent storage. This requires mounting part of the container file system to some volume. In Kubernetes, the construct for this is a persistent volume claim, which the developer defines as a volume mount for the pod, which the container uses to bind files or directories.

- This persistent volume claim binds to a persistent volume in a 1:1 mapping. For more information, see https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims
- This volume maps directly to a storage path. For more information, see https://kubernetes.io/docs/concepts/storage/persistent-volumes/
- A cluster administrator can create these persistent volumes directly in Kubernetes, but that is manual and time-consuming.
- Thus, the preferred method is a dynamic provisioner. For more information, see https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/
- A dynamic provisioner creates persistent volumes dynamically to satisfy incoming persistent volume claims that come by using its storage class. This provisioner essentially carves out space on the file system for the requested persistent volume claim. For more information, see https://kubernetes.io/docs/concepts/storage/storage-classes/
- There are 2 main options available for IBM Z on LinuxONE. These are Network File System (NFS) and Gluster File System (GlusterFS). In this document, we look at setting up dynamic provisioning for each of these options. For more information, see https://kubernetes.io/docs/concepts/storage/dynamic-provisioning/

2.3.1 Network File System (NFS)

NFS is an open-standard persistent file storage solution that is built into Linux (with RFCs for each different version of NFS). NFS allows client computers to access files from a remote server over a network. For more information, see https://developer.ibm.com/tutorials/l-network-filesystems/

NFS uses a server that has the **nfs** server package that is installed with a volume that client computers can access over the network that uses an **nfs** client package. This approach allows remote clients to mount remote volumes as local volumes, enabling easy access of files across multiple computers. In a clustered system like Kubernetes, this allows all nodes to keep up to date with the same storage. Thus, a pod can start on any node in the cluster and read its data. The NFS server must allow **mount** access from that node in its `/etc/exports` file. Figure 2-4 shows the NFS architecture.

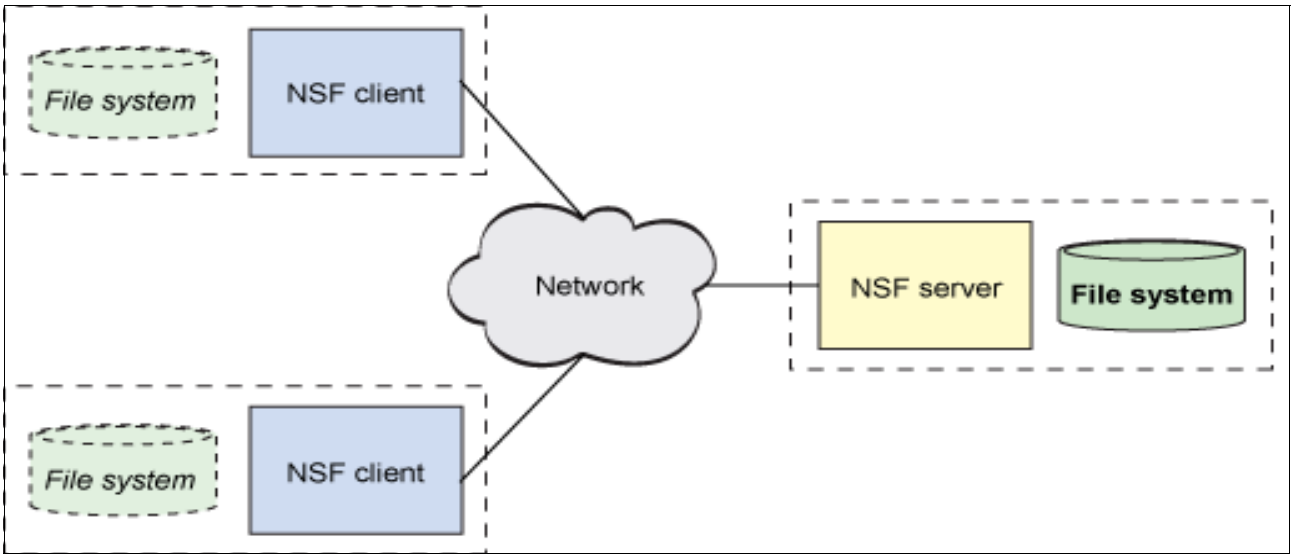


Figure 2-4 NFS architecture

Using a central server, NFS can be easy to manage and requires less resources than clustered file system alternatives. Additionally, NFS can grant users access to specific data with the granularity of splitting out different data at different mount points with different access policies for each mount point.

There are multiple versions of NFS with versions 3 and 4 being the predominant versions in use today. NFS, Version 4, includes a number of enhancements over version 3 including these options:

- Use Kerberos.
- Use only one IP port (2049).

The biggest limitation with NFS in Kubernetes is that it can stand as a single point of failure (SPOF). If the NFS server goes down, your Kubernetes cluster might stop. This scenario even negates some of the benefit of the replication that is done in the Kubernetes cluster to prevent a SPOF.

Note: You can avoid this limitation, for example, by deploying a clustered NFS solution.

Other notable limitations include the following points:

- Performance degradation during times of high network traffic.
- The use of RPCs (Remote Procedure Calls) for communication do not have the security that is needed to operate over the internet (in other words, they need to be behind a firewall).

Setting up NFS Dynamic Provisioners in Kubernetes

Kubernetes has 2 “out-of-tree” (not located in the core Kubernetes code) provisioners, which are available in the external storage folder of the Kubernetes Incubator GitHub at this URL: `https://github.com/kubernetes-incubator/external-storage`

One solution works with an existing nfs-server, and provides the client component for mounting volumes from this server when Kubernetes creates persistent volume claims. The other solution is an all-in-one solution that provides an nfs-server and client within Kubernetes. This exposes an nfs-server and mount volumes from this server when Kubernetes schedules a pod that uses this volume onto a node.

Choose the dynamic provisioner option that you prefer:

- Manage an external nfs-server. OR
- Allow Kubernetes to run it as a pod.

Then, complete the appropriate setup steps from one of the following sections:

Setting up Cluster Nodes with nfs-client package

Both NFS provisioners require the nfs-client package to be installed on each Kubernetes Worker node that will use NFS provisioning. This configuration is necessary to make mounts to the **nfs** directory so that pods that run on the worker nodes can read and write storage as clients to the **nfs** server.

On RHEL, run this command:

```
sudo yum install nfs-utils
```

On Ubuntu, run these commands:

```
sudo apt-get update
sudo apt-get install nfs-common
```

On SLES:

The nfs-client package is installed by default. There is no need to run any commands.

Note: Remember to install the nfs-client package on each worker node. Technically you need the package only on worker nodes that will use the storage. However, typically all worker nodes need the package.

Setting up NFS server primer

If an NFS server is not set up already, you can have an NFS server that runs separately from the Kubernetes cluster nodes. Set up the server by following the instructions for your Linux distribution.

- For RHEL instructions, see this web page:
  https://www.thegeekdiary.com/centos-rhel-7-configuring-an-nfs-server-and-nfs-client/
- For Ubuntu instructions, see this web page:
  https://help.ubuntu.com/lts/serverguide/network-file-system.html
- For SLES instructions, see this web page:
  https://www.suse.com/documentation/sles-12/book_sle_admin/data/sec_nfs_configuring-nfs-server.html

Here is a simplified version of how to do the setup steps:

1. Install the nfs-server package for your Linux distribution.

   On RHEL, run this command:

   ```
   sudo yum install nfs-utils rpcbind
   ```

   On Ubuntu, run this command:

   ```
   sudo apt update && sudo apt install nfs-kernel-server
   ```

   On SLES, run this command:

   ```
   zypper -n install nfs-kernel-server
   ```

2. Export a directory of the nfs-server (backed by file storage) to the worker nodes of the cluster by adding them to the `/etc/exports` file on the nfs-server linux instance. Then, load the directory.

   ```
   echo "<export_directory>
   <client_node_hostname_or_ip>(rw,fsid=0,insecure,no_subtree_check,async)" | sudo
   tee -a /etc/exports
   ```

   Add the client hostname or IP for each of the worker nodes. You can use a wildcard with hostnames such as `kubernetes-worker*` (if that was the hostname of each of your workers within your network) or an IP subnet mask (for example, **`192.168.10/24`**).

3. Start the NFS server:

   On RHEL, run these commands:

   ```
   sudo systemctl enable rpcbind
   sudo systemctl enable nfs-server
   sudo systemctl enable rpcbind
   sudo systemctl start nfs-server
   ```

   On Ubuntu, run this command:

   ```
   sudo systemctl start nfs-kernel-server.service
   ```

   On SLES, run these commands:

   ```
   sudo systemctl enable nfsserver
   sudo systemctl start nfsserver
   ```

NFS client provisioner (for use with an existing NFS server)

This solution works with an existing (external) nfs-server. It provides the client component for mounting volumes from this server when Kubernetes schedules a pod to a node using the created StorageClass.

For the home page of the NFS Client provisioner, visit this web page:
https://github.com/kubernetes-incubator/external-storage/tree/master/nfs-client

We will be working with an IBM Cloud Private package that includes a Helm chart and docker images.

This package is available on the GitHub site for this book as a release at this web page:
https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform-for-Multicloud

Here are the steps to configure and run the nfs-client dynamic provisioner

1. Download the package from the book GitHub site by running this command:

   curl -LO https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform-for-Multicloud/releases/download/v2.0.0/nfs-client-provisioner-bundle.tgz

   Typical output is shown in Example 2-1.

   Example 2-1 Downloading the package

```
curl -LO
https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain
-Platform-for-Multicloud/releases/download/v2.0.0/nfs-client-provisioner-bundle
.tgz
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   624    0   624    0     0   2091      0 --:--:-- --:--:-- --:--:--  2093
100 31.7M  100 31.7M    0     0  11.3M      0  0:00:02  0:00:02 --:--:-- 16.6M
```

2. Load the nfs-client package into IBM Cloud Private as follows:

   a. Log in to your IBM Cloud Private cluster to the `kube-system` namespace.

      ```
      cloudctl login -a https://<cluster_CA_domain>:8443 -n kube-system
      --skip-ssl-validation
      ```

      cloudctl login -a https://mycluster.icp:8443 -n kube-system --skip-ssl-validation

   b. Ensure that the Docker CLI is configured. After you configure the Docker CLI, access the image registry on your cluster by running the following command:

      ```
      docker login <cluster_CA_domain>:8500
      ```

      docker login mycluster.icp:8500

      Note: If you get an x509 error, see the *Configuring authentication for Docker CLI IBM Cloud Private* page to resolve:
      https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/manage_images/configuring_docker_cli.html

   c. Import the Helm chart by using the command line. Go to the directory where you stored the downloaded Helm chart package from the curl command earlier. Run the following command in the IBM Cloud Private CLI to import the Helm chart into your IBM Cloud Private cluster.

```
cloudctl catalog load-archive --archive nfs-client-provisioner-bundle.tgz
--registry <cluster_CA_domain>:8500/kube-system --repo mgmt-charts
```

```
cloudctl catalog load-archive --archive nfs-client-provisioner-bundle.tgz
--registry mycluster.icp:8500/kube-system --repo mgmt-charts --username
admin --password super_secure_password
```

Note: If you are using macOS, add `--username value` and `--password value`, or set these values in the environment as **DOCKER_USER** and **DOCKER_PASSWORD**.

Example 2-2 shows typical output for this command.

Example 2-2 Output of the command

```
Expanding archive
OK

Importing docker image(s)
  Processing image: nfs-client-provisioner-s390x:latest
    Loading Image
    Tagging Image
    Pushing image as:
mycluster.icp:8500/kube-system/nfs-client-provisioner-s390x:latest
...
Uploading helm chart(s)
  Processing chart: charts/nfs-client-provisioner-multiarch-2.tgz
  Updating chart values.yaml
  Uploading chart
Loaded helm chart
OK

Synch charts
Synch started
OK
```

d. Click **Catalog** in the IBM Cloud Private console, and then search for `nfs-client`. If the import was successful, the `nfs-client` tile is visible on the IBM Cloud Private Catalog page as in Figure 2-5.

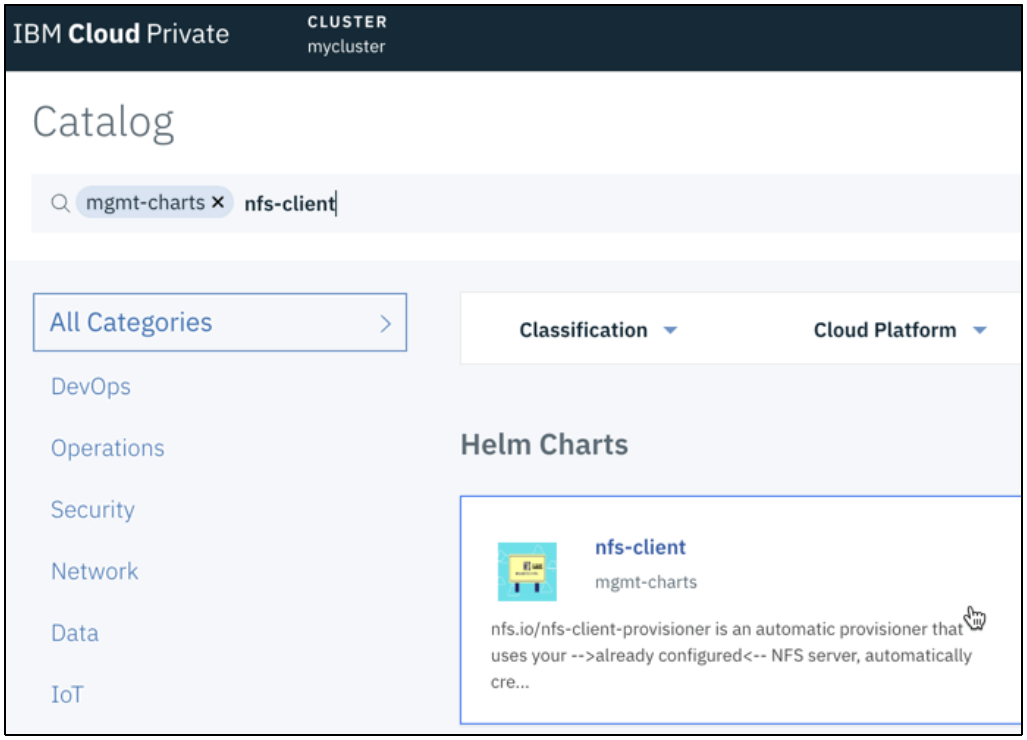


Figure 2-5 Setting up NFS server primer -1

3. Install the Helm chart through IBM Cloud Private. Click the **nfs-client** tile as in Figure 2-5, and configure the Helm chart with the following values for your cluster. We documented the figures as follows:
   - You see callout letters beside each field where you enter a value.
   - Each callout letter maps to a description in these steps.
   - At the end of each description, the default value is displayed in parentheses ().
   - Bolded values are variables that you might have to change.
   - You can apply the unbolded values in your environment if you follow the configuration that is described here.

   Values for callout letters a to c apply to Figure 2-6 on page 31.

Table 2-1 Values for callout letters a to c

a. `Helm release name` is the name for the helm release. Make sure that each helm release in the cluster has a unique name. Otherwise, you get an error, and you must redeploy the release to a different name. (nfsc)
b. `Target Namespace` is the namespace where the helm release resources will be deployed. This chart requires the `kube-system` namespace if callout letter **`j`**, **Run as system-cluster-critical** is selected. [This is the recommended option. It gives the provisioner priority over the pods that rely on it.] (kube-system)
c. `Target Cluster` is the cluster that you are targeting with the deployment. (local-cluster).

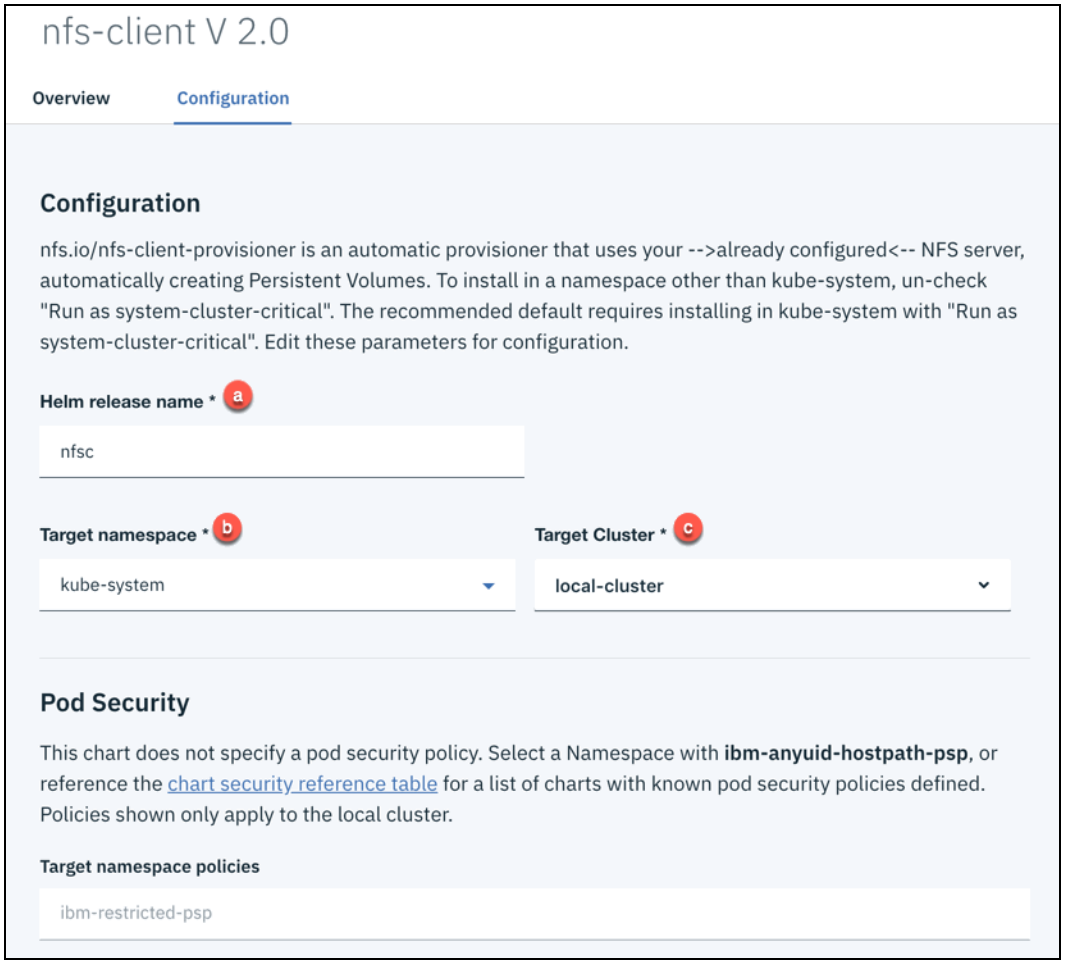


Figure 2-6 Setting up NFS server primer -2

The next values for callout letters d to f apply to Figure 2-7 on page 32.

Table 2-2 The next values, for callout letters d to f

d. `replicaCount` is the number of client provisioner pods that are created by the deployment. You can use this number to scale your provisioner as desired. But one pod works fine in most cases. (1)
e. `NFS Server Hostname` or IP is the Hostname or IP of the NFS Server that is exporting one of its directories. (9.16.8.23)
f. `NFS Server Mount Path` is the path to mount the directory that the NFS Server is exporting to use for Kubernetes pods. This value must be the path that you exported (in /etc/exports) on the NFS Server to all of your Kubernetes worker nodes. (/export/blockchain)

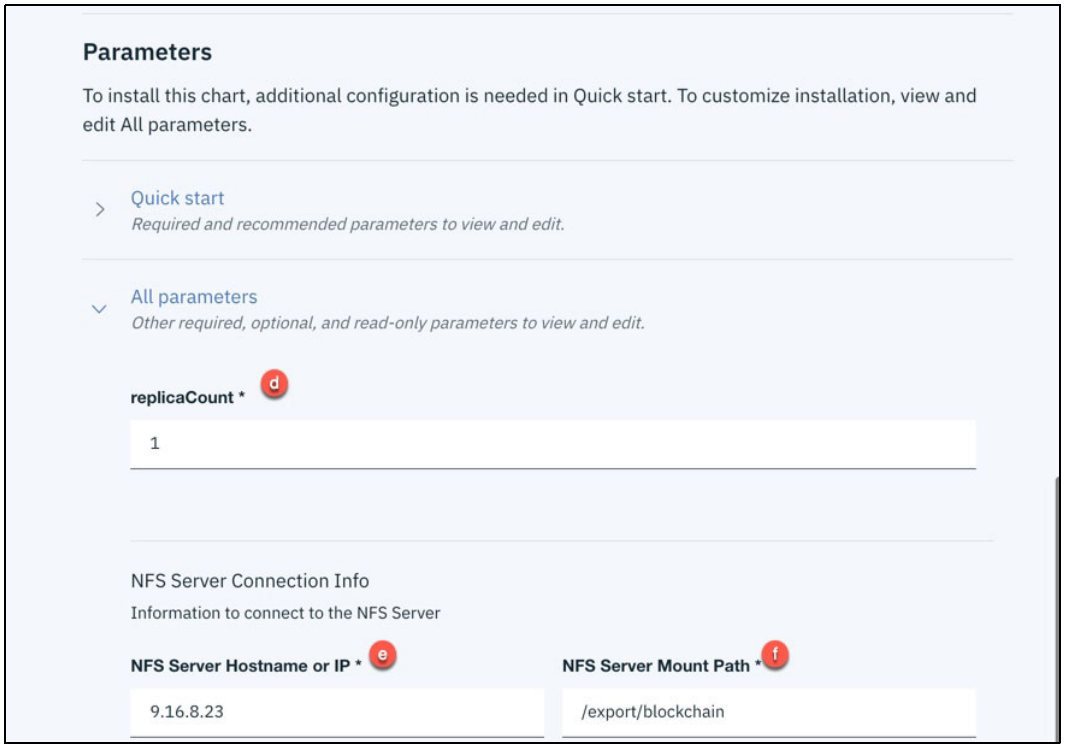


Figure 2-7 Setting up NFS server primer -3

The final values for callout letters g to k apply to Figure 2-8 on page 33.

Table 2-3 The final values, for callout letters g to k

g. `storageClassName` is the name of the StorageClass to create in Kubernetes by using the `nfs.io/nfs-client-provisioner`. (managed-nfs-storage)

h. `Make new Storage Class the default` makes the StorageClass that was created by the Helm chart into the default StorageClass in Kubernetes. This means that if a developer asks for a PersistentVolumeClaim and does not provide a StorageClass, the one created with storageClassName from callout letter **g** is used. It is good to have a default StorageClass to make PersistentVolume provisioning as easy as possible. However, if you have a different StorageClass that you want as default instead, uncheck this box. (checked)

i. `archiveOnDelete` signifies whether you want to archive PersistentVolumeClaims when they are deleted. (unchecked)

j. `Run as system-cluster-critical` is the recommended option that gives the provisioner priority over pods that rely on it. This prevents the problematic situation where the provisioner is evicted before the workloads that require it, which can lead to storage issues. (checked)
Note: If you select this option, remember to use kube-system as the `Target Namespace` value — shown by callout letter `b` in Figure 2-8 on page 33 — unless the deployment will not run.

k. `Resources` are set to recommended defaults that should work for most environments. If you need to change these values, remember that the limits cannot be lower than the requests.
Note: Limits and requests occur on a per container basis. This means that making multiple pods through changing the replicaCount of the helm release requires fulfilling the storage quota for each of those pods separately.

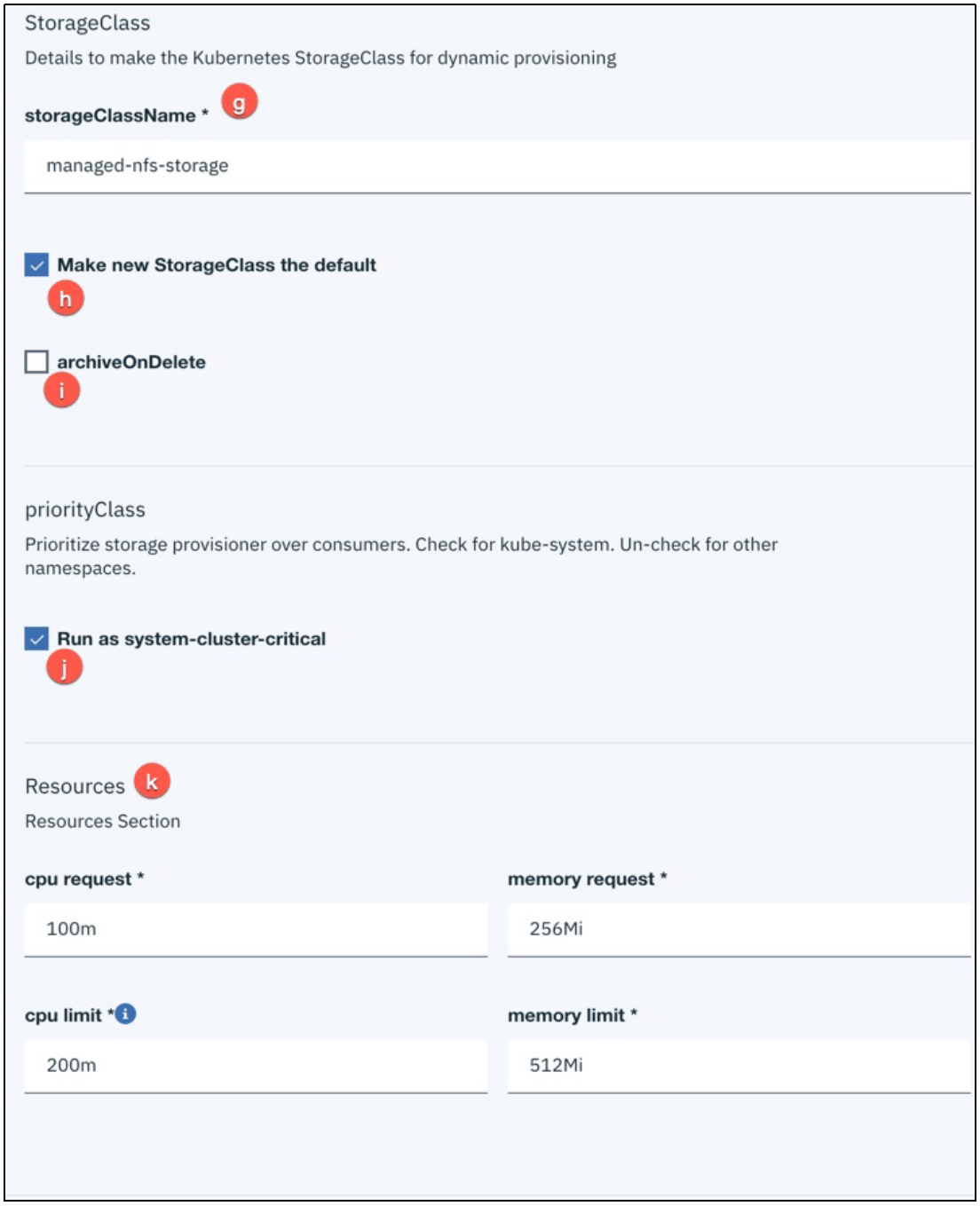


Figure 2-8 Setting up NFS server primer -4

After you select the values that your environment requires, install the chart.

4. Verify the installation by checking in the output for your helm release that the deployment is available and the test-pod has completed through either a) the UI or b) kubectl.

> **Note:** A test-pod is deployed by this release whose name begins with `test-pod-`. This pod uses a PersistentVolumeClaim that uses the newly created StorageClass to dynamically provision its PersistentVolume. The pod runs the busybox container and writes **SUCCESS** to the Persistent Volume. When this pod shows **Completed** as its status, it verifies that the dynamic provisioner is active for a basic use case.

a. Verify with the UI by following the example in Figure 2-9.

Note: It might take a minute or two for your environment to reach the statuses shown in Figure 2-9.

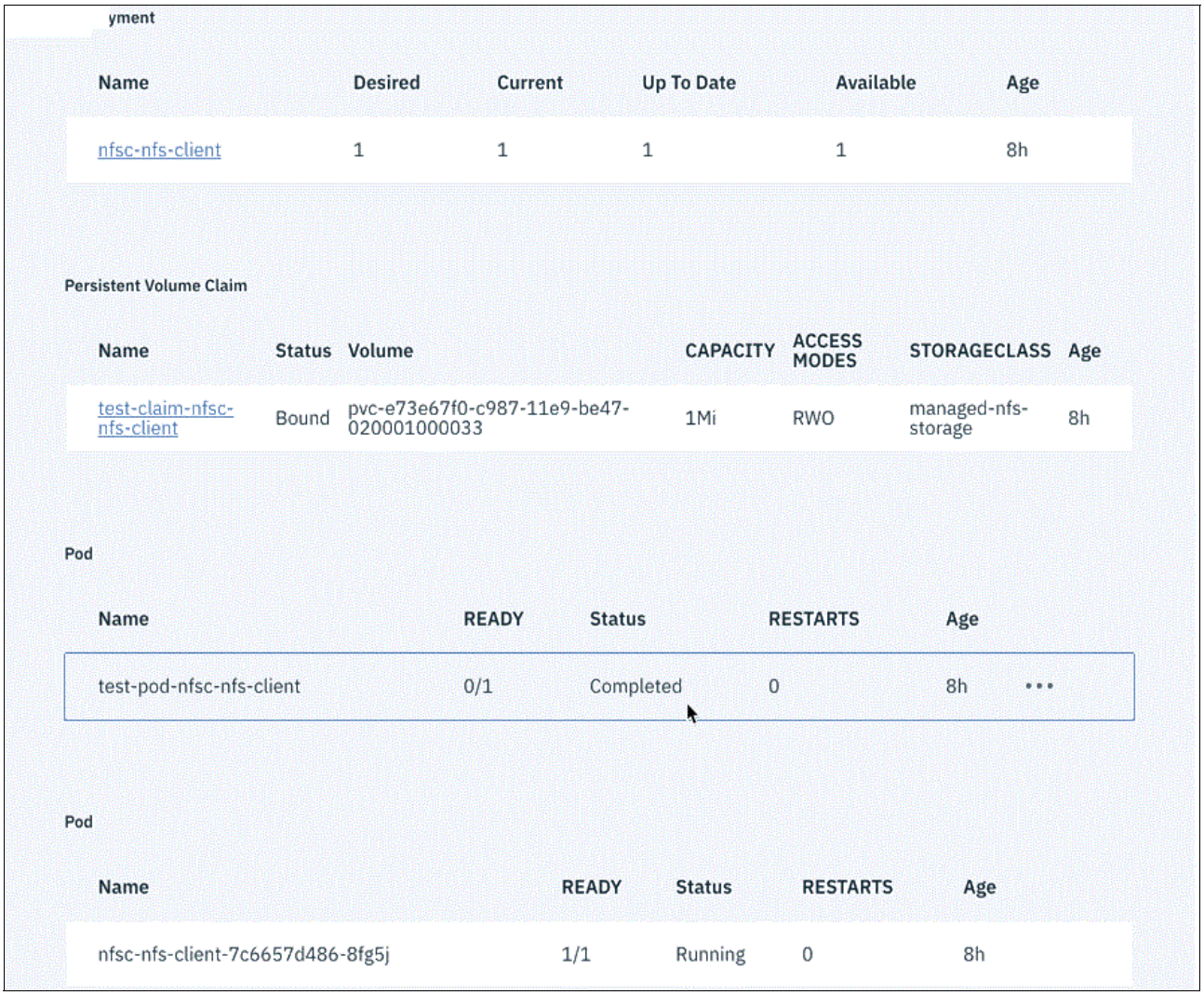


Figure 2-9 Setting up NFS server primer -5

b. Verify with kubectl by following the guidance in this step:

Note: If you deviate from the recommended path and install the helm release for nfs-client in a namespace other than `kube-system`, follow these conventions:

- Assign a name for the namespace is a value *other than* `kube-system`.
  `use -n <namespace_name>`
  where <*namespace_name*> is a value *other than* `kube-system`, as in this example statement:
  `use -n default`
- Double check that `Run as system-cluster-critical` was unchecked when you deployed the helm release. In the current version of IBM Cloud Private, that setting works with the `kube-system` namespace only.

i. Run the following command to check the pods and deployment of the nfs-client-provisioner. Typical output for the command is also shown:

```
kubectl get deploy,pods -n kube-system -l app=nfs-client
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/nfsc-nfs-client   1/1     1            1
3m51s

NAME                                    READY   STATUS    RESTARTS   AGE
pod/nfsc-nfs-client-7c6657d486-8fg5j    1/1     Running   0          3m50s
```

ii. Run the following command to check that the test pod has “Completed” successfully. Typical output for the command is also shown:

```
kubectl get pods -n kube-system -l app=test-nfs-client
NAME                        READY   STATUS      RESTARTS   AGE
test-pod-nfsc-nfs-client    0/1     Completed   0          3m24s
```

iii. Run the following command to view the newly created StorageClass and confirm that it has been made the default. Typical output for the command is also shown:

```
kubectl get sc
NAME                            PROVISIONER                    AGE
fully-managed-nfs-storage       full-nfs-provisioner           4d11h
glusterfs                       kubernetes.io/glusterfs        33h
image-manager-storage           kubernetes.io/no-provisioner   64d
logging-storage-datanode        kubernetes.io/no-provisioner   64d
managed-nfs-storage (default)   nfs.io/nfs-client-provisioner  3m34s
mongodb-storage                 kubernetes.io/no-provisioner   64d
```

NFS server and client provisioner

This section describes the process for the all-in-one solution. It provides an nfs-server and client. This configuration exposes an nfs-server and mount volumes from this server when Kubernetes schedules a pod to a node by using this volume.

For the home page of the NFS provisioner visit this web page:
https://github.com/kubernetes-incubator/external-storage/tree/master/nfs

Set provisioner git clone.

In this document, we work with an IBM Cloud Private package that includes a Helm chart and docker images. This package is available on the GitHub site for this book as a release at this web page:
https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform-for-Multicloud

Here are the steps to configure and run the nfs-client dynamic provisioner:

1. Download the package from the book GitHub site by running this command:

```
curl -LO
https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform-for-Multicloud/releases/download/v2.0.0/nfs-full-provisioner-bundle.tgz
```

Typical output is shown in Example 2-1.

Example 2-3 Downloading the package

```
curl -LO
https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Pl
atform-for-Multicloud/releases/download/v2.0.0/nfs-full-provisioner-bundle.tgz
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   622    0   622    0     0   1817      0 --:--:-- --:--:-- --:--:--  1813
100  229M  100  229M    0     0  19.1M      0  0:00:11  0:00:11 --:--:-- 21.3M
```

2. Load the nfs-client package into IBM Cloud Private as follows:

   a. Log in to your IBM Cloud Private cluster to the `kube-system` namespace.

      ```
      cloudctl login -a https://<cluster_CA_domain>:8443 -n kube-system
      --skip-ssl-validation
      ```

      cloudctl login -a https://mycluster.icp:8443 -n kube-system --skip-ssl-validation

   b. Ensure that the Docker CLI is configured. After you configure the Docker CLI, access the image registry on your cluster by running the following command:

      ```
      docker login <cluster_CA_domain>:8500
      ```

      docker login mycluster.icp:8500

      > **Note:** If you get an x509 error, see the *Configuring authentication for Docker CLI IBM Cloud Private* page to resolve the error. https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/manage_images/configuring_docker_cli.html

   c. Import the Helm chart by using the command line. Navigate to the directory where you stored the downloaded Helm chart package from the curl command in step 1. Then, run the following command in the IBM Cloud Private CLI to import the Helm chart into your IBM Cloud Private cluster.

      ```
      cloudctl catalog load-archive --archive nfs-full-provisioner-bundle.tgz
      --registry <cluster_CA_domain>:8500/kube-system --repo mgmt-charts
      ```

      cloudctl catalog load-archive --archive nfs-full-provisioner-bundle.tgz --registry mycluster.icp:8500/kube-system --repo mgmt-charts --username admin --password super_secure_password.

      > **Note:** If you are using macOS, add `--username value` and `--password value`, or set these values in the environment as **DOCKER_USER** and **DOCKER_PASSWORD**.

      Example 2-4 shows typical output for this command:

Example 2-4 Output of the command

```
Expanding archive
OK

Importing docker image(s)
  Processing image: nfs-provisioner-s390x:latest
    Loading Image
    Tagging Image
```

```
    Pushing image as: mycluster.icp:8500/kube-system/nfs-provisioner-s390x:latest
  Processing image: nfs-provisioner-amd64:latest
    Loading Image
    Tagging Image
    Pushing image as: mycluster.icp:8500/kube-system/nfs-provisioner-amd64:latest
    Creating manifest list as:
...
Uploading helm chart(s)
  Processing chart: charts/nfs-full-2.0.0.tgz
  Updating chart values.yaml
  Uploading chart
Loaded helm chart
OK

Synch charts
Synch started
OK

Archive finished processing
```

d. Click **Catalog** in the IBM Cloud Private console. Then, search for `nfs-full`. If the import was successful, the `nfs-full` tile is visible on the IBM Cloud Private Catalog page as shown in Figure 2-10 on page 37.

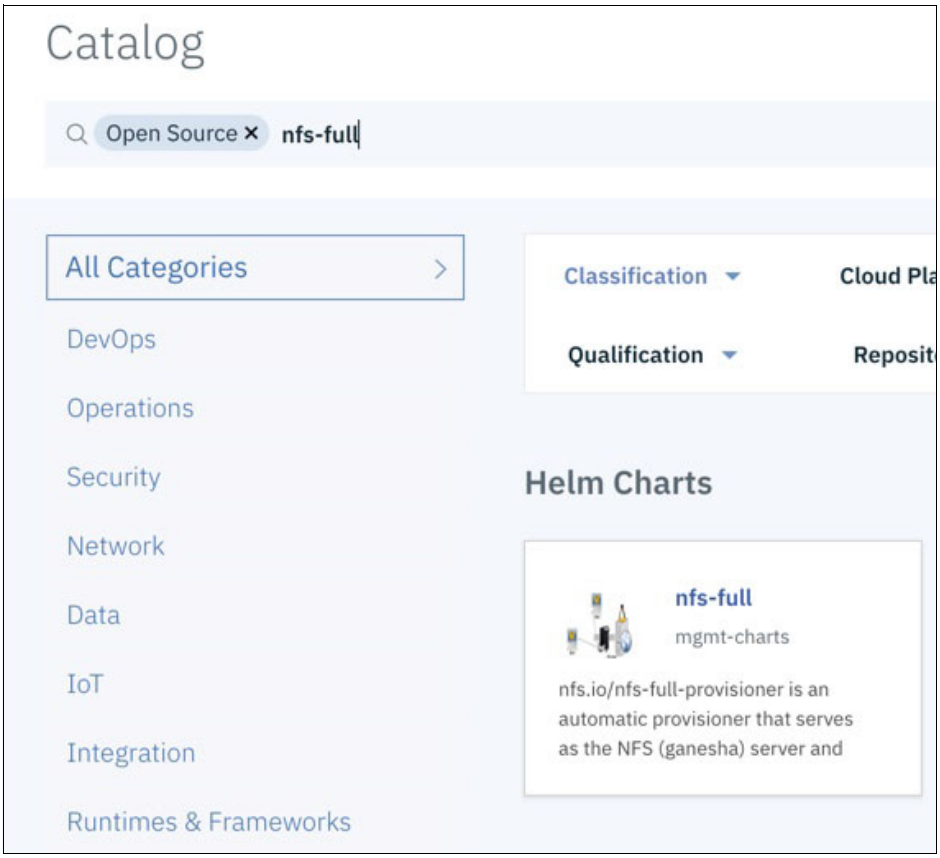


Figure 2-10 Setting up NFS server primer -6

3. Install the Helm chart through IBM Cloud Private. Click the **nfs-full** tile as in Figure 2-10. Configure the Helm chart with the values for your cluster that are listed in this step. We documented the figures as follows:

- You see callout letters beside each field where you enter a value.
- Each callout letter maps to a description in these steps.
- At the end of each description, the default value is displayed in parentheses ().
- Bolded values are variables that you might have to change.
- You can apply the unbolded values in your environment if you follow the configuration that is described here.

Values for callout letters a to c apply to Figure 2-11.

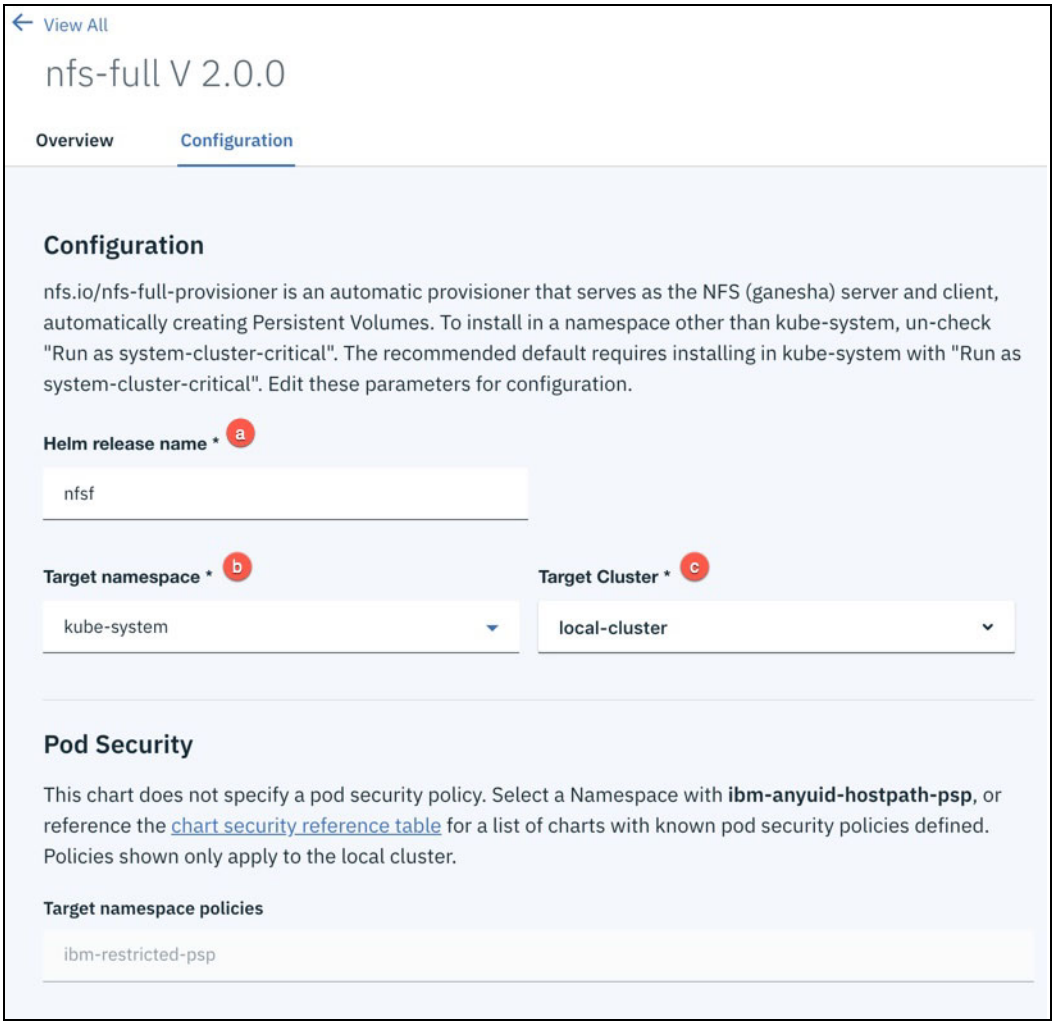


Figure 2-11 Setting up NFS server primer -7

Table 2-4 Values, for callout letters a to c

a. `Helm release name` is the name for the helm release. Make sure that each helm release in the cluster has a unique name. Otherwise, you get an error and you must redeploy the release to a different name. (nfsf)
b. `Target Namespace` is the namespace where the helm release resources will be deployed. This chart requires the `kube-system` namespace in callout letter i. "`Run as system-cluster-critical`" [This is the recommended option, which gives the provisioner priority over pods that rely on it] is selected. (kube-system)
c. `Target Cluster` is the cluster that you are targeting with the deployment. (local-cluster)

Values for callout letters d to f apply to Figure 2-12 on page 39.

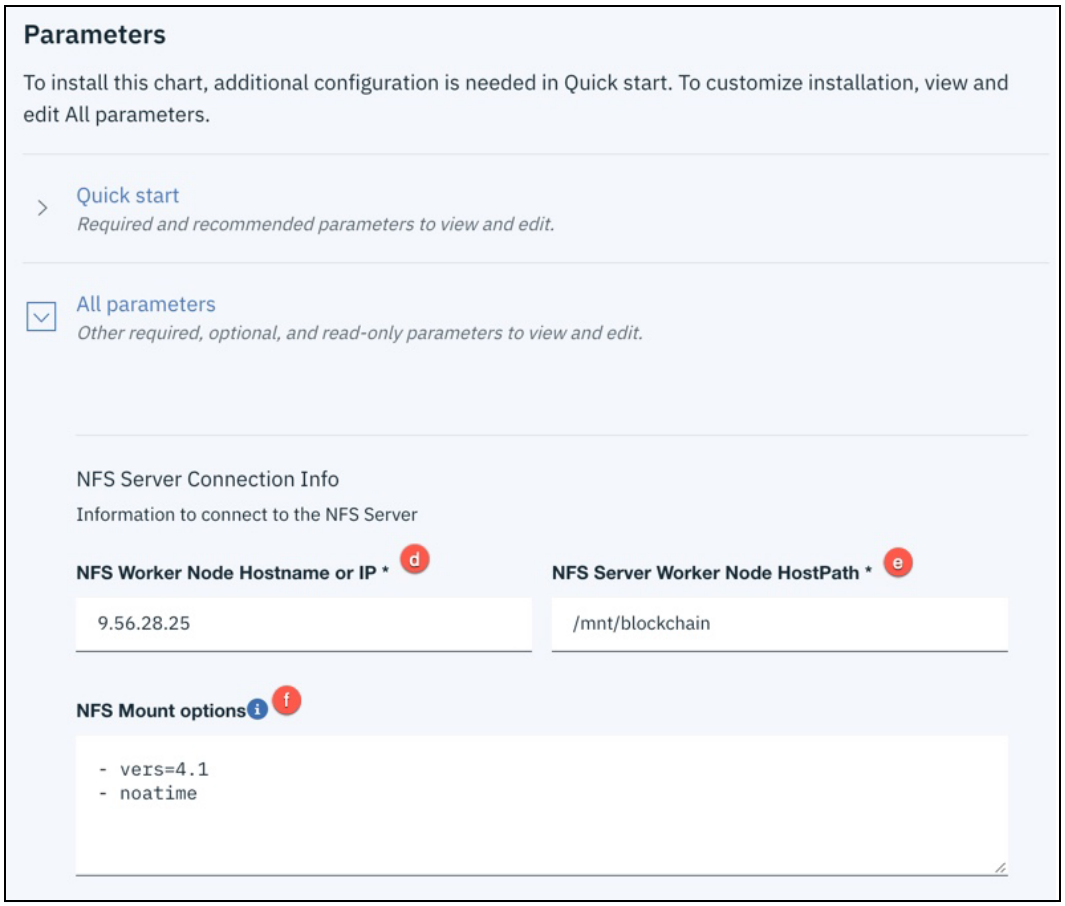


Figure 2-12 Setting up NFS server primer -8

Table 2-5 Values, for callout letters d to f

d. `NFS Worker Node Hostname or IP` is the Hostname or IP of Worker Node that will host the NFS Server pod that is exporting one of its directories. (9.16.8.23)
e. `NFS Server Worker Node HostPath` is the path to the local storage the NFS server will use as the basis of its nfs export. Make sure that you have a storage device that is attached to the cluster worker node with the hostname or IP in callout letter d. This is the directory NFS Server pod is exporting to use as storage for Kubernetes pods. (/mnt/blockchain)
f. `NFS Mount Options` are NFS mount options that you want for your nfs mounts that the provisioner makes to the NFS Server provided in callout letter `d`, “NFS Server Hostname or IP” above. The defaults are vers=4.1 and noatime. You should keep the defaults, unless you have specific options that you want to use. (vers=4.1, noatime)

The final values for callout letters g to j apply to Figure 2-13 on page 40.

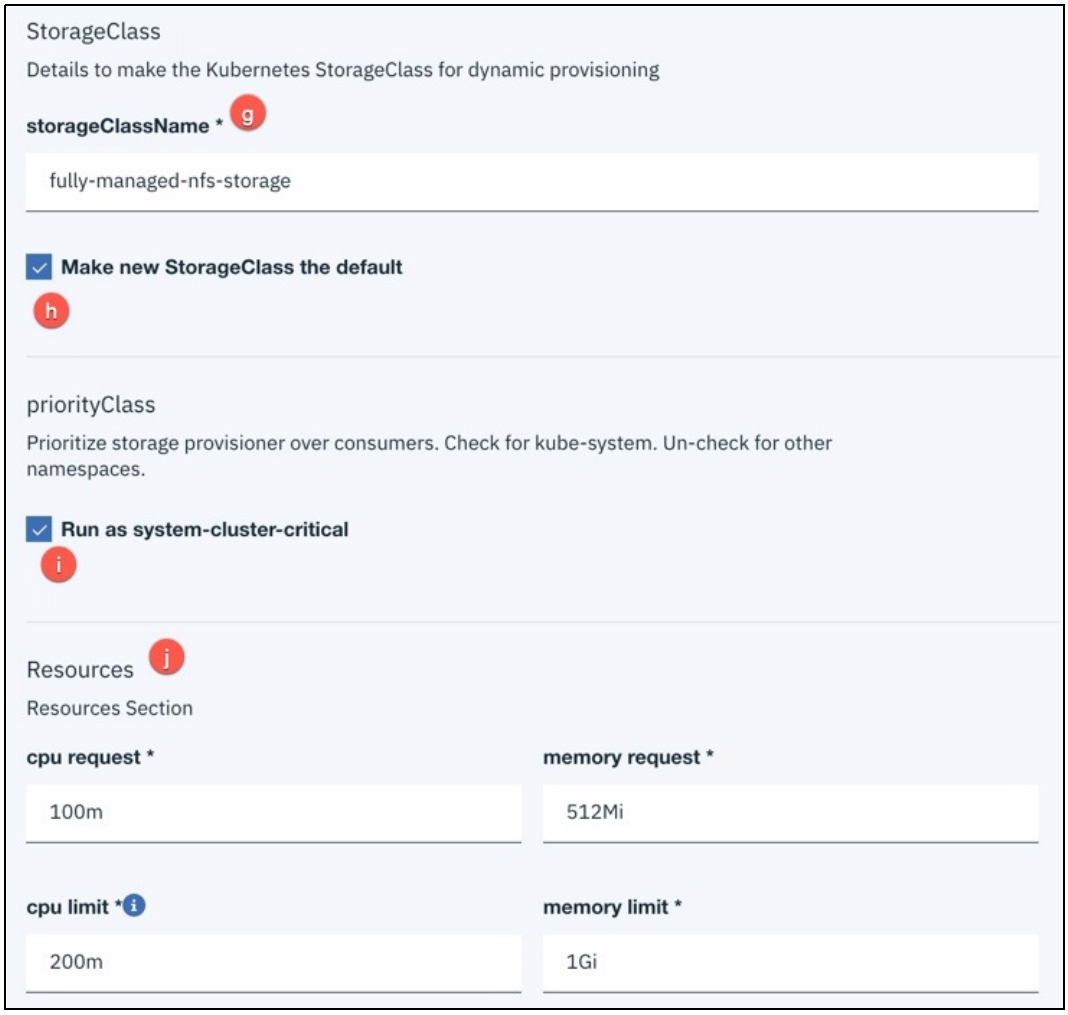


Figure 2-13 Setting up NFS server primer -9

Table 2-6 Final values, for callout letters g to j

g. `storageClassName` is the name of the StorageClass to create in Kubernetes by using the `nfs.io/nfs-full-provisioner`. (fully-managed-nfs-storage) h. `Make new Storage Class the default` makes the StorageClass that was created by the Helm chart the default StorageClass in Kubernetes. This means that if a developer asks for a PersistentVolumeClaim and does not provide a StorageClass, the one created with `storageClassName` from callout letter `g` is used. It is good to have a default `StorageClass` to make PersistentVolume provisioning as easy as possible. (checked) (However, if you have a different `StorageClass` that you want as default instead, uncheck this box.) i. `Run as system-cluster-critical` is the recommended option that gives the provisioner priority over pods that rely on it. This prevents the problematic situation where the provisioner is evicted before the workloads that require it, which can lead to storage issues. (checked) **Note:** If you select this option, remember to use kube-system as the Target Namespace value — shown by callout letter `b` in Figure 2-11 on page 38 — unless the deployment does not run. j. `Resources` are set to recommended defaults that should work for most environments. If you need to change these values, remember that the limits cannot be lower than the requests.

After you select the values for your system, install the chart.

4. Verify the installation by checking in the output for your helm release that the deployment is available and the test-pod has completed through either a) the UI or b) kubectl.

Note: There is a test-pod that is deployed by this release whose name begins with `test-pod-`. This pod uses a PersistentVolumeClaim using the newly created StorageClass to dynamically provision its PersistentVolume. The pod runs the busybox container and writes **SUCCESS** to the Persistent Volume. This pod that shows **Completed** as its status verifies that the dynamic provisioner is in business for a basic use case.

a. Verify with the UI by following the example in Figure 2-14.

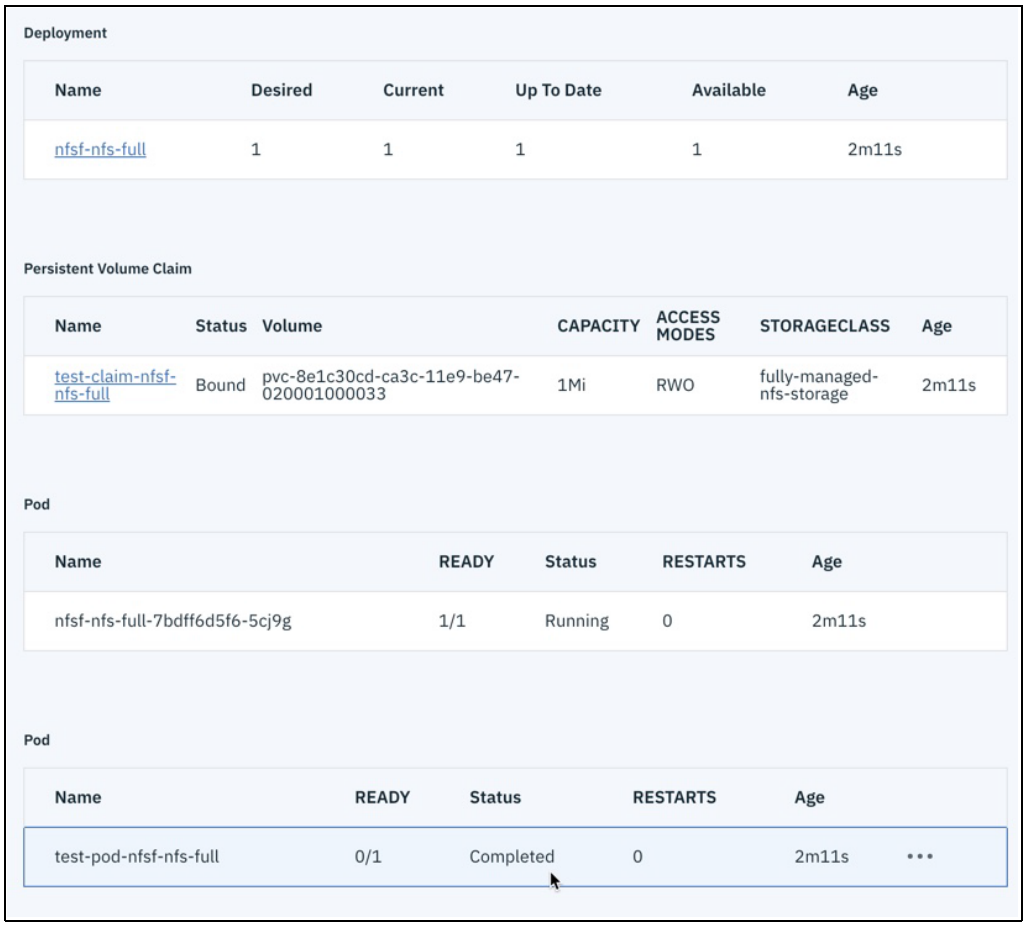


Figure 2-14 Setting up NFS server primer -10

Note: It might take a minute or two for your environment to reach the statuses shown in Figure 2-14

b. Verify with kubectl by following the guidance in this step:

Note: If you deviate from the recommended path and install the helm release for nfs-client in a namespace other than `kube-system`, follow these conventions:

- Assign a name for the namespace is a value *other than* `kube-system`.
  `use -n <namespace_name>`
  where <*namespace_name*> is a value *other than* `kube-system`, as in this example statement:
  `use -n default`
- Double check that `Run as system-cluster-critical` was unchecked when you deployed the helm release. In the current version of IBM Cloud Private, that setting works with the `kube-system` namespace only.

i. Run the following command to check the pods and deployment of the nfs-full-provisioner. Typical output for the command is also shown:

```
kubectl get pod,deploy -l app=nfs-full
NAME                                  READY   STATUS    RESTARTS   AGE
pod/nfsf-nfs-full-7bdff6d5f6-w9ld9    1/1     Running   0          2m43s

NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/nfsf-nfs-full    1/1     1            1           2m44s
```

ii. Run the following command to check that the test pod has “Completed” successfully. Typical output for the command is also shown:

```
kubectl get pod,deploy -l app=test-nfs-full
NAME                          READY   STATUS      RESTARTS   AGE
pod/test-pod-nfsf-nfs-full    0/1     Completed   0          3m18s
```

iii. Run the following command to view the newly created StorageClass and confirm that it has been made the default. Typical output for the command is also shown:

```
kubectl get sc
NAME                                  PROVISIONER                    AGE
fully-managed-nfs-storage (default)   nfs.io/nfs-full-provisioner    3m25s
glusterfs                             kubernetes.io/glusterfs        2d6h
image-manager-storage                 kubernetes.io/no-provisioner   65d
logging-storage-datanode              kubernetes.io/no-provisioner   65d
mongodb-storage                       kubernetes.io/no-provisioner   65d
```

2.3.2 Gluster File System (GlusterFS)

GlusterFS is an open source clustered file storage solution (scale-out network attached storage file system) designed by Gluster, Inc. now formerly RedHat (since its acquisition in 2011). Instead of having code directly in the kernel like NFS, GlusterFS is a File System in User Space (FUSE). In FUSE, the file system code runs in user space and makes calls to the FUSE module, which connects it to the kernel interfaces.

GlusterFS uses Heketi, which is the RESTful management interface that manages your GlusterFS volumes. Heketi enables the administrator to check the state of its gluster file system with a number of commands. The commands can show everything from the overall topology of the scale-out network to the state of the current cluster, nodes, and devices.

Kubernetes has a GlusterFS provisioner plug-in that is built in that creates GlusterFS persistent volumes in response to persistent VolumeClaims for storage. There are many

types of storage types for GlusterFS, but the suggested one for Kubernetes out of the box is replication mode. When replication mode is set, each node in the cluster replication set creates a replicated brick (which is an exported directory on storage device in the pool) for this volume, for redundancy and high availability. Figure 2-15 shows the GlusterFS architecture with each GlusterFS node, including backing storage devices to which bricks write.

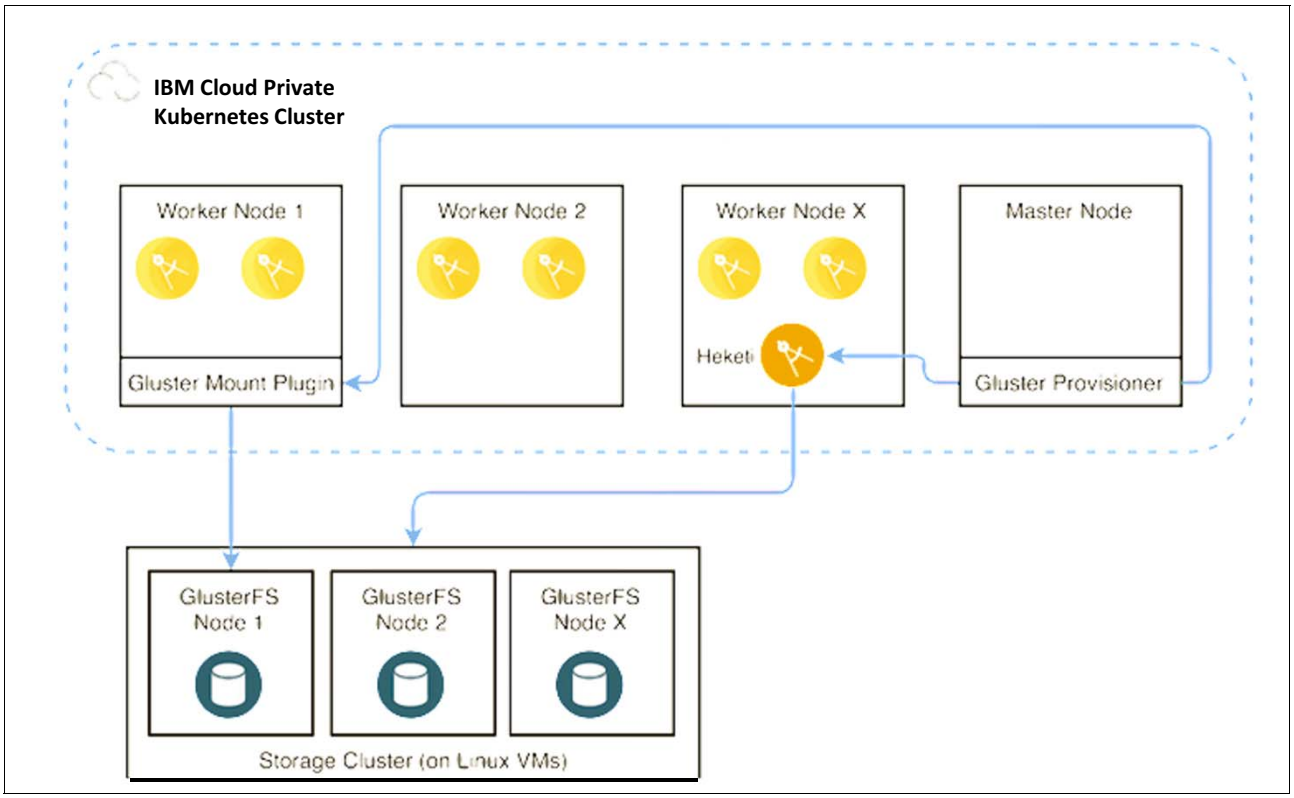


Figure 2-15 GlusterFS architecture

GlusterFS linearly scales well and provides replication to prevent a single point of failure. This feature provides a good solution for high availability as part of a clustered node system. Keep in mind that multiple nodes also mean that nodes need to agree on state. This requirement can lead to split-brain scenarios that lead to performance degradation, especially when multiple small writes occur on the same sets of data in short periods of time. Also, multiple nodes mean that you have multiple nodes to manage in terms of backup and other management tasks.

Note: The GlusterFS storage nodes can either be set up on their own storage node cluster with a storage host group or located on various worker nodes. The example architecture here shows the GlusterFS nodes on their own dedicated storage cluster host group. The following example deploys these on regular worker nodes for the Redbooks development environment. To deploy on a separate host group follow instructions in labeling nodes for that approach.

Setting up GlusterFS Nodes

To use GlusterFS for dynamic provisioning later, you must first prepare all of your worker nodes to use GlusterFS.

GlusterFS Client Installation

For each node to properly use GlusterFS storage, each GlusterFS storage node needs to have `dm_thin_pool` configured, and each worker node needs to have the GlusterFS Client installed. In the following example, the worker node hosts the GlusterFS storage so both `dm_think_pool` and the `GlusterFS client` are configured on the same node. The following steps describe how to do this.

Note: The examples in this book use RedHat Linux on s390x. The instructions for other Linux distributions and architectures follow the same patterns, but the commands are slightly different. For the commands for other Linux distributions and other computer architectures, see the IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/manage_cluster/prepare_nodes.html.

1. Configure the `dm_thin_pool` kernel module.

   ```
   echo dm_thin_pool | sudo tee -a /etc/modules-load.d/dm_thin_pool.conf
   dm_thin_pool
   ```

2. Install GlusterFS Client as in Example 2-5.

Example 2-5 Install GlusterFS client

```
rpm --import
https://oplab9.parqtec.unicamp.br/pub/key/openpower-gpgkey-public.asc

rpm -q gpg-pubkey --qf '%{NAME}-%{VERSION}-%{RELEASE}\t%{SUMMARY}\n'
gpg-pubkey-fd431d51-4ae0493bgpg(Red Hat, Inc. (release key 2)
<security@redhat.com>)
gpg-pubkey-2fa658e0-45700c69gpg(Red Hat, Inc. (auxiliary key)
<security@redhat.com>)
gpg-pubkey-75b92fa7-5ccf44cbgpg(OpenPower Unicamp Lab (OpenPower)
<openpower@ic.unicamp.br>)

sudo yum install yum-utils && sudo yum-config-manager --add-repo
https://oplab9.parqtec.unicamp.br/pub/s390x/glusterfs/rhel/7.6/
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager
rhel-7-for-system-z-extras-rpms                          | 2.1 kB     00:00
rhel-7-for-system-z-optional-rpms                        | 2.0 kB     00:00
rhel-7-for-system-z-rpms                                 | 2.0 kB     00:00
rhel-7-for-system-z-supplementary-rpms                   | 2.0 kB     00:00
...

sudo yum update && sudo yum install glusterfs-client gluster-cli
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
              : manager

rhel-7-for-system-z-extras-rpms                          | 2.1 kB     00:00
rhel-7-for-system-z-optional-rpms                        | 2.0 kB     00:00
rhel-7-for-system-z-rpms                                 | 2.0 kB     00:00
rhel-7-for-system-z-supplementary-rpms                   | 2.0 kB     00:00
Resolving Dependencies
--> Running transaction check
---> Package atomic-registries.s390x 1:1.22.1-26.gitb507039.el7 will be updated
---> Package atomic-registries.s390x 1:1.22.1-28.gitb507039.el7 will be an
update
```

```
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
=
 Package Arch  Version                    Repository                       Size
================================================================================
=
Updating:
 atomic-registries
         s390x 1:1.22.1-28.gitb507039.el7 rhel-7-for-system-z-extras-rpms  35 k

Transaction Summary
================================================================================
=
Upgrade  1 Package

Total download size: 35 k
Is this ok [y/d/N]: y
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
atomic-registries-1.22.1-28.gitb507039.el7.s390x.rpm       |  35 kB   00:00
....
```

GlusterFS disk preparation

After the node is ready for GlusterFS, you must attach disks for the backing storage to be used by the solution. For each storage node, we prepare these disks for installation of GlusterFS with the following steps:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/manage_cluster/prepare_disks.html#symlink

Note: The following instructions work when symlinks are automatically generated for devices such as in RHEL or Ubuntu. For SLES, you must create the symlinks manually by following the instructions at: https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/manage_cluster/prepare_disks.html#manual

1. Identify storage devices to use (must have at least 25 GB capacity) in Example 2-6.

Example 2-6 identify storage devices to use

```
sudo fdisk -l

Disk /dev/dasdb: 7 MB, 7372800 bytes, 1800 sectors
Units = sectors of 1 * 4096 = 4096 bytes
Sector size (logical/physical): 4096 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
...
Disk /dev/dasdo: 44.3 GB, 44311265280 bytes, 10818180 sectors
Units = sectors of 1 * 4096 = 4096 bytes
Sector size (logical/physical): 4096 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

```
Disk /dev/dasdp: 44.3 GB, 44311265280 bytes, 10818180 sectors
Units = sectors of 1 * 4096 = 4096 bytes
Sector size (logical/physical): 4096 bytes / 4096 bytes
I/O size (minimum/optimal): 4096 bytes / 4096 bytes
```

The continued example uses devices **/dev/dasdo** and **/dev/dasdp** from Example 2-6.

2. Wipe the selected device(s) of all file-system, RAID, and partition-table signatures.

```
sudo wipefs --all --force /dev/dasdo
sudo wipefs --all --force /dev/dasdp
```

3. Get the symlink of the device as shown in Example 2-7.

Example 2-7 Get the symlink

```
ls -altr /dev/disk/*
/dev/disk/by-uuid:
total 0
drwxr-xr-x. 5 root root 100 Aug 12 17:24 ..
lrwxrwxrwx. 1 root root  12 Aug 12 17:24 a5bf998c-7272-49f9-afe2-928a81ee2014 -> ../../dasda2
lrwxrwxrwx. 1 root root  12 Aug 12 17:24 54be6c22-c5d3-4702-a55d-a0dda944566d -> ../../dasda1
lrwxrwxrwx. 1 root root  10 Aug 12 17:24 5be05cbe-6c75-4bee-ab70-89352cf7112f -> ../../dm-0
lrwxrwxrwx. 1 root root  10 Aug 12 17:24 ce0d209a-0746-4be5-8ce1-8fa7870da765 -> ../../dm-1
drwxr-xr-x. 2 root root 120 Aug 14 13:37 .

/dev/disk/by-path:
total 0
drwxr-xr-x. 5 root root 100 Aug 12 17:24 ..
lrwxrwxrwx. 1 root root  11 Aug 12 17:24 ccw-0.0.0202 -> ../../dasdc
lrwxrwxrwx. 1 root root  11 Aug 12 17:24 ccw-0.0.0100 -> ../../dasda
lrwxrwxrwx. 1 root root  11 Aug 12 17:24 ccw-0.0.0203 -> ../../dasdh
lrwxrwxrwx. 1 root root  12 Aug 12 17:24 ccw-0.0.0203-part1 -> ../../dasdh1
...
lrwxrwxrwx. 1 root root  11 Aug 14 14:46 ccw-0.0.0501 -> ../../dasdp
lrwxrwxrwx. 1 root root  12 Aug 14 16:35 ccw-0.0.0501-part1 -> ../../dasdp1
lrwxrwxrwx. 1 root root  11 Aug 15 17:06 ccw-0.0.0500 -> ../../dasdo
drwxr-xr-x. 2 root root 700 Aug 15 17:06 .
lrwxrwxrwx. 1 root root  12 Aug 15 20:56 ccw-0.0.0500-part1 -> ../../dasdo1
```

4. Save the full path (link path and symlink) to use when you provision GlusterFS.

Note: Any of the following link paths are acceptable:
`/dev/disk/by-path, /dev/disk/by-id, /dev/disk/by-uuid, or /dev/disk/by-label.`

The full path above for the partition of `dasdo` from part 2) is
`/dev/disk/by-path/ccw-0.0.0500-part1`

The full path above for the partition `dasdp` from part 2) is
`/dev/disk/by-path/ccw-0.0.0501-part1`

Note: If you use more than one node (default is 3), find storage devices on each node by following the same process for the other nodes in the storage group. This process is shown above and is detailed further at this web page: https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/manage_cluster/prepare_disks.html#manual.

Label GlusterFS node and create the Heketi secret

If you use existing Worker Nodes for GlusterFS, label these nodes with GlusterFS:

```
kubectl label nodes <node1_ipaddr node2_ipaddr node3_ipaddr...>
storagenode=glusterfs
```

For the example node in this document, this amounts to the following command:

```
kubectl label nodes 9.56.28.25 storagenode=glusterfs
```

Next, create a heketi-secret with an admin password with this command:

```
kubectl create secret generic heketi-secret -n kube-system
--type='kubernetes.io/glusterfs' --from-literal=admin_password='<my-password>' &&
kubectl label secret heketi-secret -n kube-system glusterfs="heketi-secret"
```

Here is a literal example of this command:

```
kubectl create secret generic heketi-secret -n kube-system
--type='kubernetes.io/glusterfs' --from-literal=admin_password='super_secure' &&
kubectl label secret heketi-secret -n kube-system glusterfs="heketi-secret"
```

Setting up GlusterFS Dynamic Provisioner in Kubernetes

GlusterFS is an in-tree dynamic provisioner (internal provisioner) in Kubernetes. This means that there is a plug-in that enables dynamic provisioning after the cluster administrator creates a proper storage class to connect to backing glusterfs storage. In IBM Cloud Private, to deploy glusterfs and set up this storage class easily, you can either run an add-on command or use the Helm chart, which does the setup for you.

GlusterFS Helm release

The GlusterFS Helm chart deploys one management Heketi server and a daemonset of GlusterFS servers, with one server on each worker node. The Heketi server sets up a GlusterFS topology with the backing devices that are prepared in the “GlusterFS disk preparation” on page 45 section. It also sets up the GlusterFS storage class to take advantage of this newly deployed GlusterFS storage system. The GlusterFS Helm chart is here: https://github.com/IBM/charts/tree/master/stable/ibm-glusterfs

The IBM Cloud Private installer automatically loads the glusterfs Helm chart to the local mgmt-charts repository of the IBM Cloud Private cluster as part of the IBM Cloud Private installation. You can find this chart in the catalog by searching for glusterfs as shown in Figure 2-16.

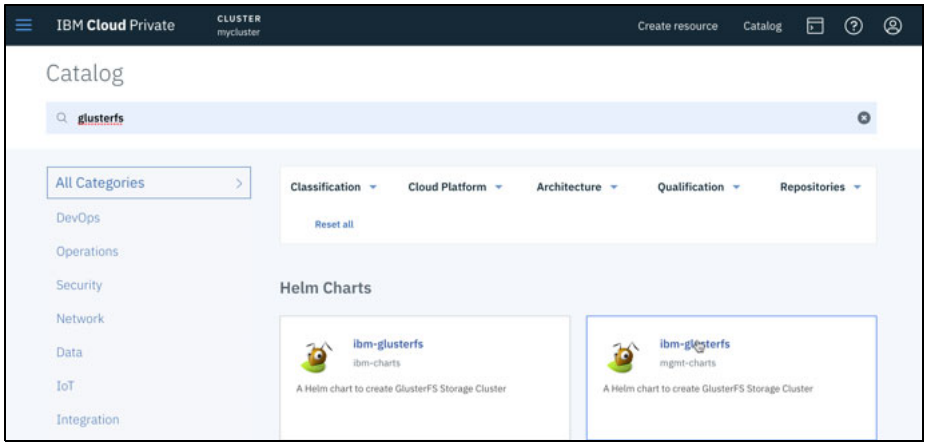


Figure 2-16 GlusterFS Helm chart

Configure the chart with the following values for your cluster. We have labeled the pictures with callout numbers that map to each labeled value. The default value is shown entered in parentheses (). Values that have no callout number can be safely left at their defaults.

The first three values apply to Figure 2-17 on page 48.

Table 2-7 First 3 values, for callout numbers 1 to 3

1. **Helm release name** is the name for the helm release. Make sure that each helm release in the cluster has a unique name. Otherwise, you get an error, and you must redeploy the release to a different name. (glusterfs-storage).
2. **Target namespace** is the namespace where the helm release resources will be deployed. This chart requires the kube-system namespace. (kube-system).
3. **License** is a mandatory box to check, indicating your consent to the License agreement. (checked).

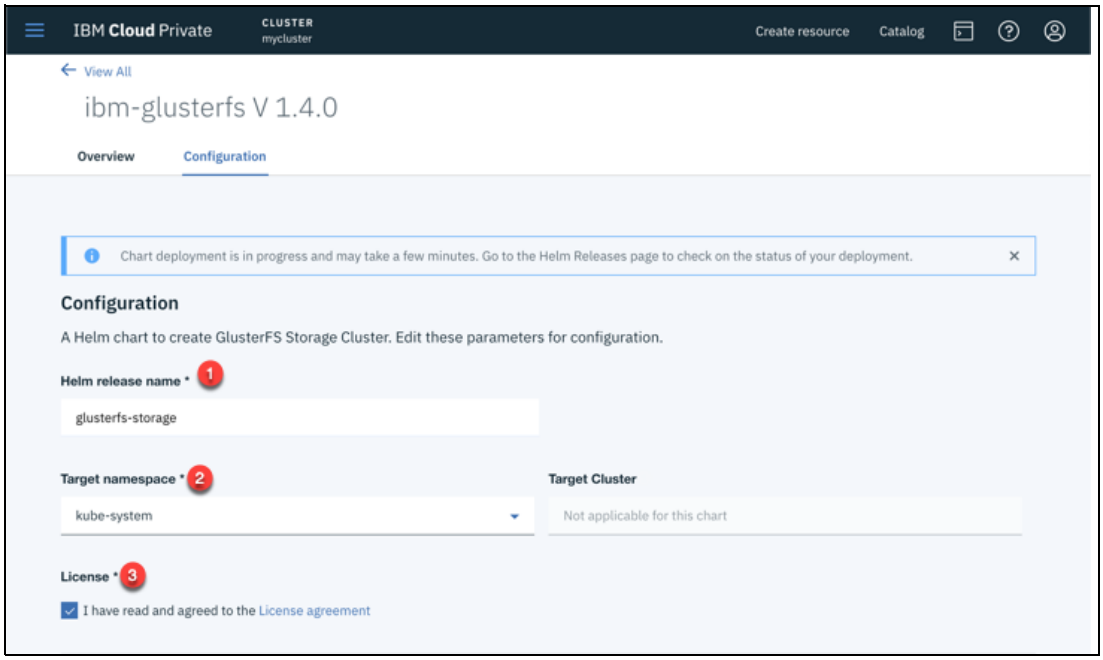


Figure 2-17 Configuring the ibm-glusterfs Helm chart -1

Values 4 and 5 apply to Figure 2-18.

Table 2-8 Values for callout numbers 4 and 5

4. The **All Parameters** drop-down menu gives access to all of the parameters for the Helm chart. We need to click it. [All parameters Drop-down open].
5. **Heketi Topology** is the layout of the backing devices that are used to provision glusterfs volumes per node. In this case, there are two volumes (with paths that were found through the *ls -altr /dev/disk/** command earlier.) If there were more nodes, each node would have its own devices for storage. A typical setup is three nodes, each with at least one device each for replication. (The topology that is shown in the Figure is edited from sample topology of a 3-node cluster in the Helm chart).

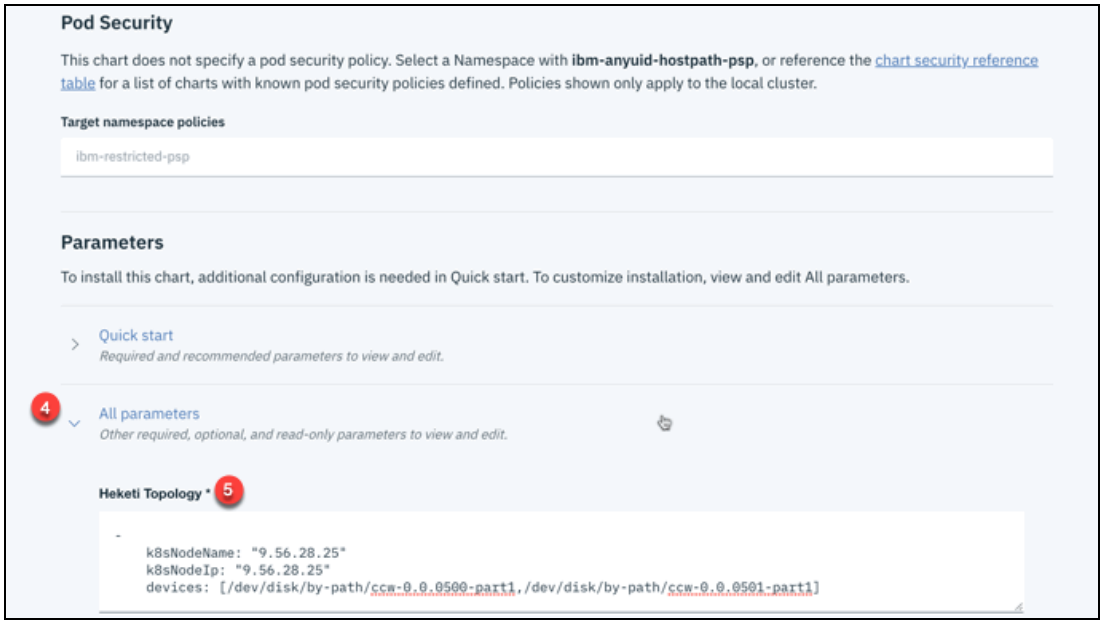


Figure 2-18 Configuring the ibm-glusterfs Helm chart -2

Values 6-8 specify which architecture nodes to schedule on. Choose the architectures of the nodes you want to use for storage as [3-Most preferred] as in Figure 2-19 on page 50.

Table 2-9 Values for callout numbers 6 to 8

6. amd64 scheduling preference (used to schedule on x86 nodes) [0 - Do not use].
7. ppc64le scheduling preference (used to schedule on power nodes) [0 - Do not use].
8. s390x scheduling preference (used to schedule on IBM LinuxONE/ IBM Z nodes) [3 -Most preferred].

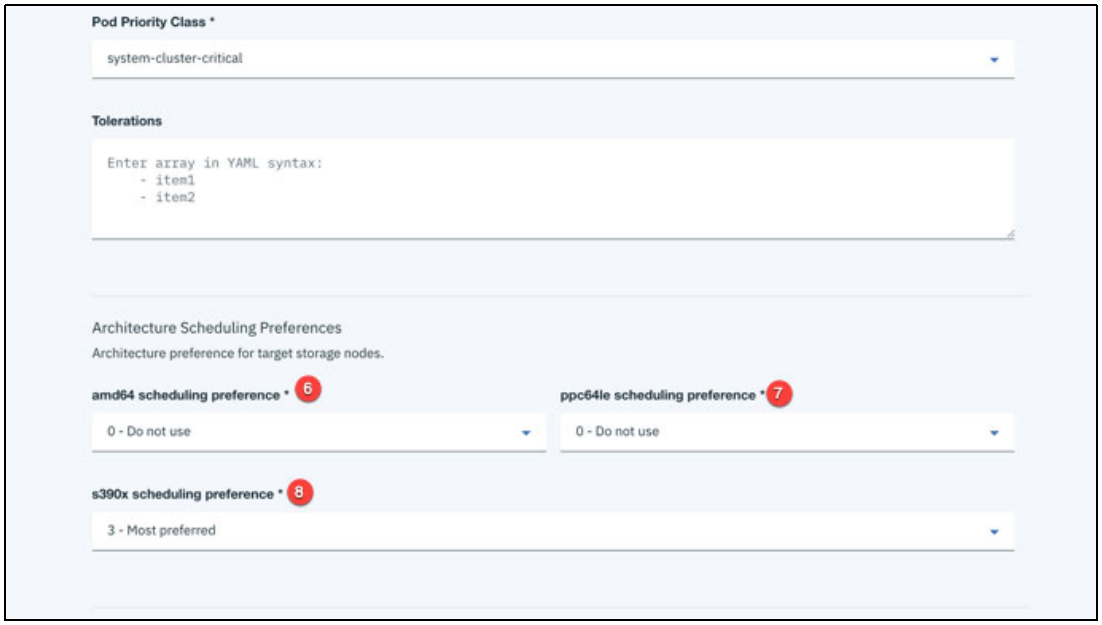


Figure 2-19 Configuring the ibm-glusterfs Helm chart -3

The next set of values are left at their defaults, so there are no values for the user to enter (and thus no callout numbers) in Figure 2-20.

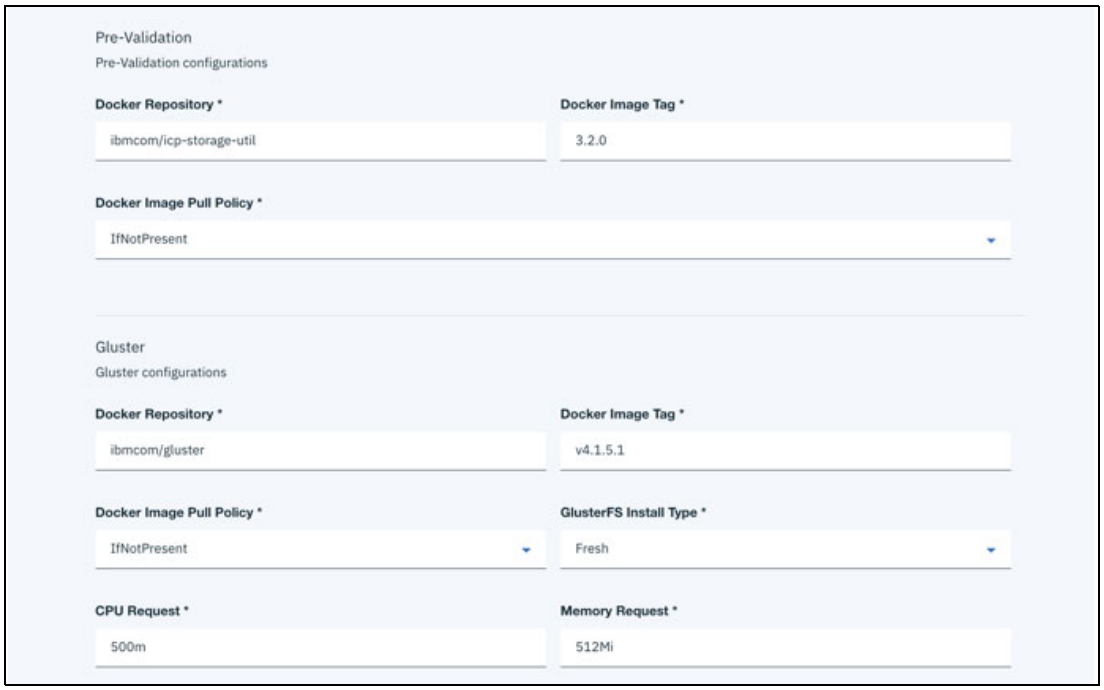


Figure 2-20 Configuring the ibm-glusterfs Helm chart -4

Table 2-10 Values for callout numbers 9 to 12

9. See Figure 2-21.
   Authentication Secret is the name of the secret that holds the `admin_password` for heketi (`heketi-secret`).
   Note: If you do not remember the name of the secret that you created, run this command to find it:
   `kubectl get secret --field-selector=type='kubernetes.io/glusterfs' -n kube-system`
10. See Figure 2-22 on page 52.
    TLS Secret Name is the name of the TLS secret that the Helm chart will create with the IBM Cloud Private CA. [glusterfs-tls-secret].
11. See Figure 2-23 on page 52.
    Volume Type refers to the Volume Type that was used to create GlusterFS volumes. This value is set in the GlusterFS storage class that was created by this chart. The recommended configuration is to use Replicate:3 with 3 different nodes above. The testing configuration here just uses one node, and so replication is set to none. (none).
    Note: Failure occurs if you set replication with only one node, even if there are multiple devices on this node such as the two devices set to the one node in the example.
12. See Figure 2-24 on page 53.
    Key is the key that is used for the NodeSelector, which identifies the nodes that make up the GlusterFS storage cluster. This should match the label that is on all the nodes that will host the actual GlusterFS storage volumes for use by Kubernetes applications. (storagenode).

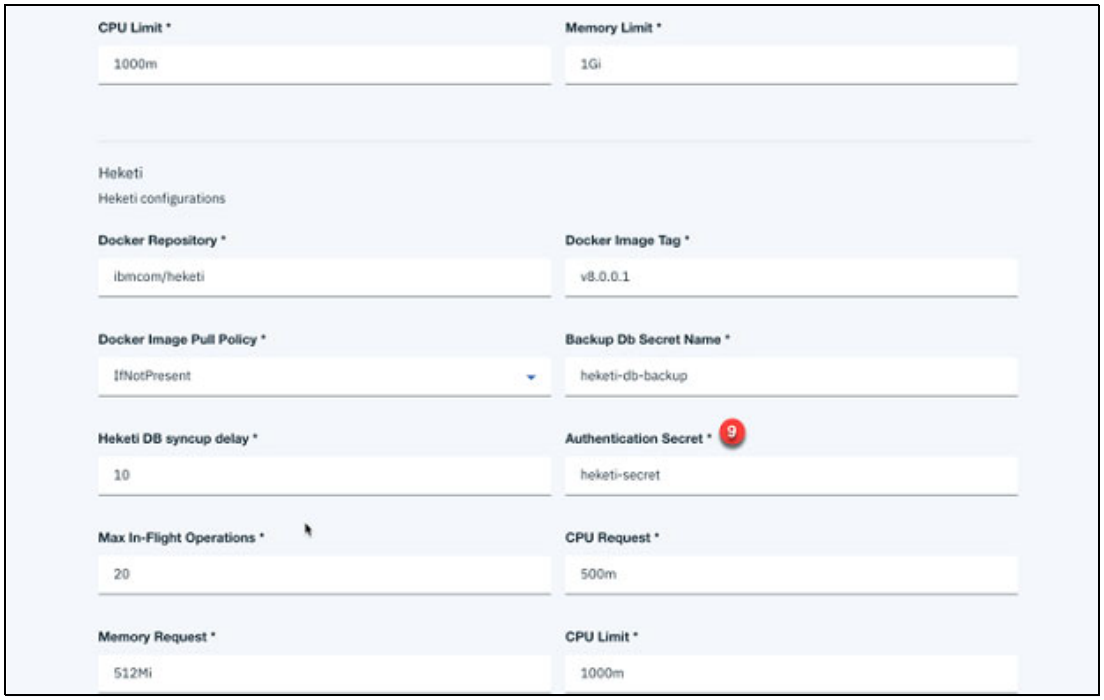


Figure 2-21 Configuring the ibm-glusterfs Helm chart -5

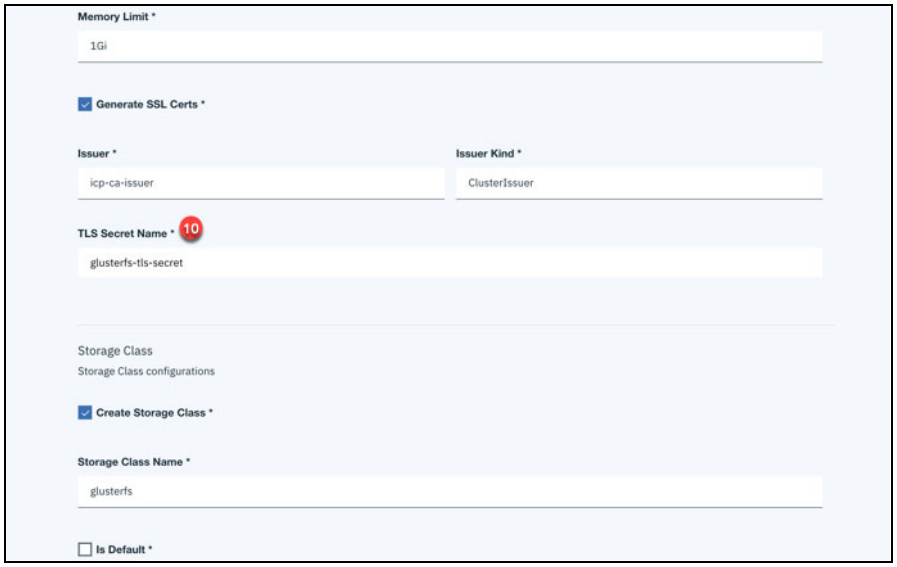


Figure 2-22 Configuring the ibm-glusterfs Helm chart -6

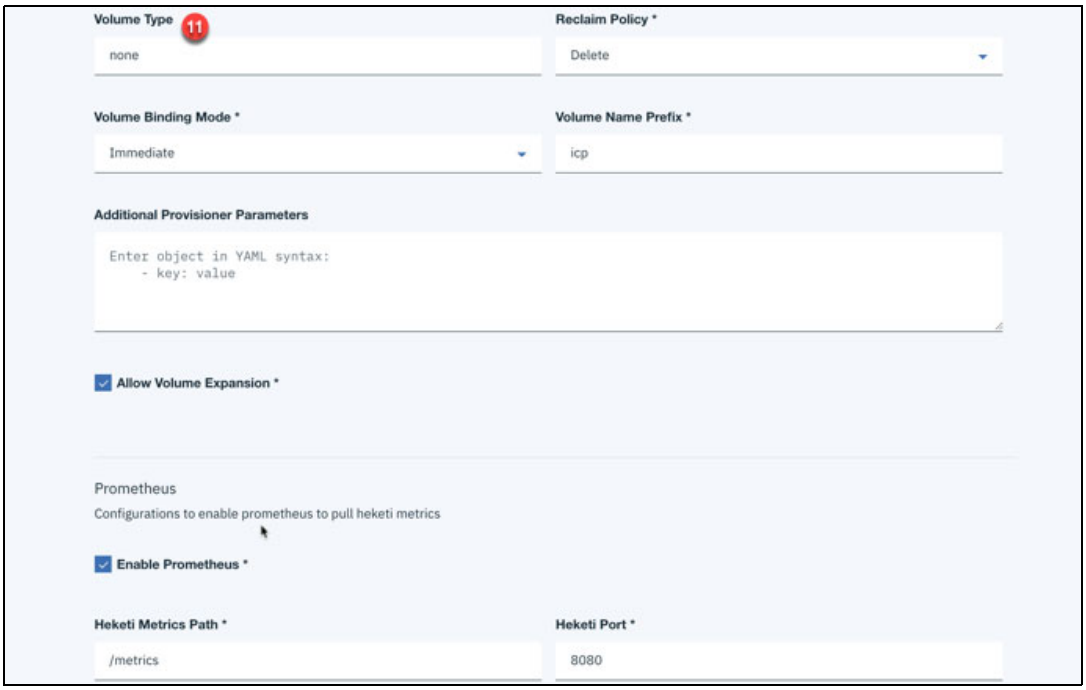


Figure 2-23 Configuring the ibm-glusterfs Helm chart -7

Key is the key that is used for the NodeSelector, which identifies the nodes that make up the GlusterFS storage cluster. This should match the label that is on all the nodes that will host the actual GlusterFS storage volumes for use by Kubernetes applications. (storagenode).
Note: You can check the labels on your nodes with this command:
`kubectl get nodes --show-labels`

Figure 2-24 Configuring the ibm-glusterfs Helm chart -8

Verification of Gluster storage

You can check the helm release and components in the UI by selecting the helm release from the helm releases section in the UI. (Navigate there through **Workloads** → **Helm Releases** by using the navigation bar in the UI). You can click the different components to inspect how the release is progressing. The output of the helm release should be similar to Figure 2-25.

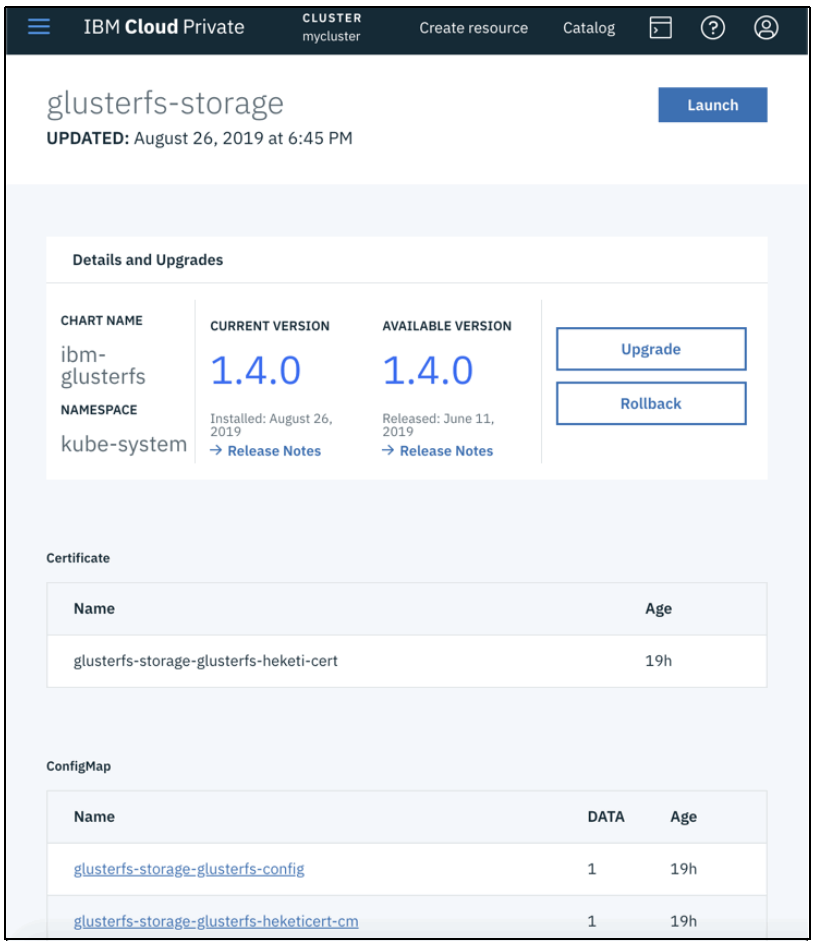


Figure 2-25 Verification of Gluster Storage

To check the glusterfs storage after deploying it, run the following command to check your pods to make sure that they are proceeding:

```
kubectl get pods -n kube-system -l app=glusterfs
```

Then follow the logs of the heketi pod by running this command:

```
kubectl logs $(kubectl get pods -n kube-system -l app=glusterfs,glusterfs=heketi-pod -o jsonpath='{.items[0].metadata.name}') -n kube-system --all-containers -f | grep -v "200" | grep -v "GET" | grep -v "TLS handshake"
```

Sample output of these commands is shown in Example 2-8. The figure shows the GlusterFS nodes and volumes that were added and also the topology verification. If you see errors in the logs (after filtering out TLS and the initial failed-to-connect error that occurs as the system wait for readiness of components), refer to the chart README (https://github.com/IBM/charts/tree/master/stable/ibm-glusterfs) for more information.

Note: If you redeploy GlusterFS with the same storage nodes, do not fail to clean up the old storage. First, follow the instructions here: https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/manage_cluster/gluster_cleanup.html.

Example 2-8 kubectl get pods

```
kubectl get pods -n kube-system -l app=glusterfs
NAME                                                    READY   STATUS      RESTARTS   AGE
glusterfs-storage-glusterfs-daemonset-rs5dl             1/1     Running     0          18h
glusterfs-storage-glusterfs-heketi-deployment-66595f6749-9m6x9   1/1     Running     0          18h
glusterfs-storage-glusterfs-heketicert-job-qnv4k        0/1     Completed   0          18h

kubectl logs $(kubectl get pods -n kube-system -l app=glusterfs,glusterfs=heketi-pod -o jsonpath='{.items[0].metadata.name}') -n kube-system --all-containers -f | grep -v "200" | grep -v "GET" | grep -v "TLS handshake"
is_gluster_daemon_up: Checking GlusterFS Daemonset's Readiness
is_gluster_daemon_up: GlusterFS Daemonset is not ready yet!
...
heketi-topology-load: Before loading topology. Topology Info: START |  | END
heketi-topology-load: Loading heketi topology
[negroni] Started POST /clusters
[heketi] INFO 2019/08/27 01:46:58 Backup successful
[negroni] Completed 201 Created in 58.898981ms
[negroni] Started POST /nodes
...
[heketi] INFO 2019/08/27 01:46:58 Added node f2bcd11ff21a3627bfaba35893b29159
...
[heketi] INFO 2019/08/27 01:46:58 Adding device /dev/disk/by-path/ccw-0.0.0500-part1 to node
f2bcd11ff21a3627bfaba35893b29159
...
[heketi] INFO 2019/08/27 02:23:24 Added device /dev/disk/by-path/ccw-0.0.0500-part1
...
heketi-topology-load: After loading topology. Topology Info: START |  | END
heketi-topology-load: Successfully loaded heketi topology
heketi-topology-verify: Start topology verification check
....
Hello from Heketi
...
heketi-topology-verify: Topology Info: START |
Cluster Id: 1ae80246445709692cec30bfc53a287c
```

```
    File:  true
    Block: true

    Volumes:

    Nodes:

   Node Id: f2bcd11ff21a3627bfaba35893b29159
   State: online
   Cluster Id: 1ae80246445709692cec30bfc53a287c
   Zone: 1
   Management Hostnames: 9.56.28.25
   Storage Hostnames: 9.56.28.25
   Devices:
      Id:6bc01ff90225b1d14d950adcfecb13bb   Name:/dev/disk/by-path/ccw-0.0.0501-part1State:online    Size
(GiB):41      Used (GiB):0       Free (GiB):41
         Bricks:
      Id:b9f0154a610a0ff54e06a084e1301b7f   Name:/dev/disk/by-path/ccw-0.0.0500-part1State:online    Size
(GiB):41      Used (GiB):0       Free (GiB):41
         Bricks: | END
heketi-topology-verify: Storage node count: start | 1 | End
heketi-topology-verify: topology verification complete
..
--log-level $LOG_LEVEL $GLUSTERD_OPTIONS (code=exited, status=0/SUCCESS)
```

After the heketi deployment is available, as shown in **bold** in Example 2-9, you know that the topology verification has passed, and everything you specified has been set up correctly.

Example 2-9 Topology verification has passed

```
kubectl -n kube-system get deployment -l app=glusterfs
NAME                                        READY   UP-TO-DATE  AVAILABLE   AGE
glusterfs-storage-glusterfs-heketi-deployment 1/1     1           1           19h
```

You can use this storage in Kubernetes for `PersistentVolumeClaims` by providing the StorageClass `glusterfs`. This storage class should now be in your storage classes for the cluster as shown in Example 2-10:

Example 2-10 kubectl get sc

```
kubectl get sc
NAME                       PROVISIONER                    AGE
glusterfs                  kubernetes.io/glusterfs        18h
image-manager-storage      kubernetes.io/no-provisioner   63d
logging-storage-datanode   kubernetes.io/no-provisioner   63d
mongodb-storage            kubernetes.io/no-provisioner   63d
```

Note: While your pods are in the “Running” state, allow time for GlusterFS. The deployment of GlusterFS can take up to 2 hours after the helm Chart release. You must allow time for the storage devices to be set up.

Example 2-11 shows the typical output that you see when everything is up and running.

Example 2-11 Everything is up and running

```
kubectl get all -l app=glusterfs -n kube-system
NAME                                                              READY   STATUS      RESTARTS   AGE
pod/glusterfs-storage-glusterfs-daemonset-rs5dl                   1/1     Running     0          78m
pod/glusterfs-storage-glusterfs-heketi-deployment-66595f6749-9m6x9   1/1     Running     0          78m
pod/glusterfs-storage-glusterfs-heketicert-job-qnv4k              0/1     Completed   0          78m

NAME                                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/glusterfs-storage-glusterfs-heketi-service   ClusterIP   10.0.22.69   <none>        8080/TCP   78m

NAME                                                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR           AGE
daemonset.apps/glusterfs-storage-glusterfs-daemonset   1         1         1       1            1
storagenode=glusterfs   78m

NAME                                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/glusterfs-storage-glusterfs-heketi-deployment   1/1     1            1           78m

NAME                                                              DESIRED   CURRENT   READY   AGE
replicaset.apps/glusterfs-storage-glusterfs-heketi-deployment-66595f6749   1         1         1       78m
replicaset.apps/glusterfs-storage-glusterfs-heketi-deployment-9578984c4    0         0         0       78m

NAME                                                COMPLETIONS   DURATION   AGE
job.batch/glusterfs-storage-glusterfs-heketicert-job   1/1           39s        78m
```

2.4 Sizing

Sizing is important when you plan for an installation of the IBM Blockchain Platform for Multicloud on IBM Cloud Private (ICP). Each component (certificate authority, orderer, and peer) of the IBM Blockchain Platform for Multicloud needs a virtual core processor unit (vCPU), memory (RAM), and storage to work. Each of these components requires various resource allocations, depending on their purpose within a blockchain network.

The default resource allocations for each node are shown in Table 2-11 on page 56.

Table 2-11 Default resource allocations

Component (all containers)	vCPU	Memory (GB)	Storage (GB)
Peer	1.7	3.0	200
CA	0.1	0.2	20
Ordering Service	0.45	0.9	100
Total	**2.25**	**4.1**	**320**

In this case, a vCPU unit is 1 vCPU as seen by Kubernetes. There are generally 4 vCPUs for each Integrated Facility for Linux (IFL). You can also have 2 vCPUs to 1 IFL. The amount of virtualization you apply to an IFL is depends on the environment you are trying to create (minimal, pilot, or production), use case, and target for transactions per second. The Memory in Table 2-11 on page 56 indicates the amount of RAM that is used by the component on the worker node where the developer deploys it. The Storage indicates the amount of backing disk space to reserve for the instance in production. As a blockchain grows over time, it

expands. That is why the default is 100GB for both the peer database and the state database (couchdb).

Note: You might see Virtual Processing Core (VPC) and vCPU used interchangeably. VPC is the metric that is used for pricing.

- If a software hypervisor is used (such as KVM, z/VM, and so on), a VPC is a virtual core that is assigned to a virtual machine (VM) in which ICP can run.
- If no software hypervisor is used (in other words, no z/VM and no KVM), then a VPC is a physical core (IFL). You must license each VPC that is made available to ICP. Additionally, whereas a VPC is a licensing term, a vCPU is the technical implementation. The number of vCPUs defines the capacity that is made available for use by ICP.

2.4.1 IBM Blockchain Platform console

The IBM Blockchain for Multicloud solution offers a nice user interface (UI), known as a console, for all administrators to deploy and create their blockchain network. To deploy a blockchain console, small amounts of resources are needed. Throughout all the network environments, there is a requirement for a console for each one, as explained in the following sections. Table 2-12 shows the resources that are required for deployment the IBM Blockchain Platform for Multicloud console:

Note: Though this UI is for the IBM Blockchain Platform for Multicloud, it is the same UI across all of the deployment options. The UI has the same look, feel, and functionality in the IBM Cloud, on-premises, and various cloud vendors that are supported.

Table 2-12 Resources required to deploy the IBM Blockchain Platform for Multicloud console

	vCPU	**Memory**
Console	1.225	2.45

Table 2-13 on page 57 shows the container breakdown of the IBM Blockchain Platform for Multicloud console:

Table 2-13 Container breakdown of the IBM Blockchain Platform for Multicloud console

	vCPU	**Memory**
Optools	0.5	1.0
Configtxlator	0.025	0.05
CouchDB	0.5	1.0
Operator	0.1	0.2
Deployer	0.1	0.2
Total	**1.225**	**2.45**

Note: The numbers that are calculated above are the requests. The limits can be the same for the requests that are shown above. You have the option to make the limits twice as much as the requests, but that isn't necessarily required. For more information on requests and limits, refer to 2.4.6, "Resource reallocation" on page 65.

2.4.2 Minimum network

A minimum network to test out the offering would consist of 1 peer, 1 ordering service node, and 2 Certificate Authorities (CAs). That means 1 for the peer organization and 1 for the ordering service. Such a network enables a developer to deploy chaincode, run it, and see the blockchain operations for administration of the blockchain that are available in the console. Figure 2-26 shows the topology of such a network.

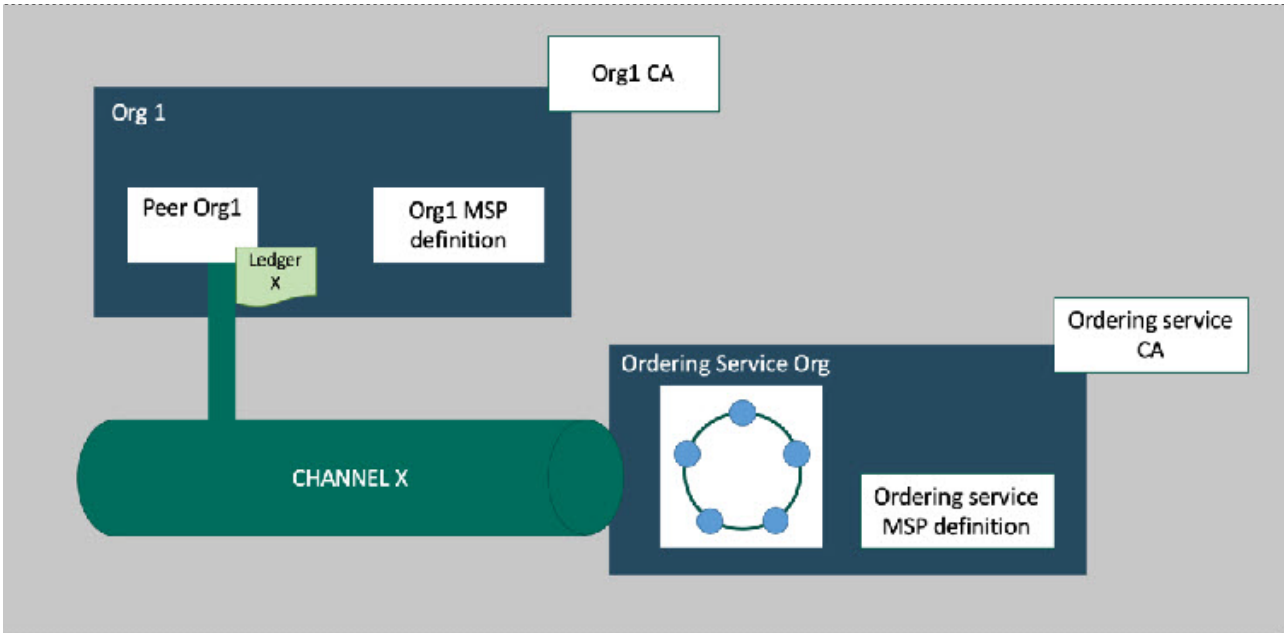


Figure 2-26 The minimum network

Table 2-14 shows the resources for minimum network components.

Table 2-14 Resources for minimum network components

Number of Component	Component (all containers)	Total vCPU	Total Memory (GB)	Storage (GB)
1	Peer	(1) x (1.7) = **1.7**	(1) x (3.0) = **3.0**	(1) x(200) = **200**
2	CA	*(2) x (0.1)* = **0.2**	*(2) x (0.2)* = **0.4**	*(2) x (20)* = **40**
1	Ordering node	*(1) x (0.45)* = **0.45**	*(1) x (0.9)* = **0.9**	*(1) x (100)* = **100**
4	**Total**	**2.35**	**4.3**	**340**

To break it down a little further, here are the exact resource allocation by each component and the container within each component.

Table 2-15 shows the container breakdown of the CA:

Table 2-15 : Container breakdown

Container	Total vCPU	Total Memory (GB)	Storage (GB)
CA	*(2) x (0.1)* = **0.2**	*(2) x (0.2)* = **0.4**	*(2) x(20)* = **40**
Total	**0.2**	**0.4**	**40**

Table 2-16 shows the container breakdown of the ordering service:

Table 2-16 Container breakdown of the ordering service

Container	Total vCPU	Total Memory (GB)	Storage (GB)
Orderer	*(1) x (0.75)* = **0.75**	*(1) x (0.75)* = **0.75**	*(1) x (100)* = ***100***
gRPC Proxy	*(1) x (0.1)* = **0.5**	*(1) x (0.2)* = **1**	***N/A***
Total	**4.25**	**4.75**	**100**

Table 2-17 shows the container breakdown of the peer:

Table 2-17 Container breakdown of the peer

Container	Total vCPU	Total Memory (GB)	Storage (GB)
Peer	*(1) x (0.2)* = **0.2**	*(1) x (1)* = **1**	*(1) x (100)* = **100**
CouchDB	*(1) x (0.2)* = **0.2**	*(1) x (0.4)* = **0.4**	*(1) x (100)* = **100**
Smart Contract	*(1) x (1)* = **1**	*(1) x (1)* = **1**	***N/A***
gRPC Proxy	*(1) x (0.2)* = **0.2**	*(1) x (0.4)* = **0.4**	***N/A***
Log Collector	*(1) x (0.1)* = **0.1**	*(1) x (0.2)* = **0.2**	***N/A***
Total	**1.7**	**3**	**200**

2.4.3 Pilot network

In the pilot network, you experiment with a blockchain network that simulates a production environment. To achieve this goal, the sizing metrics for the pilot network are made to closely match the sizings for the production network. Also, the pilot network might connect to existing systems of record, such as existing transaction systems or databases. Figure 2-27 shows the topology of such a network.

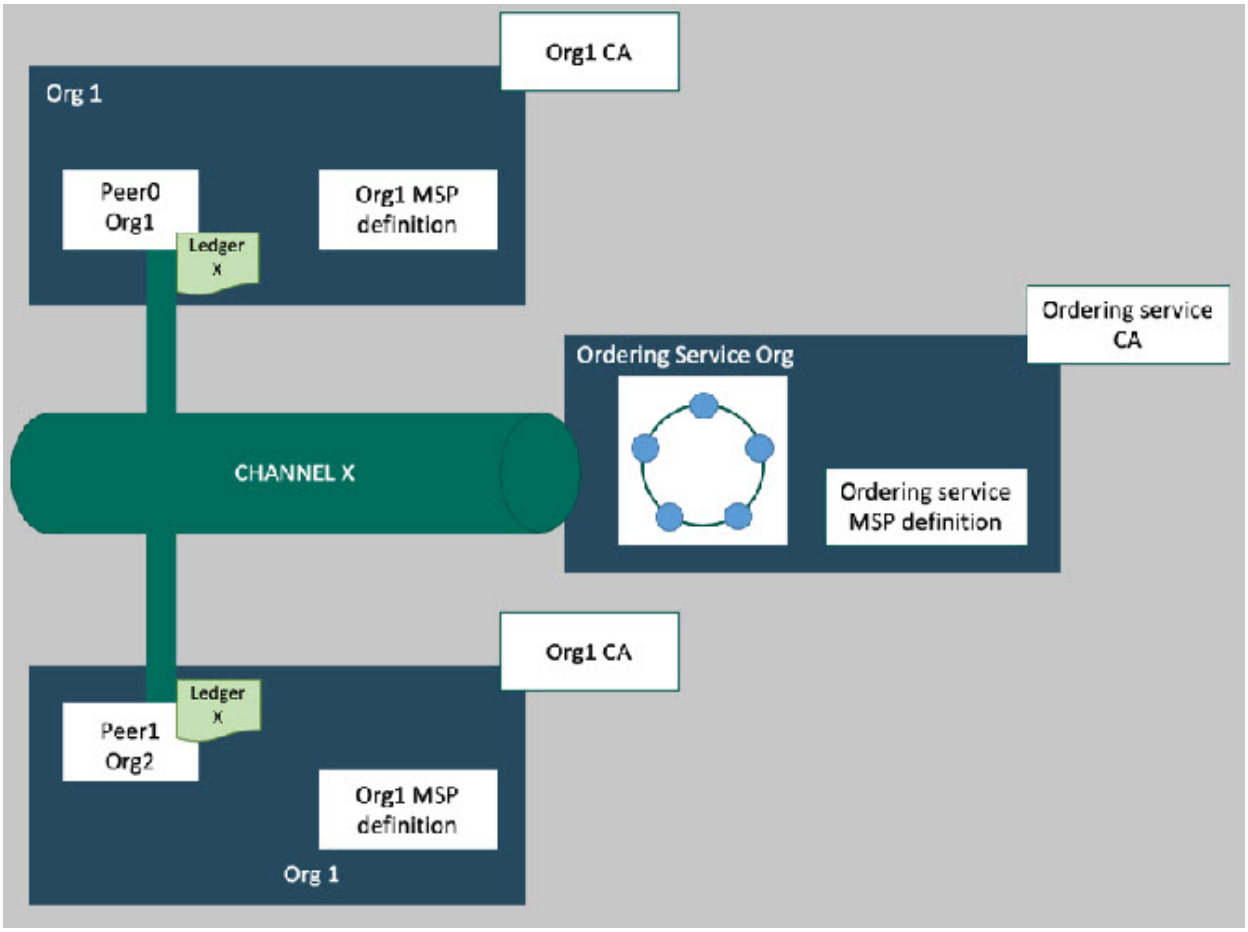


Figure 2-27 The pilot network

Based on Figure 2-27, we are deploying 10 total components; two peers to have a highly available solution, three CAs (one for our ordering service, one for the first peer, and the last CA for the second peer), and then five Raft nodes for our ordering service.

Table 2-18 is a chart of all of our components and their resources for a pilot network:

Table 2-18 .Components and their resources for a pilot network

Number of Component	Component	Total vCPU	Total Memory (GB)	Storage (GB)
2	Peer	*(2) x (4.2)* = **8.4**	*(2) x (5.4)* = **10.8**	*(2) x (200)* **400**
3	CA	*(3) x (0.1)* = **0.3**	*(3) x (0.2)* = **0.6**	*(3) x (20)* = **60**
5	Ordering node	*(5) x (0.85)* = **4.25**	*(5) x (0.95)* = **4.75**	*(5) x (100)* = **500**
10	**Total**	**12.95**	**16.15**	**960**

To break it down a little further, here are the exact resource allocation by each component and the container within each component.

Table 2-19 on page 61 is the container breakdown of the CA:

Table 2-19 : Container breakdown of the CA

Container	Total vCPU	Total Memory (GB)	Storage (GB)
CA	*(3) x (0.1)* = **0.3**	*(3) x (0.2)* = **0.6**	*(3) x(20)* = **60**
Total	**0.3**	**0.6**	**60**

Table 2-20 is the container breakdown of the ordering service:

Table 2-20 Container breakdown of the ordering service

Container	Total vCPU	Total Memory (GB)	Storage (GB)
Orderer	*(5) x (0.75)* = **3.75**	*(5) x (0.75)* = **3.75**	*(5) x (100)* = ***500***
gRPC Proxy	*(5) x (0.1)* = **0.5**	*(5) x (0.2)* = **1**	***N/A***
Total	**4.25**	**4.75**	**500**

Table 2-21 is the container breakdown for the peers:

Table 2-21 Container breakdown for the peers

Container	Total vCPU	Total Memory (GB)	Storage (GB)
Peer	*(2) x (2)* = **4**	*(2) x (1)* = **2**	*(2) x (100)* = **200**
CouchDB	*(2) x (1)* = **2**	*(2) x (1)* = **2**	*(2) x (100)* = **200**
Smart Contract	*(2) x (1)* = **2**	*(2) x (3)* = **6**	***N/A***
gRPC Proxy	*(2) x (0.1)* = **0.2**	*(2) x (0.2)* = **0.4**	***N/A***
Log Collector	*(2) x (0.1)* = **0.2**	*(2) x (0.2)* = **0.4**	***N/A***
Total	**8.4**	**10.8**	**400**

2.4.4 Production network

A production network must be in operation and available to end users and clients, either by mobile applications or web interfaces. After the blockchain network is in production, very limited downtime can be accepted, and the smart contracts have been finalized between network participants. Ideally, much of the final testing of smart contracts, network participants, and integration of existing transaction systems are done in minimal or pilot network environment. Additionally, production networks include a highly available (HA) and disaster recovery (DR) plan. Figure 2-28, “The production network” on page 62 shows the topology of such a network.

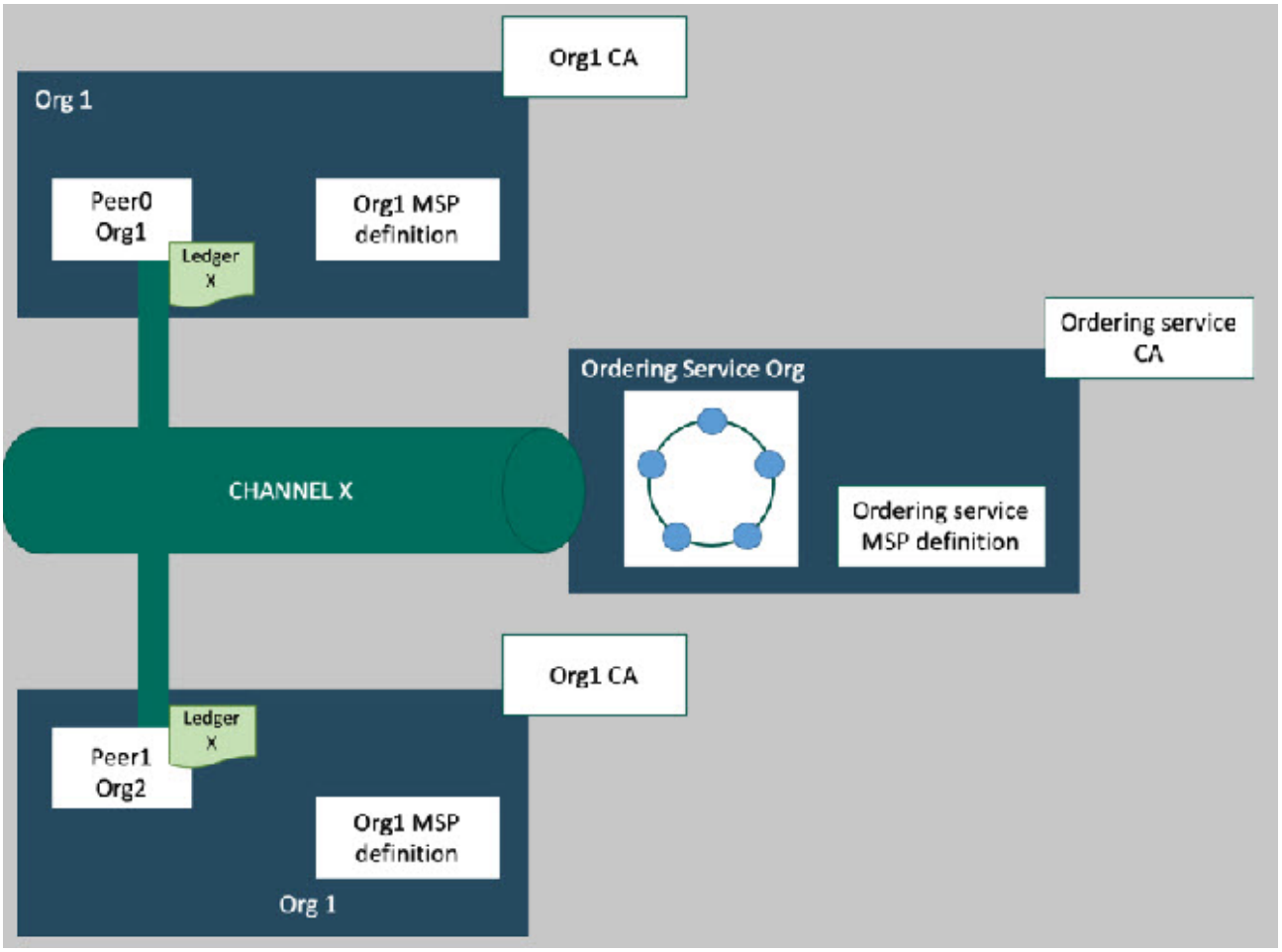


Figure 2-28 The production network

Just like the pilot network, there are a total 10 components; two peers to have a highly available solution, three CAs (one for our ordering service, one for the first peer, and another CA for the second peer), and five Raft nodes for the ordering service.

It is important to note, that this network is for one organization. Currently, each additional organization would have to create their certificate authority for the peer and then the peer itself. One organization can deploy the ordering service (five Raft ordering nodes) and that deployment is sufficient for the entire network. Currently, there is no HA capability for the ordering service, but Raft is a crash fault tolerant (CFT) ordering service. Being CFT means that if two of the five ordering nodes stop working then the ordering service can still process transactions into blocks, while the failing ordering nodes are coming back.

Table 2-22 is the recommended sizing and allocation per component for a production blockchain network:

Table 2-22 Recommended sizing and allocation per component for a production blockchain network

Number of Component	Component	Total vCPU	Total Memory (GB)	Storage (GB)
2	Peer	*(2) x (6.2)* = **12.4**	*(2) x (6.9)* = **13.8**	(2) x 200 = **400**
3	CA	*(3) x (0.1)* = **0.3**	*(3) x (0.2)* = **0.6**	*(3) x (20)* = **60**

Number of Component	Component	Total vCPU	Total Memory (GB)	Storage (GB)
5	Ordering node	*(5) x (5.1)* = **5.5**	*(5) x (1.2)* = **6**	*(5) x (100)* = **500**
10	**Total**	**18.2**	**18.4**	**960**

To break it down a little further, here are the exact resource allocations by each component and the container within each component.

Table 2-23 is the container breakdown of the CA:

Table 2-23 container breakdown of the CA

Container	Total vCPU	Total Memory (GB)	Storage (GB)
CA	*(3) x (0.1)* = **0.3**	*(3) x (0.2)* = **0.6**	*(3) x(20)* = **60**
Total	**0.3**	**0.6**	**60**

It is possible to integrate your external certificate authorities if they are in X.509 format. For more information on integrating your external certificate authority, go to https://cloud.ibm.com/docs/services/blockchain/howto?topic=blockchain-ibp-console-build-network#ibp-console-build-network-third-party-ca.

Note: The IBM Blockchain Platform for Multicloud doesn't support intermediate certificate authorities.

Table 2-24 shows the container breakdown of the ordering service:

Table 2-24 Container breakdown of the ordering service

Container	Total vCPU	Total Memory (GB)	Storage (GB)
Orderer	*(5) x (1)* = **5**	*(5) x (1)* = **5**	*(5) x (100)* = ***500***
gRPC Proxy	*(5) x (0.1)* = **0.5**	*(5) x (0.2)* = **1.0**	***N/A***
Total	**5.5**	**6**	**500**

Note: Choose five ordering nodes (Raft) to create a CFT ordering service. These five ordering nodes are all apart of the Raft protocol, which is apart of etcd. Etcd is a distributed open source key-value store.

Table 2-25 shows the container breakdown for the peers:

Table 2-25 Container breakdown for the peers

Container	Total vCPU	Total Memory (GB)	Storage (GB)
Peer	*(2) x (3)* = **6**	*(2) x (1.25)* = **2.5**	*(2) x (100)* = **200**
CouchDB	*(2) x (1.5)* = **3**	*(2) x (1.25)* = **2.5**	*(2) x (100)* = **200**
Smart Contract	*(2) x (1.5)* = **3**	*(2) x (4)* = **8**	***N/A***
gRPC Proxy	*(2) x (0.1)* = **0.2**	*(2) x (0.2)* = **0.4**	***N/A***

Container	Total vCPU	Total Memory (GB)	Storage (GB)
Log Collector	*(2) x (0.1)* = **0.2**	*(2) x (0.2)* = **0.4**	***N/A***
Total	**12.4**	**13.8**	**400**

Note: Currently IBM Blockchain Platform for Multicloud supports coudchdb only, and not leveldb.

Proper sizing of all the components of the blockchain network is important. But sizing of the peer is the most important. Typically, the more resources that are allocated to the peer, the more transactions per second the network can achieve.

Note: The smart contract container's memory has a higher memory allocation. This is required because when transactions come through — by either an application or web interface — the execution of the transaction occurs in the smart contract container. The more memory that you allocate for that container, the faster that your network typically runs.

2.4.5 Component containers

In each of the blockchain networks, there is the specific component, but there are containers within each component. In this section, we explain the containers in each of the components and what they are responsible for.

Table 2-26 shows the component breakdown of the CA's container:

Table 2-26 Component breakdown of the CA's container

Container	Description
CA	This container is responsible for registering and enrolling nodes and users. Nodes analogous to a peer or ordering service. Users are analogous to an end user who might interact with the blockchain network through a mobile app or web interface. Additionally, the certificate authority saves a copy of every certificate it creates.

Table 2-27 shows the component breakdown of the ordering service's containers:

Table 2-27 Component breakdown of the ordering service's containers

Container	Description
Orderer	This container is responsible for ordering transactions and processing configuration updates, such as channel updates. Additionally, this container contains a copy of the blockchain ledger for each of the channels the ordering service it is a part of. The ordering service is a Raft ordering node, whether that is 1 ordering node or 5 ordering nodes.
gRPC	This container is contains the processes that allow the console (user interface) to communicate with this node (peer). It connects the IBM Cloud Private Instance to the IBM Blockchain for Multicloud instance.

Table 2-28 shows the component breakdown of the peer's containers:

Table 2-28 Component breakdown of the peer's containers

Container	Description
Peer	This container holds the smart contracts that are installed and instantiated on it. Also, this containers holds the ledger for all the channels that the peer is joined to.
CouchDB	This container is the world state database for each channel that the peer is joined to. The world state contains the current value for all the assets recorded on the blockchain ledger. An analogy would be a potato that is an asset in the blockchain network. Imagine that the potato is passed around 100 times. The world state database would contain the information of who has the potato on the 100th transfer. It is the most up-to-date information in regard to a specific asset. By default, the world state database is CouchDB.
Smart Contract	This container is actually a docker-in-docker (dind) container. The smart contracts are stored here, when there is an installation and instantiation of a smart contract.
Log Collector	This container is responsible for the log messages that are coming from the smart contract container (dind).
gRPC	This container contains the processes that allow the console (user interface) to communicate with this node (peer). It connects the IBM Cloud Private Instance to the IBM Blockchain for Multicloud instance.

2.4.6 Resource reallocation

When you are creating each component, you give an initial resource allocation for that component. This allocation includes giving the component a certain amount of vCPU and memory. By default, you cannot change the storage on some of the containers. The defaults in the resource allocation are sufficient to run a successful blockchain network. Figure 2-29 on page 65 is a picture of the sample default resource allocation for a CA:

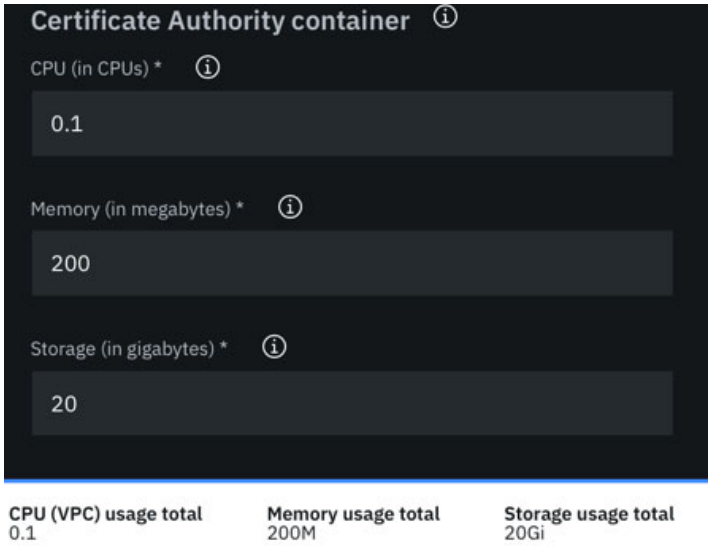


Figure 2-29 sample default resource allocation for a CA:

However, consider a scenario where you use Grafana (https://grafana.com/) to monitor your components and their resources. When you check Grafana, you notice that a certain component could use more or less resources than what you originally allocated. If you want to

reallocate resources, you can do that easily through the blockchain user interface. Figure 2-30 through Figure 2-32 on page 67 show how to actually do this:

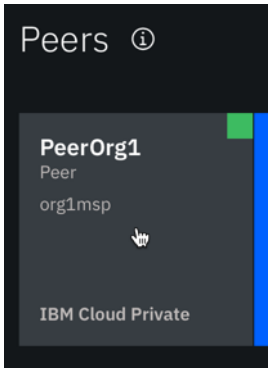


Figure 2-30 Reallocating resources for a CA -1

Figure 2-31 on page 66

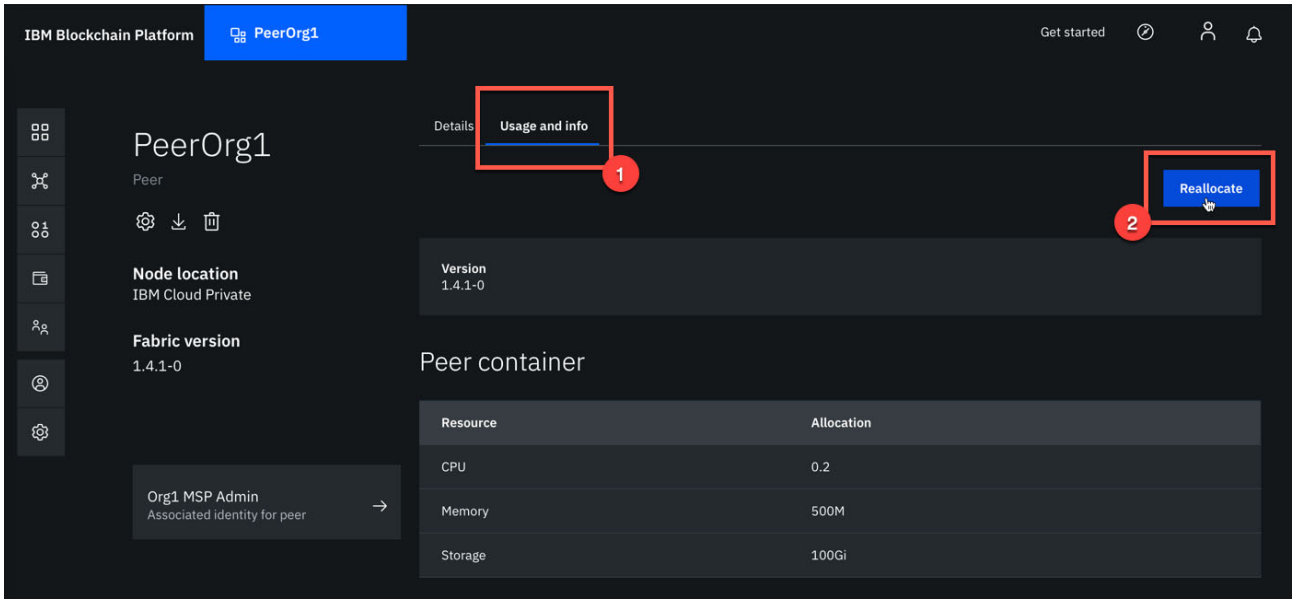


Figure 2-31 Reallocating resources for a CA -2

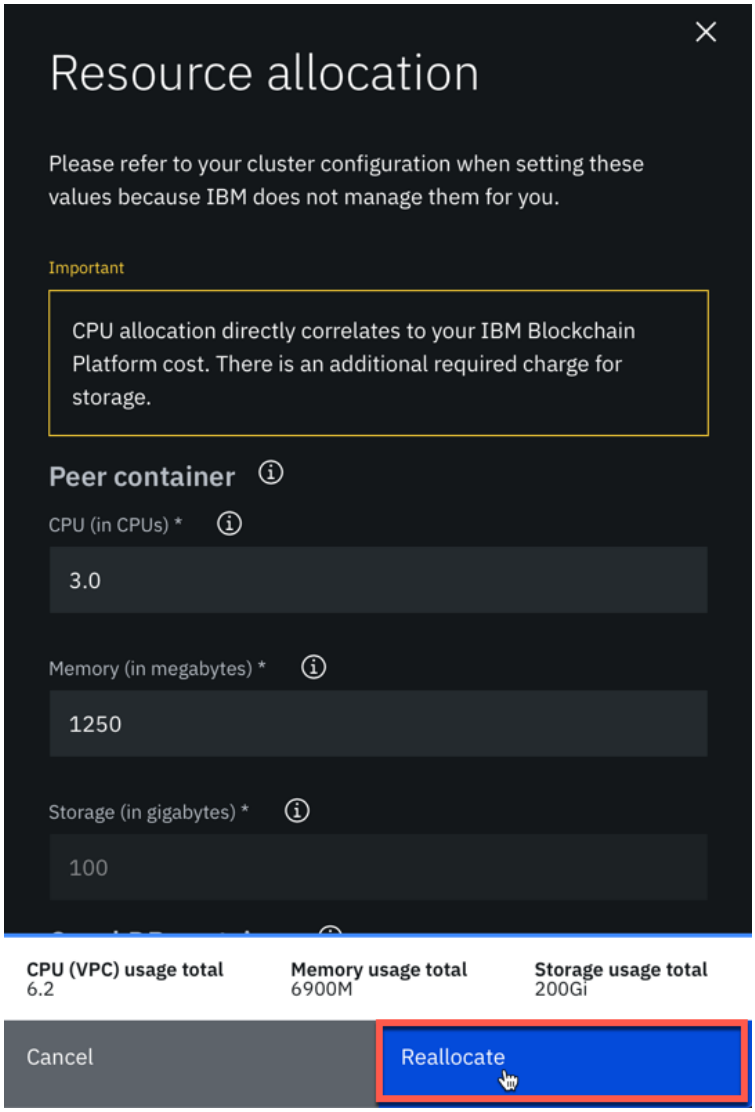


Figure 2-32 Reallocating resources for a CA -3

Note: After you reallocate the component, the Kubernetes pod of that component restarts. Depending on the environment in which you are reallocating the component, this can be a significant event.

Imagine that you want to reallocate the orderer in the production environment. Because there are multiple Raft ordering nodes running at the same time, we can maintain the network without any downtime. We reallocate 1 ordering node at a time, and wait for the orderer pod to say *available*. This means that the new orderer node is up and working and can process transactions as they come through.

A minimal network environment performs differently. You run only 1 Raft ordering node in such an environment. When you want to reallocate the orderer, the network must be down temporarily.

When you define the resource allocation for the components, there is a concept of requests and limits in Kubernetes. When you allocate resources in the blockchain user interface, this

setting applies to both the request and limit. This means that if you were to create a CA with 0.1 vCPU (equivalent to 100m vCPU), it would be set as both the request and limit.

The following scenario reveals a benefit of knowing this concept of requests and limits:

- You request a certain amount of vCPU, but put a limit on how much vCPU the component can actually take up. For example, you might have a CA request of 0.1 vCPU and put a limit on of 0.5 vCPU.
- As a result, the CA take up 0.1 vCPU but can go up to 0.5 vCPU, if needed. *It is the upmost importance that you know your environment architecture before you configure the requests and limits*. Figure 2-33 shows the ellipsis icon (...) that you click when you want to edit the configuration of the requests and limits of your deployed components:

Figure 2-33 Editing the configuration of the requests and limits of your deployed components - 1

Click **Submit** when you have edited the deployment as shown in Figure 2-34.

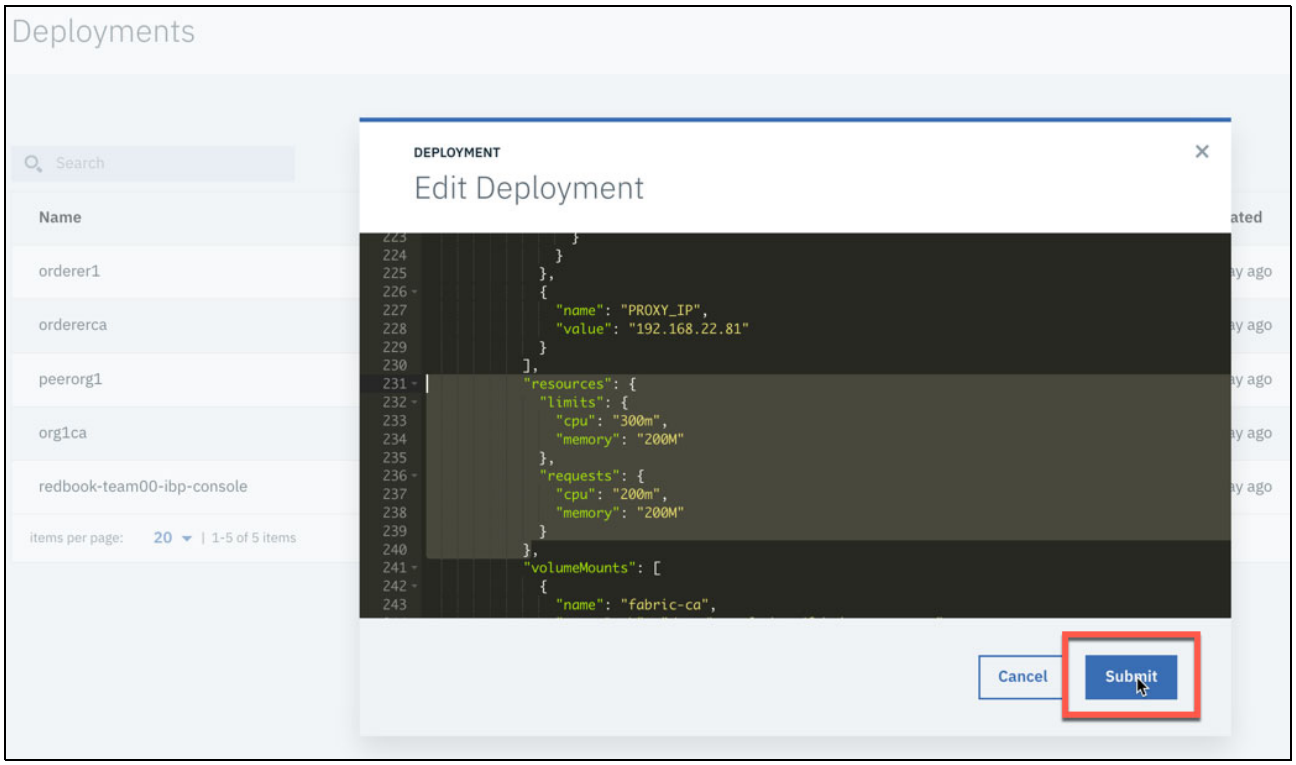


Figure 2-34 Editing the configuration of the requests and limits of your deployed components - 2

2.5 Considerations for specific use cases

Each installation comes with a unique set of challenges, depending on the infrastructure environment. Some environments might be able to open their system to the internet, and some environments might be completely closed to the internet. Also, a typical enterprise requires a highly available (HA) blockchain network and disaster recovery (DR) plan if the system goes down. For more information on such scenarios, see Chapter 5, “Specific scenarios” on page 197.

3

Secure Service Container installation and configuration

In this chapter, we describe the installation and configuration process of the SSC partition, including the components that are used for IBM Cloud Private (ICP).

This chapter includes the following topics:

3.1 Secure Service Container architecture

The Secure Service Container for IBM Cloud Private (ICP) requires a server (x86 or Linux on Z). The server acts as master node to manage worker and proxy nodes that are hosted on the Secure Service Container partition of a LinuxONE server. Figure 3-1 illustrates the SSC architecture.

The downloaded Secure Service Container image file should be copied to this x86 or Linux on Z server. Then, the installation of IBM Cloud Private can be started. During the installation, the worker or proxy nodes are provisioned in the Secure Service Container of LinuxONE servers.

After the IBM Cloud Private cluster is set up, you can deploy containerized IBM Middleware applications. You can also use common management tooling for deploying home-grown or third-party Docker- and Kubernetes-based applications.

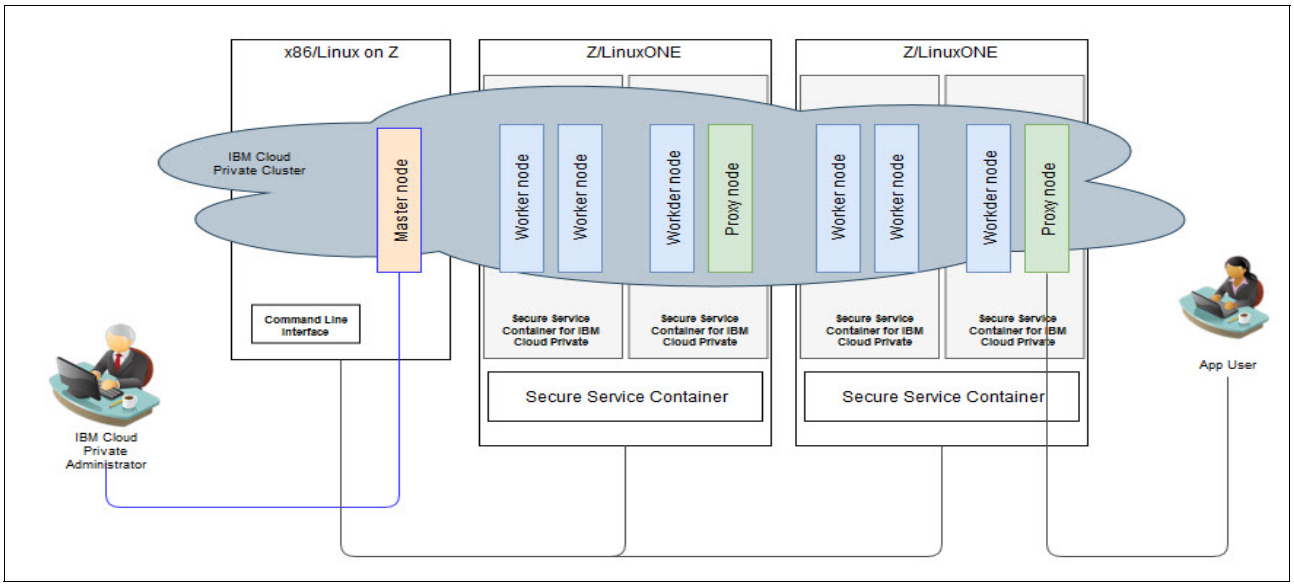


Figure 3-1 Secure Service Container architecture for ICP

3.2 An overview of SSC configuration and installation

This section describes the required steps and all requirements to install and configure an SSC into a LinuxONE Logical Partition (LPAR). You can use this information to deploy your blockchain environment.

SSC components and requirements are described in section 3.3, “Hardware and software requirements” on page 72.

3.2.1 SSC bootloader overview

The appliance contains the operating system and application. Figure 3-2 illustrates an overview of the process to initiate an SSC partition.

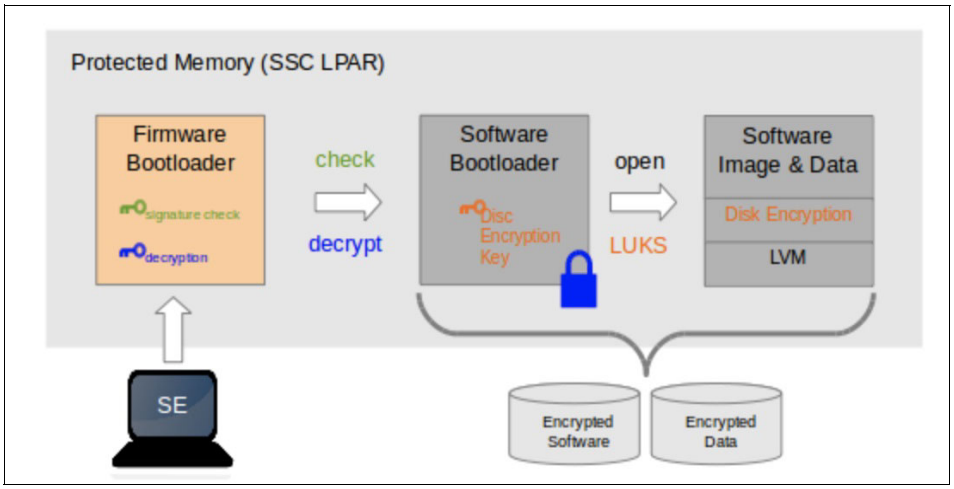


Figure 3-2 SSC bootloader overview diagram

1. Firmware bootloader is loaded in memory.
2. Firmware loads the software bootloader from disk.
   a. Checks integrity of software bootloader.
   b. Decrypts software bootloader.
3. Software bootloader activates encrypted disks.
   a. Key stored in software bootloader (encrypted).
   b. Encryption/decryption done on the fly when it accesses appliance code and data.
4. Appliance designed to be managed by remote APIs only.
   a. REST APIs to configure Linux and apps.
   b. No ssh (allowed only in dev mode).

3.2.2 Download the image

To install SSC blockchain appliance, download the official packages from the IBM Passport Advantage site `https://www.ibm.com/software/passportadvantage/pao_customer.html`. Follow steps below to get access to the SSC images:

1. Log in to IBM Passport Advantage by using your IBM ID and password. Contact your sales representative if you do not have one.
2. Go to My Programs, and then select the IBM Secure Service Container for IBM Cloud Private program.
3. Download the Secure Service Container for IBM Cloud Private image to your x86 or Linux on Z server.
4. Create an installation directory to store the Secure Service Container for IBM Cloud Private image and configuration files. For example:
   `mkdir /opt/<installation-directory>/cluster/images`
5. Change to the installation directory, and extract the compressed file on the x86 server.
   `cd /opt/<installation-directory>/cluster/images`
   `tar -zxvf CNYC7EN.tar.gz`
6. After decompressing the file, you should have the following files.
   a. **`secure-service-container-for-icp_x86.tar.gz`**, which supports master node on an x86 server.
   b. **`secure-service-container-for-icp_s390x.tar.gz`,** which supports master node on a Linux on Z server.

On your x86 or Linux on Z server, extract the chosen image to get the following compressed files:

a. **ssc4icp-cli-installer.docker-image.tar**
The Secure Service Container for IBM Cloud Private command line tool to create IBM Cloud Private nodes and configure the IBM Cloud Private cluster.
b. **secure-service-container-for-icp.appliance.<version_number>.img.gz**,
The Secure Service Container for IBM Cloud Private software appliance to be installed on the IBM Z or LinuxONE system.
c. **config/ICPIsolatedvm.tar.gz**
The isolated VM image for hosting IBM Cloud Private proxy and worker nodes.

3.3 Hardware and software requirements

This section presents the required software, hardware, and system configuration settings for setting up an IBM Cloud Private cluster on Secure Service Container.

3.3.1 Hardware requirements for the 64-bit x86 server or Linux on Z server

The x86 server or Linux on Z server is used to complete these tasks:

- Download Secure Service Container for IBM Cloud Private (SSC4ICP) and IBM Cloud Private (ICP) installation binary.
- Install the SSC4ICP and ICP.
- Configure the network for the ICP cluster.
- Also, act as the master and boot node in the ICP cluster.

Minimal requirements are as follows:

- 8 or more cores with at least 2.4 GHz
- 16 GB RAM
- 300 GB disk space

3.3.2 Hardware requirements for Secure Service Container partition

The Secure Service Container partitions are used for hosting worker and proxy nodes. You can find the hardware specifications for SSC partitions in Section 1.3.5, “IBM Secure Service Container” on page 10 of this document. The following list shows the minimal requirement for one Secure Service Container partition, which hosts one worker node and one proxy node.

Minimal (one worker + one proxy)

- 2 Integrated Facility (IFL)
- 1 Open System Adapter (OSA) card with two virtual devices (one for internal and one for external data traffic)
- 12 GB RAM
- 530 GB disk space, including
  - 50 GB for the Secure Service Container for IBM Cloud Private appliance
  - 200 GB in the data pool for each node on the Secure Service Container partition
  - 80 GB for each GlusterFS node on the Secure Service Container partition

Note: For each worker node that runs on the Secure Service Container partition, you must allocate as least 200 GB in the data pool (140 GB for the docker file system and 60 GB for the root file system). See *Hardware requirements and recommendations of IBM Cloud Private* for more details:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/supported_system_config/hardware_reqs.html

For each Secure Service Container partition, you can use either SCSI or extended count key data (IBM ECKD) disks as the storage subsystem. You can allocate 50 GB for the appliance, and at least 200 GB distributed over one or more disks for the data pool.

Secure Service Container for IBM Cloud Private supports hostPath persistent volumes as the storage solution. For more information about the hostPath volume, see hostPath:
https://kubernetes.io/docs/concepts/storage/#hostpath

Secure Service Container for IBM Cloud Private also supports GlusterFS as the persistent volumes. You need to allocate at least 80 GB for each GlusterFS node. For more information, see Deploying GlusterFS:
https://www.ibm.com/support/knowledgecenter/SSUPZ7_1.1.0.3/topics/deploy_glusterfs.html

3.3.3 Networking

To work properly, the Secure Service Container for IBM Cloud Private requires two levels of network:

- Network among cluster nodes by using the internal IP addresses
- Network for proxy nodes for external requests to the services inside the cluster

Table 3-1 shows the supported network interfaces on the Secure Service Container partitions.

Table 3-1 Supported network interfaces on the Secure Service Container partitions

Interface	Layer 2 network	Layer 3 network
Ethernet	Yes	Yes
VLAN	Yes	Yes
Bond	Yes	Yes

3.3.4 Supported operating systems and platforms

The operating system for running the worker and proxy nodes is Ubuntu 18.04, which is encapsulated into the Secure Service Container for IBM Cloud Private. This operating system is installed into the Secure Service Container partition as a docker image during the installation.

However, you must set up an x86 or Linux on Z server to host the master node. This node is configured with one of the supported operating systems in the following list:

- Red Hat Enterprise Linux (RHEL) 7.3, 7.4, 7.5, 7.6 and 7.7
- Ubuntu 16.04 LTS and 18.04 LTS
- SUSE Linux Enterprise Server (SLES) 12 SP4, and 15

Note: Linux Unified Key Setup (LUKS) hardware encryption on the x86 or Linux on Z server can protect the hardware from faulty access. When you install the Ubuntu onto the x86 or Linux on Z server, select the **Encrypt the new Ubuntu installation for Security** option to encrypt the hard disk.

3.3.5 Software requirements

You must invest in the following software infrastructure to run the Secure Service Container for IBM Cloud Private solution:

- IBM Cloud Private
- IBM Secure Service Container for IBM Cloud Private, which you can get from IBM Passport Advantage site. See 3.2.2, “Download the image” on page 71 for more details.
- Feature Code 0104 (Container Hosting Foundation), which is required by the IBM Secure Service Container. It can be ordered on the IBM Z14, IBM LinuxONE Emperor II, and IBM LinuxONE Rockerhopper II servers from the eConfig fulfillment system.

You can contact your sales representatives to obtain the required access to IBM Passport Advantage site (Information provided earlier in this chapter) and eConfig system.

3.3.6 Supported Docker versions

The Docker release that is required by Secure Service Container for IBM Cloud Private is identical to the requirements of IBM Cloud Private. See *IBM Cloud Private Supported Docker Versions* for more details:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/supported_system_config/supported_docker.html

You must install one of supported Docker versions on the x86 server.
Note: The supported Kubernetes version for the master node is 1.11. The Docker/Kubernetes environment on the Secure Service Container partition is configured during the installation of Secure Service Container for IBM Cloud Private.

3.3.7 Supported IBM Cloud Private versions

The Secure Service Container for IBM Cloud Private solution is tested and developed on the following IBM Cloud Private bundles. (Cloud Native is not available for these versions).

- 3.2.0
- 3.1.2
- 3.1.1

Note: To install the IBM Cloud Private Community edition, the x86 or Linux on Z server must have internet access to install the required docker images that are hosted on the external site.

The installation packages of IBM Cloud Private Enterprise Edition can be acquired from the IBM Passport Advantage site.

The code snippets or links to IBM Cloud Private documentation use Version 3.2.0 to maintain consistency. You can also use a different supported version number or refer to its documentation, as needed.

3.3.8 Required ports

The required ports of Secure Service Container for IBM Cloud Private are identical to the ones for IBM Cloud Private. For more information, see *Required ports* at https://www.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/supported_system_config/required_ports.html.

3.3.9 Defining the lab environment

To build the SSC environment to install ICP, use information and table sheets in 1.4, “Our lab environment” on page 12.

3.4 Deploying and configuring SSC for ICP in our lab environment

This section describes the steps to deploy an IBM Blockchain Platform into an SSC partition on a Linux on Z host system. Download the *IBM Z Secure Service Container User's Guide* at https://www-01.ibm.com/support/docview.wss?uid=isg26ab9a15cbe12a81d85258194006e4a89. Some of the instructions are available in this publication.

Deployment consists of the following tasks:

- Create Secure Service Container Partition
- Install the Secure Service Container for IBM Cloud Private appliance
- Install the Secure Service Container for IBM Cloud Private CLI tool
- Install IBM Cloud Private Cluster
  - Configure Secure Service Container storage
  - Configure the appliance network
  - Configure the network for worker and proxy nodes
  - Configure the cluster resources
  - Create the cluster nodes
  - Configure the network on the master node
- Deploy IBM Cloud Private
- Deploy containerized applications

3.4.1 Creating Secure Service Container partitions

Configure and start a Secure Service Container partition with the boot option **Secure Service Container Installer** selected. For additional instructions, see the following topics in *IBM Z Secure Service Container User's Guide* https://www-01.ibm.com/support/docview.wss?uid=isg26ab9a15cbe12a81d85258194006e4a89:

Refer to the following chapters for a standard mode system:

- *Chapter 3 - Configuring a Secure Service Container partition on a standard mode system*
- *Chapter 4 - Starting a Secure Service Container partition on a standard mode system*

Refer to the following chapters for a DPM-enabled system:

- *Chapter 8 - Creating a Secure Service Container partition on a DPM-enabled system*
- *Chapter 9 - Starting a Secure Service Container partition on a DPM-enabled system*

Note: Write down the following values that are specified in the image profile (standard mode system) or the partition definition (DPM-enabled system) for the Secure Service Container partition when you configure the Secure Service Container. You need them when you configure the appliance network and create cluster nodes.

- Master user ID
- Master password
- IP address

The following are the screen captures for the SSC partition that we created in our lab environment:

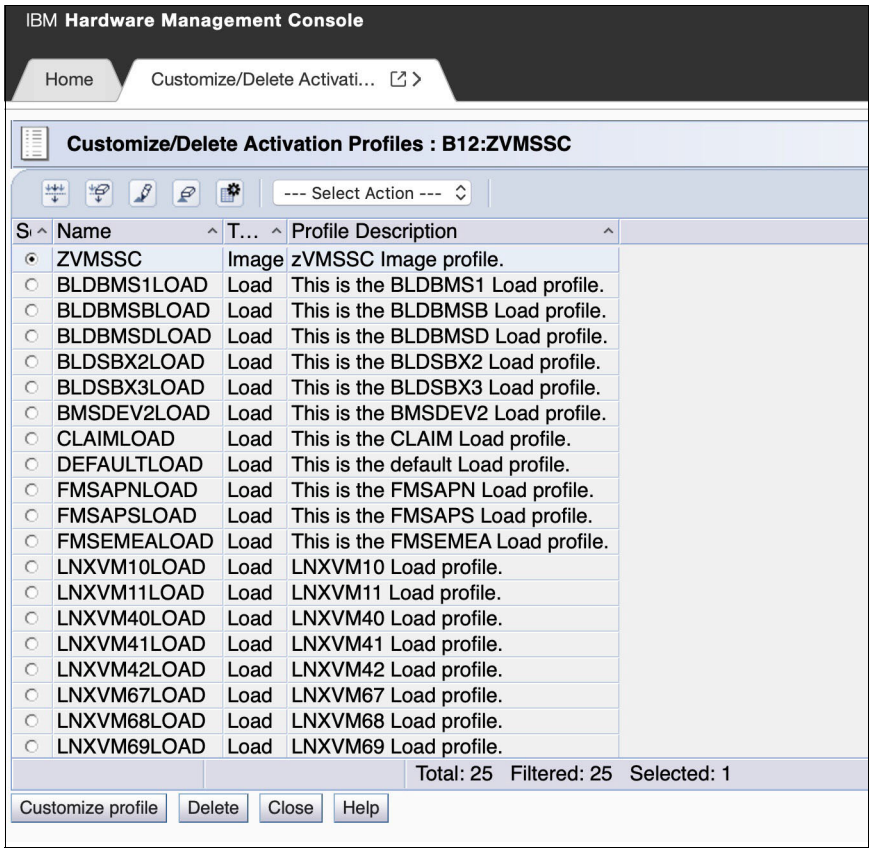


Figure 3-3 Customize/delete Image Profiles window

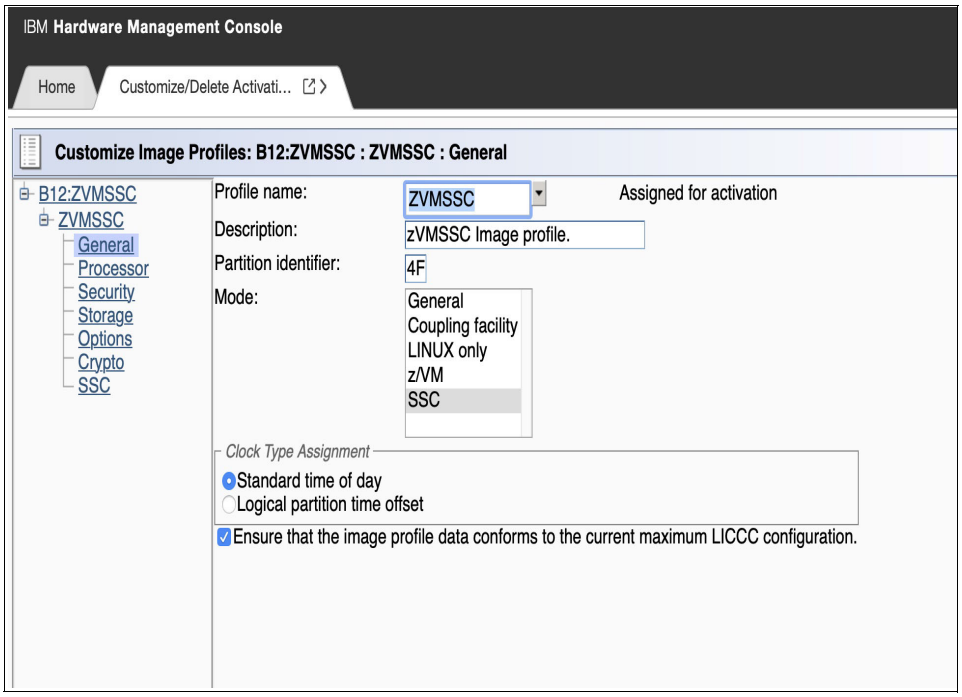


Figure 3-4 General SSC profile information

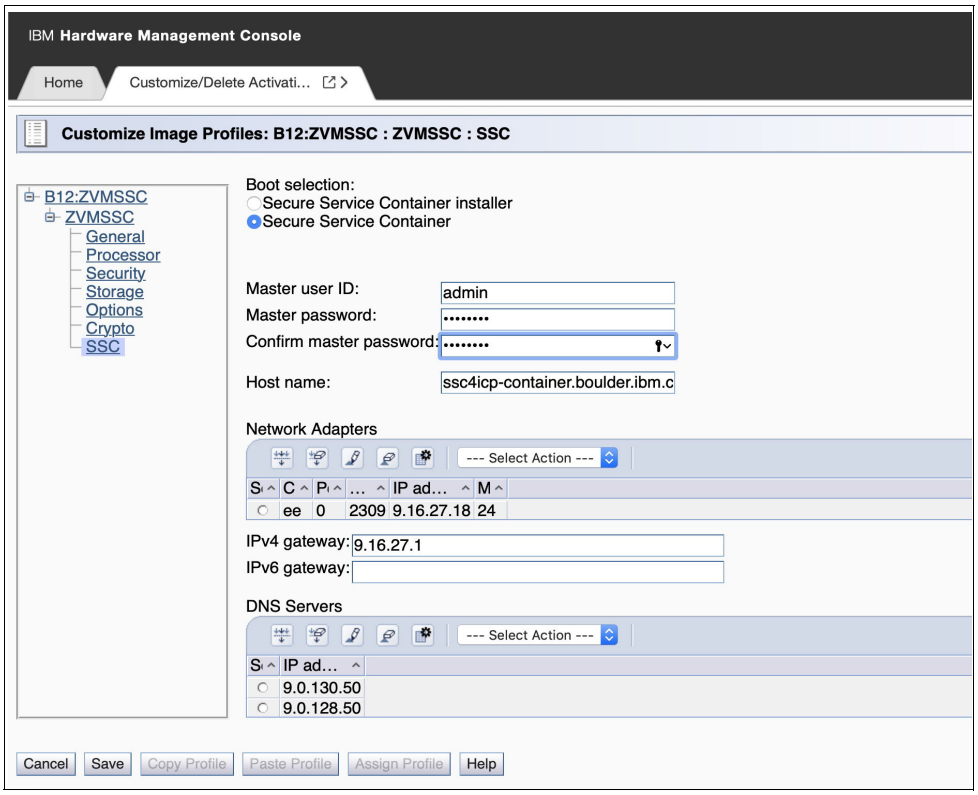


Figure 3-5 SSC profile login information

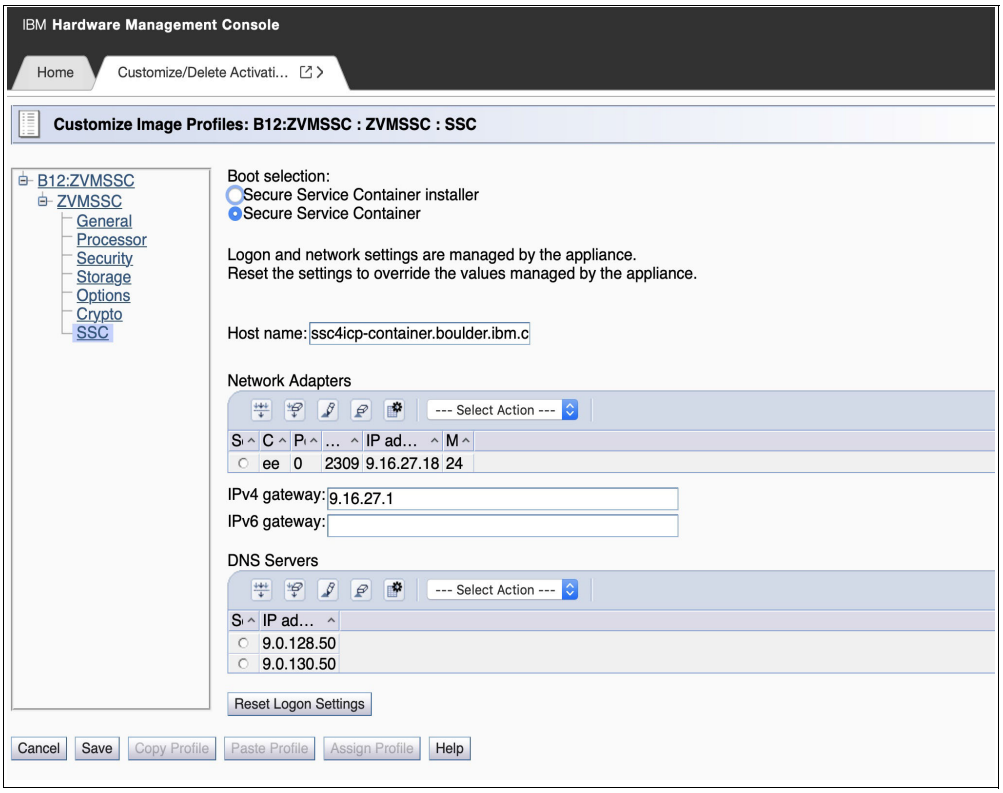


Figure 3-6 SSC profile network information

3.4.2 Installing the Secure Service Container for IBM Cloud Private appliance

The Secure Service Container for IBM Cloud Private is a type of software appliance. Therefore, you need to Install the appliance in the Secure Service Container partition as a new software appliance. For instructions, see the following topic in *IBM Z Secure Service Container User's Guide* at https://www-01.ibm.com/support/docview.wss?uid=isg26ab9a15cbe12a81d85258194006e4a89:

See *Chapter 13 - Installing a new software appliance in a Secure Service Container partition*

Note: You can specify only one disk (either DASD or FCP) during the appliance installation stage. Before you start the installation, ensure that you have the disks available.

For our lab environment, the installation disk is $0.0.313F$.

Follow these steps in Chapter 13 of *IBM Z Secure Service Container User's Guide* at https://www-01.ibm.com/support/docview.wss?uid=isg26ab9a15cbe12a81d85258194006e4a89.

1. Connect to the Secure Service Container installer (**`https://9.16.27.18`**) *through the Chrome or Firefox browsers.*
2. On the login page, enter the master user ID and password values that are defined in the image LPAR profile.
3. On the main page, click the plus (+) icon to install image files from local media. The page that is displayed changes to the Install Software Appliance page.

4. On the Install Software Appliance page, complete the fields as shown in Figure 3-7 on page 79.
   a. Select **Upload image** to target disk.
   b. Under Local Installation Image, click **Browse** and navigate to the location where you downloaded the software appliance on our local disk. Then, select the software appliance image and click **Open**. The Image Details section is populated with the information about the selected software appliance.
   c. Under Target Disk on Server, select **FICON DASD** as the device type. Then, click the down arrow in the **Disk field** to display a list of available disks on the server. Type **`0.0.3f3F`** in the text box, which is the disk address where the appliance image will be installed.
   d. Then, click **Apply** to upload the software appliance image to the target disk on the server.

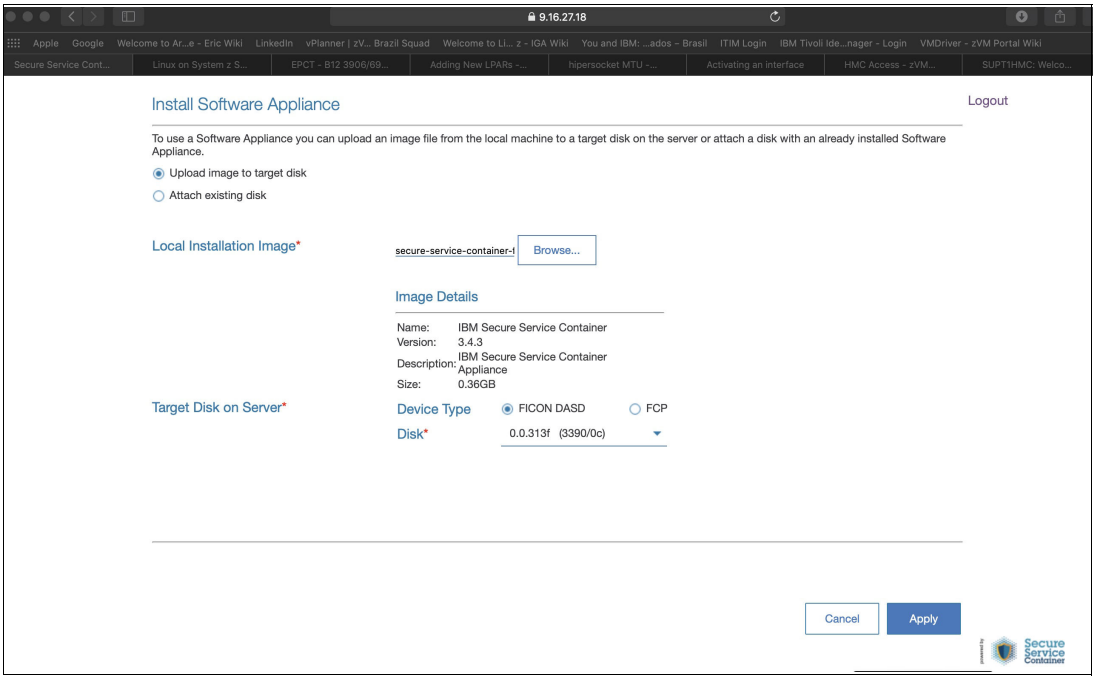


Figure 3-7 Uploading the software appliance image

A confirmation dialog box is displayed, as shown in Figure 3-8.

   e. In the confirmation dialog box, complete the following steps.
      i. Click **Reboot** to have the installer automatically reactivate the partition.
      ii. Click **Yes** to continue with the installation.

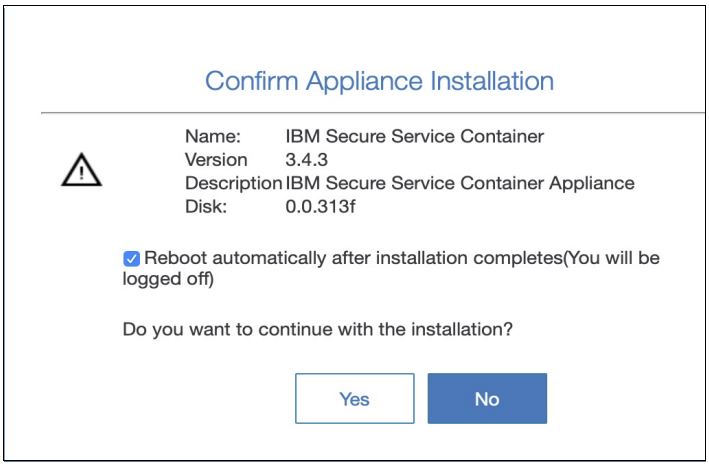


Figure 3-8 Confirmation dialog page

Figure 3-9 on page 80 shows a message that the image is being uploaded.

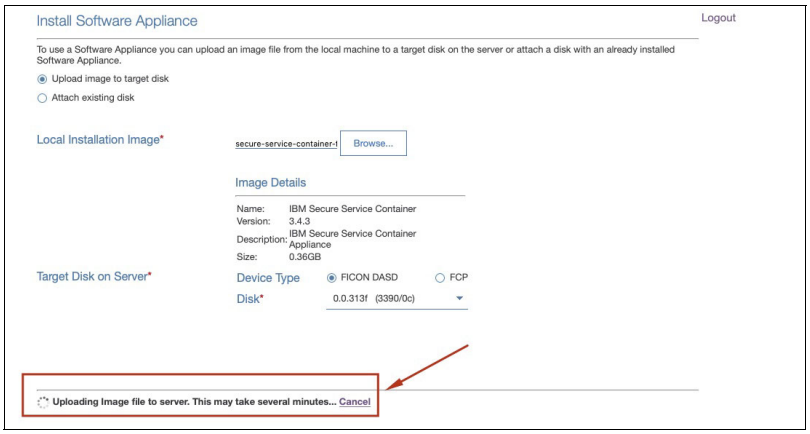


Figure 3-9 Uploading the image

f. The Secure Service Container installer uploaded the appliance image to the target disk, and rebooted the partition to load the appliance on the `0.0.313F` disk.
g. To confirm that appliance was installed, we accessed the appliance through a web browser.

3.4.3 Installing the Secure Service Container for IBM Cloud Private CLI tool

IBM Secure Service Container for IBM Cloud Private is a secure node infrastructure that is deployed on IBM Secure Service Container of LinuxONE. The product has a REST API layer for creating nodes/isolated virtual machines (VMs) that can become the cluster nodes for IBM Cloud Private after it is installed.

The Secure Service Container for IBM Cloud Private command line interface (CLI) tool automates the infrastructure setup by creating all the necessary cluster nodes/isolated VMs. The tool also provisions them with appropriate network, storage, CPU, memory resources.

The CLI tool configuration and installation is the prerequisite for an IBM Cloud Private installation on the Secure Service Container for IBM Cloud Private environment.

On Linux on Z server, we completed the following steps.

1. Log in as a root user, and install the *jq*[1] and *network-manager*[2] utilities. See the commands in Example 3-1.

Example 3-1 Installing jq and network-manager utilities

```
#Our Linux on Z is Redhat, And we ran:
   yum install -y network-manager-applet
   wget https://github.com/stedolan/jq/releases/download/jq-1.5/jq-1.5.tar.gz
   gunzip -d jq-1.5.tar.gz
   tar -xvf jq-1.5.tar
   cd jq-1.5/
   ./configure --disable-maintainer-mode
   make
   make install
```

2. Navigate to the installation directory and extract the Secure Service Container for IBM Cloud Private archive file as in Downloading Secure Service Container for IBM Cloud Private. Then, install the docker image of the command line tool as shown in Example 3-2.

Example 3-2 Entered into installation directory and installing the docker image

```
cd /opt/blockchain/
docker load < ssc4icp-cli-installer.docker-image.tar
```

3. Downloaded configuration templates to the Linux on Z server by using the command listed in Example 3-3. After the command was completed, a config directory was created with a **hosts** and **ssc4icp-config.yaml** files. These files are using default values and they need an update. These changes are shown in Section 3.5, “Installing IBM Cloud Private cluster” section.

Example 3-3 Downloading the configuration templates file for the Linux on Z server

```
docker run --network=host --rm -v $(pwd):/data
ibmzcontainers/ssc4icp-cli-installer:1.1.0.3 cp -r config /data
```

4. Copy the isolated VM archive file `ICPIsolatedvm.tar.gz` to `/opt/blockchain/config` folder. You need to copy it from where you extract the installation archive file. Keep in mind that the CLI tool uses the isolated VM archive to create cluster nodes on the SSC partition.

3.5 Installing IBM Cloud Private cluster

The following steps were required to set up an IBM Cloud Private environment to deploy and manage dockerized applications in our lab environment.

3.5.1 Configuring Secure Service Container storage

We used the following procedure to make resources such as storage devices and network connections that are assigned to the Secure Service Container partition available in the Secure Service Container for IBM Cloud Private. Keep in mind that these resources can then be used by the containerized applications that run on worker nodes inside the Secure Service Container.

1. To match the disk space requirements of the containerized application, we added storage disks to the Logical Volume (LV) Data Pool Appliance Data, which is provided by the

[1] https://stedolan.github.io/jq/
[2] https://wiki.gnome.org/Projects/NetworkManager

Secure Service Container. For additional instructions, see the following topic in *IBM Z Secure Service Container User's Guide* at https://www-01.ibm.com/support/docview.wss?uid=isg26ab9a15cbe12a81d85258194006e4a89.

– Chapter 14, "Using the Secure Service Container user interface", section "*Viewing and managing storage resources*"

For our SSC partition, we added 13 FICON DASD disks (model 54) as shown in Figure 3-10.

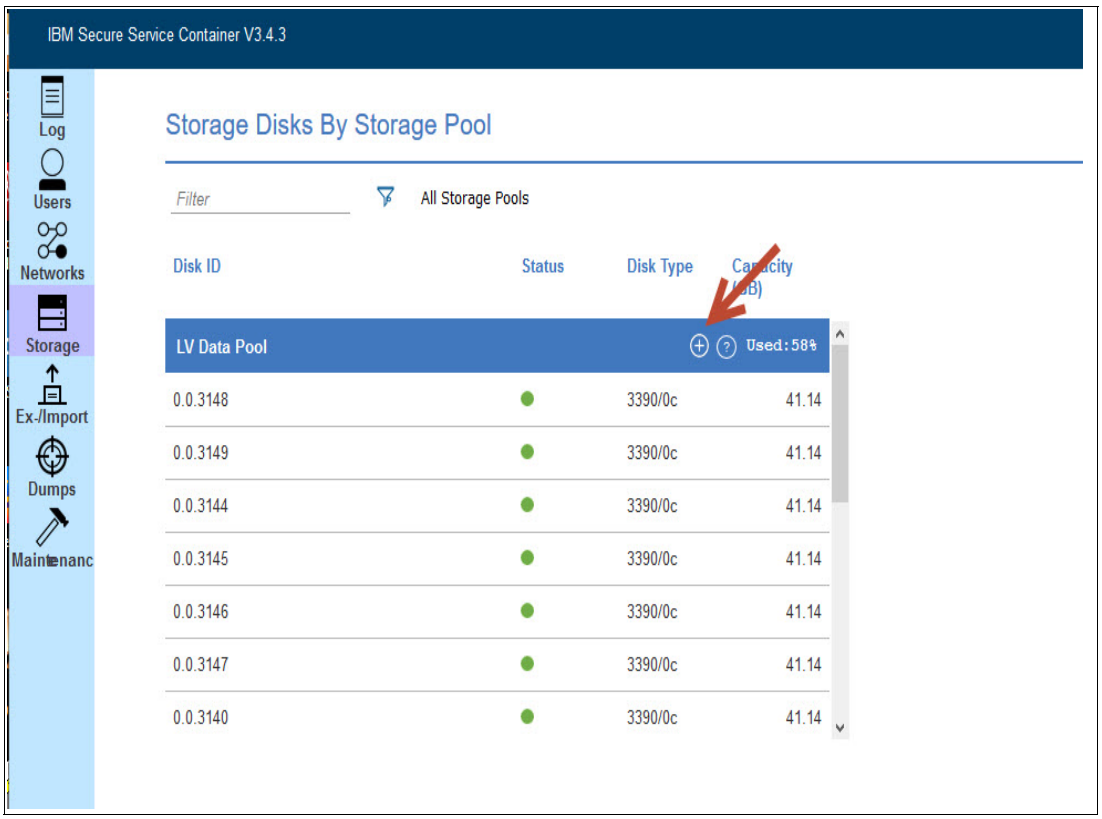


Figure 3-10 Storage allocation using SSC

Tips:

- You can configure the LV data pool size by using the REST API. See the Secure Service Container for IBM Cloud Private System APIs for a full list of REST API endpoints.
- For each worker or proxy node that runs on the Secure Service Container, you must allocate at least 200 GB for the data pool. So, we added more than 400 GB.

3.5.2 Configuring the appliance network

We can configure the network devices for the Secure Service Container for IBM Cloud Private appliance by using the Secure Service Container user interface. The cluster nodes on the Secure Service Container partitions communicate through the Ethernet-type or VLAN-type connections over the network devices that are bound to Open Systems Adapter-Express (OSA-Express) devices.

In our environment, we have both worker and proxy nodes on one Secure Service Container partition. Therefore, we configured two network devices with one for internal communication among the cluster nodes, and another for external access through the proxy node. We can configure one network device to each of the OSA-Express devices on the Secure Service Container partitions, or multiple network devices on one OSA-Express device.

We completed the following steps to configure the network devices.

1. Connect to the Secure Service Container partition (`https://9.16.27.18`) through the browser of your choice, Chrome or Firefox (Safari is not supported).
2. On the Login page, enter the master use ID and password values that you supplied in the image profile, and click **Login**.

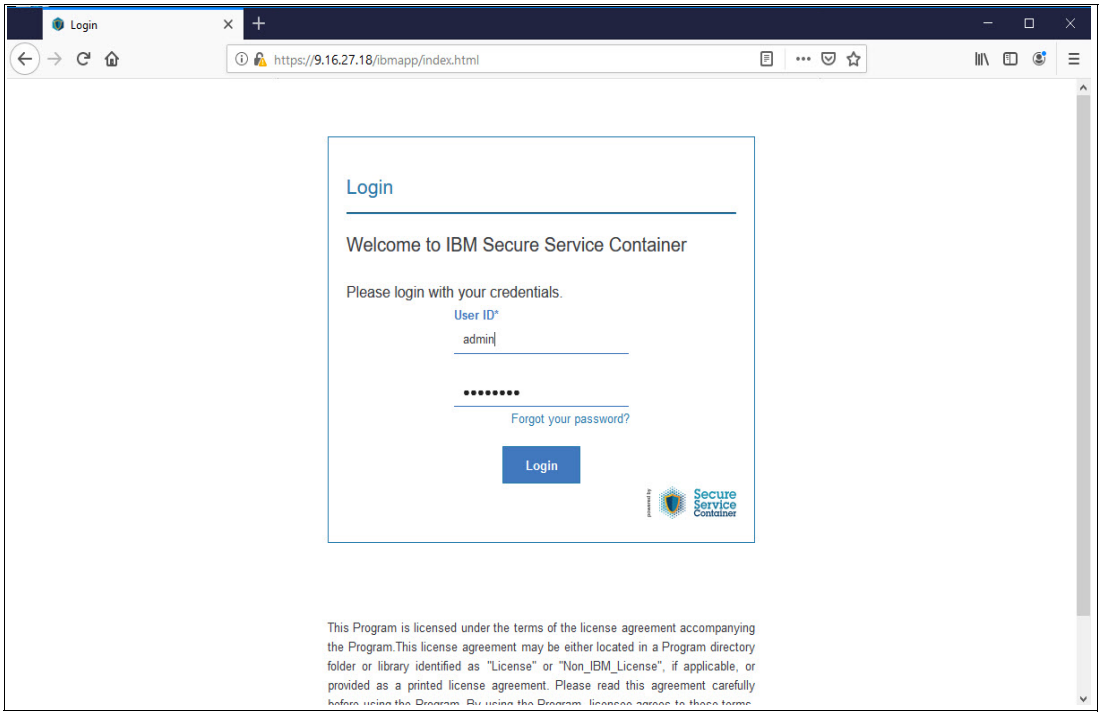


Figure 3-11 Connecting to SSC partition

3. In the navigation pane, click the **Network** icon to display the network connections page.

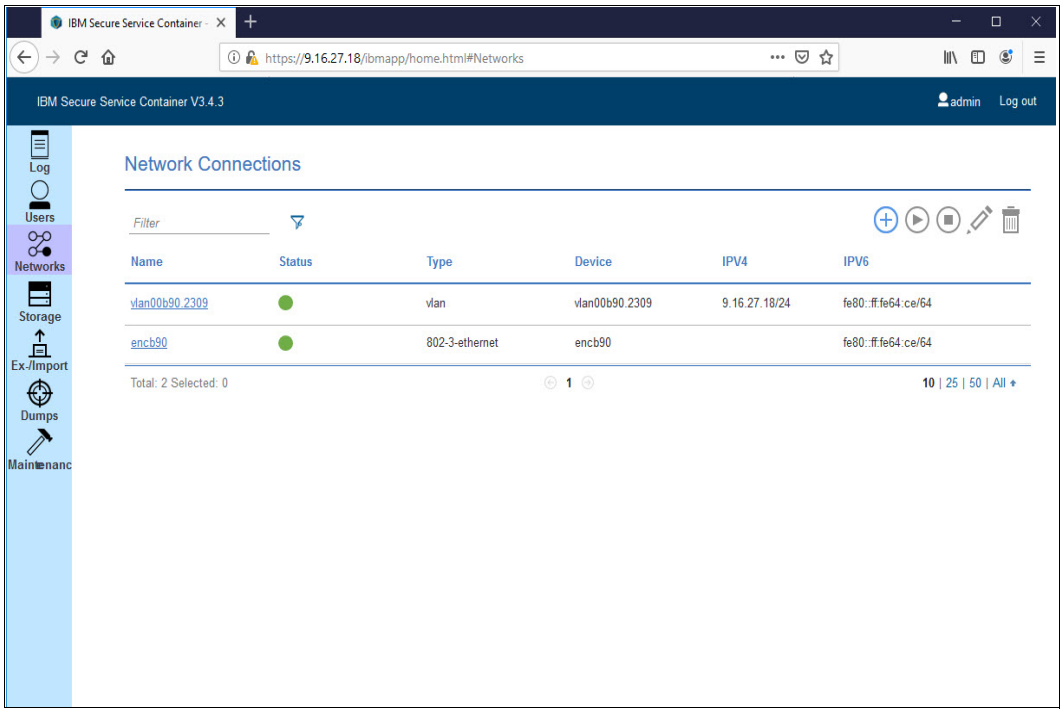


Figure 3-12 SSC network connections

4. Click the plus (+) icon to add a connection for chpid E2 and use device address 0b53. For example, enccw0.0.0b53 is the network device name. After that, select the **B53** network device.

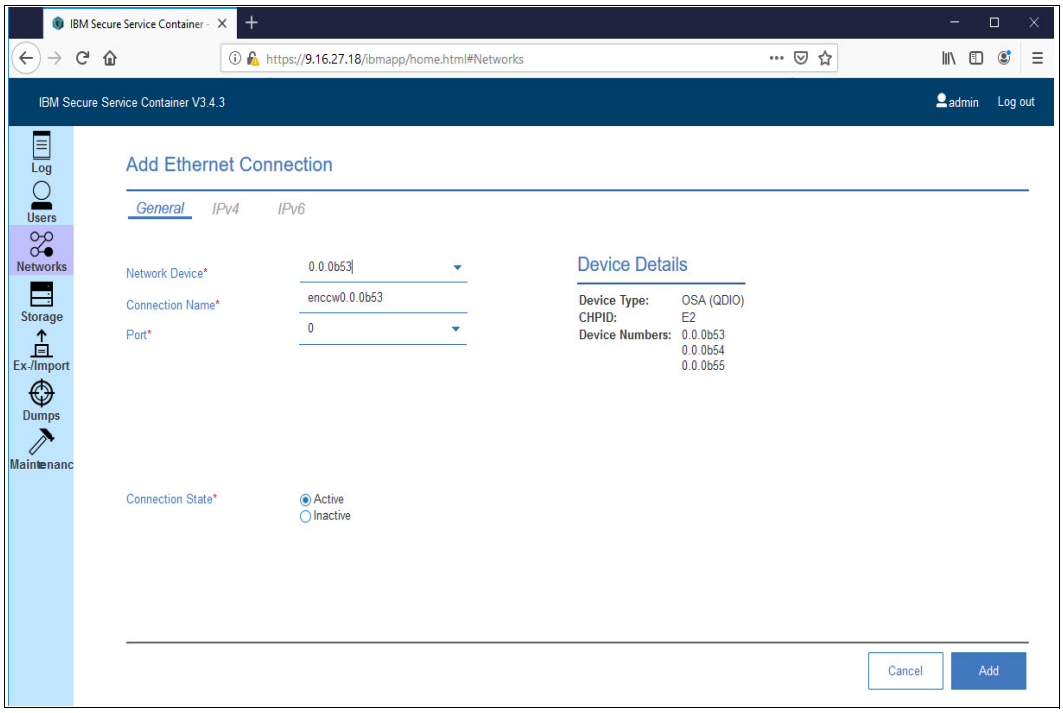


Figure 3-13 Adding the B53 OSA card

5. After you click **Add**, the device is added to the SSC partition. See Figure 3-14.

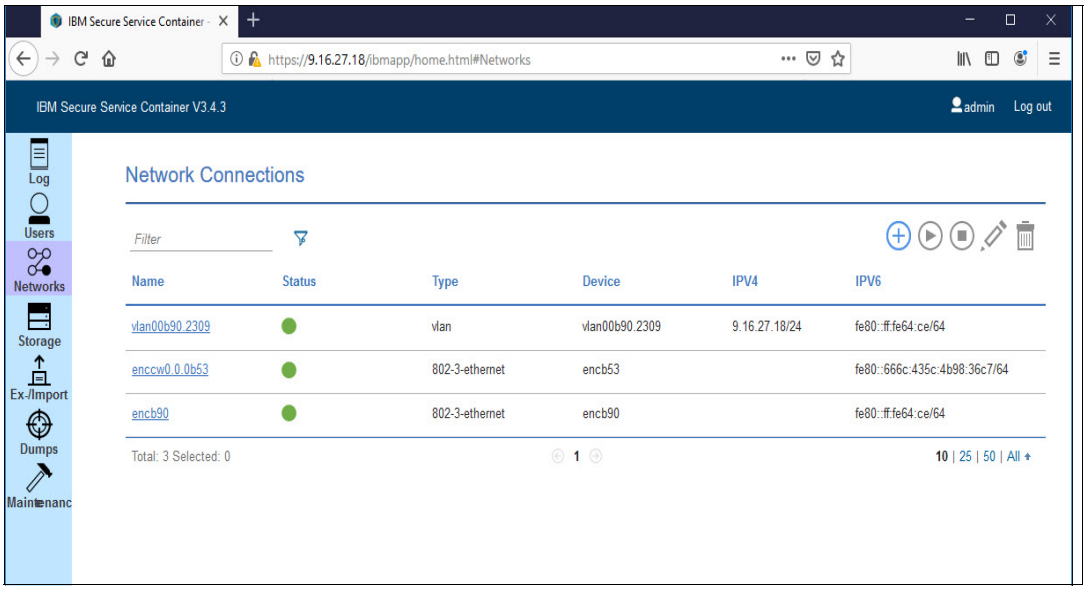


Figure 3-14 Listing all network connections

We have the two network devices to configure the cluster resources. The following list gives some information about each device.

- **vlan00b90.2309**
  A VLAN-type connection for vlan number 2309. We use this device for external cluster communication and to allow connection to our appliance via **`9.16.27.18`**.
- **encb53**
  An ethernet-type connection that we use for internal cluster communication (**`192.168.0.x/24`**).

Additional information about how to add network connections is shown here, in case you need to add new network devices to the SSC partition.

- For an ethernet-type connection:
  a. Click the plus (**+**) icon to add a connection, and then select **Ethernet** as the connection type.
  b. Select a new network device from the drop-down list. Ensure that the CHPID in the Device Details section is different from the one in step 4. For example, the network device name is encf700, and the CHPID is AB.
  c. Use the default value for the Port field, and set the connection state to Active.
- For a VLAN-type connection:

  Ensure that your OSA device is tagged with a VLAN ID (for example, 1121) and the OSA device is connected to the trunk port of the switch.

  a. Click the plus (**+**) icon to add a connection, and then select VLAN as the connection type.
  b. Select a parent device (also known as a tagged OSA device) from the drop-down list. If the parent device is not available, click the plus (**+**) icon to create a parent device. For example, the parent device name is encf300.
  c. Enter the VLAN ID by which the OSA device is tagged, for example, **1121**.
  d. Use the auto-generated connection name, for example, **vxlan0f300.1121**.

e. If the DHCP is not configured in your network, select the **Manual checkbox** on the IPv4 tab, and assign an appropriate IP address according to your network.

f. Set the connection state on the General tab to **Active**.

g. Click **ADD** to save the changes.

Notes:

- Repeat all the steps on each Secure Service Container partition that will be used to host the cluster nodes.
- Record the network device name that will be used for internal communications among the cluster nodes, or by the proxy node for external access. You will use those values for the parent parameter in the **ssc4icp-config.yaml** file. For more information, see *Configuring the cluster resources*: https://www.ibm.com/support/knowledgecenter/en/SSUPZ7_1.1.0.3/topics/configure_cluster_resources.html

For more information, see the following topic in *IBM Z Secure Service Container User's Guide,* https://www-01.ibm.com/support/docview.wss?uid=isg26ab9a15cbe12a81d85258194006e4a89.

– Chapter 14, "Using the Secure Service Container user interface", section "Viewing and managing network connections".

3.5.3 Configuring the cluster resources

Before starting creation of clusters, you must define the resource specifications for each node on the Secure Service Container partitions. The resource specification includes the CPU numbers, memory size, port range, and network settings.

On the Linux on Z server, we completed the following steps as a root user.

1. Go to the config directory (for example `/opt/blockchain/config`) where you extracted the Secure Service Container for IBM Cloud Private archive file. Update the hosts file to configure the password authentication for cluster nodes.

   In the following example **hosts** file, the Master user ID in the login setting of the Secure Service Container partition with IP address 9.16.27.19 was set to `master` and the Master password to `somesecurepassword`.

Example 3-4 Sample of /opt/blockchain/config/hosts file

```
[master]
9.16.27.19 ansible_user="root" ansible_ssh_pass="somesecurepassword"
ansible_ssh_common_args="-oPubkeyAuthentication=no" ansible_become="true"
[worker]
9.16.27.18  rest_user="admin" rest_pass="ssc_master_password"
```

– The `[master]` section contains Linux on Z server details with IP address, username, password for IBM Cloud Private master installation.

– The `[worker]` section contains Secure Service Container partition IP address, username, and password for the zAppliance REST API. Keep in mind that if you use different Secure Service Container partitions to host the worker or proxy nodes, you must list each partition under the `[worker]` section. The zAppliance REST API username rest_user and password rest_pass values are the user ID and password used to log in to the Secure Service Container partition and UI. This values are initially set in the login setting of the Secure Service Container partition as:

  - Master user ID

- Master password

2. Create the **ssc4icp-config.yaml** file to configure the nodes on the Secure Service Container partition with required CPU, memory, port range, and network specifications with values of our lab. See Example 3-5.

   The **ssc4icp-config.yaml** file (Example 3-5) shows that one cluster DemoCluster contains these nodes:

   – One worker node with resources specified in the template1 on the Secure Service Container partition with an IP address 9.16.27.18.

   – One proxy node with resources specified in the template2 on the same Secure Service Container partition.
   See the **ssc4icp-config.yaml** file that we used in our cluster configuration.

3. Remember to use the information in section 1.4, “Our lab environment” on page 12 to build the file.

Example 3-5 Sample of ssc4icp-config.yaml file for our lab environment

```
cluster:
  name: "DemoCluster"
  masterconfig:
    internal_ips: ['192.168.0.251']
    subnet: "192.168.0.0/24"

LPARS:
  - ipaddress: '9.16.27.18'
    containers:
      - template: "template1"
        count: 1
        internal_ips: ['192.168.0.253']

  - ipaddress: '9.16.27.18'
    containers:
      - template: "template2"
        count: 1
        internal_ips: ['192.168.0.254']
        proxy_external_ips: ['9.16.27.25']

template1:
 name: "worker1"
 type: "WORKER"
 # cpu defined by Number of threads
 cpu: "2"
 # Memory in MB
 memory: "4098"
 port_range: '15000'
 root_storage: "50G"
 icp_storage: "140G"
 internal_network:
   subnet: "192.168.0.0/24"
   gateway: "192.168.0.1"
   parent: "encb53"

template2:
 name: "proxy1"
 type: "PROXY"
```

```
# cpu defined by Number of threads
cpu: "2"
# Memory in MB
memory: "4096"
port_range: '16000'
root_storage: "50G"
icp_storage: "140G"
internal_network:
  subnet: "192.168.0.0/24"
  gateway: "192.168.0.1"
  parent: "encb53"
proxy_external_network:
  subnet: "9.16.27.0/24"
  gateway: "9.16.27.1"
  parent: "vlan00b90.2309"
```

Table 3-2 describes each parameter of the **ssc4icp-config.yaml** file.

Table 3-2 Parameters description for ssc4icp-conf.yaml file

Parameter	Description
datapool	Defines that the quotagroup still exists after the containers are deleted. If the value is not set for the datapool parameter, the CLI deletes quotagroup after the uninstallation. Notice that we did not define any datapool. However, description is here in case you need to set up it.
masterconfig	Defines the network configurations for the master node.
internal_ips	Defines an array of IP addresses for each worker and proxy node.
proxy_external_ips	Defines an external IP addresses for the proxy node. The external workloads can use the value of proxy_external_network_ip to access the proxy node.
count	Defines the number of nodes that will be created on the partition. Note that the value of count is an integer and can not be enclosed by using the quotation marks.
cpu	Defines the number of CPU cores to be assigned for the node.
memory	Defines the memory size (in MB) to be assigned for the node.
type	Defines the type of node to be created by the CLI tool. The value must be WORKER, PROXY, or STORAGE
name	Defines the name of the proxy or worker node. The maximum length of a node name is 20 characters.
internal_network	Defines the subnet, gateway, and parent network interface settings of the worker or proxy node.
proxy_external_network	Defines the external subnet, gateway, and parent network interface settings of the proxy node.
parent	Defines the parent network device name that data traffic will physically go through on the node.
port_range	Defines the starting port number on each Secure Service Container partition to be assigned to each node.

Parameter	Description
root_storage	Defines the size of storage (G for GB or M for MB) allocated to the root file system. It must be set to at least what is required by IBM Cloud Private. IBM Cloud Private requires storage under the root file system that is used to store temporary files during the installation. For example, IBM Cloud Private 3.2.0 requires 50 GB under the root file system. Therefore, the root_storage parameter must be set to 50 GB plus an adequate buffer for the operating system on the node itself.
icp_storage	Defines the size of the storage (G for GB or M for MB) allocated to the IBM Cloud Private node runtime. It must be set to at least the sum of the directory sizes used by a node at runtime, as specified by the IBM Cloud Private system requirements. For example, based on IBM Cloud Private 3.2.0 system requirements, a node requires at runtime: ► at least 100 GB under /var/lib/docker ► at least 10 GB under /var/lib/kubelet Therefore, the icp_storage parameter must be set to 110 GB plus an adequate buffer to run custom Kubernetes applications. The default value is 140 GB.

3.5.4 Creating the cluster nodes

We used the Secure Service Container for IBM Cloud Private command line interface (CLI) tool to create all the necessary cluster nodes and provision them with appropriate network, storage, CPU, memory resources.

The CLI tool configuration and installation is the prerequisite for IBM Cloud Private installation on the Secure Service Container for IBM Cloud Private environment.

We completed the following steps as a root user on the Linux on Z server (master node):

1. Run the command line tool to create the cluster nodes.
   You must run in the parent directory of the config directory, for example, `/opt/blockchain`.

```
docker run --network=host --rm -it -v
$(pwd)/config:/ssc4icp-cli-installer/config
ibmzcontainers/ssc4icp-cli-installer:1.1.0.3 install
```

After the installation is complete, the following directories and files are created under the config directory (`/opt/blockchain/config`) for future reference and use. See Table 3-3.

Table 3-3 List of directories and files created by the ssc4icp installer

File/Directory name	Description
Logs	A directory that contains logs for all operations being performed
Cluster-status.yaml	A file that is used to capture the current status of installation

File/Directory name	Description
DemoCluster	A directory with your cluster name that contains the following files for the cluster: **cluster-configuration.yaml** - A file indicates that the cluster configurations in the ssc4icp-config.yaml file are applied successfully. **quotagroup-symlink.yaml** - A file contains details of storage containers, quotagroups and device names if you specify GlusterFS configurations in the ssc4icp-config.yaml file. For more information, see Deploying GlusterFS. **ipsec.conf** - A file that contains the network topology of the cluster. **ipsec.secret** - A file that contains a randomly generated Pre-Shared-Key (PSK) that will be used as an authorization token to the IPSec network. An ssh key pair **ssh_key** and **ssh_key.pub** files that provide SSH access for the IBM Cloud Private installer to all the cluster nodes. In order for the IBM Cloud Private installer to access your master or boot node over SSH and also to use the generated SSH key, you must follow the instructions in the *Before you begin* section of *Deploying IBM Cloud Private*: https://www.ibm.com/support/knowledgecenter/en/SSUPZ7_1.1.0.3/topics/install_icp.html

The following **cluster-configuration.yaml** example file (Example 3-6 on page 90) was generated based on the cluster configuration that is specified in our **ssc4icp-config.yaml** file.

Example 3-6 Output of /opt/blockchain/config/DemoCluster/cluster-configuration.yaml file

```
LPARS:
- containers:
  - cpu: '2'
    icp_storage: 140000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.253
      parent: encb53
      subnet: 192.168.0.0/24
    memory: '4098'
    name: worker1-15001
    port: '15001'
    root_storage: 50000M
  ipaddress: 9.16.27.18
- containers:
  - cpu: '2'
    icp_storage: 140000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.254
      parent: encb53
      subnet: 192.168.0.0/24
    memory: '4096'
    name: proxy1-16001
    port: '16001'
    proxy_external_network:
      gateway: 9.16.27.1
```

```
      ip: 9.16.27.25
      parent: vlan00b90.2309
      subnet: 9.16.27.0/24
    root_storage: 50000M
  ipaddress: 9.16.27.18
cluster:
  ipsecsecrets:
KjOrL+/az8yN3rEeqLP9mgzTUYI38x6qMFZk1DU9cqiz8lXzbS5NYGNnplQoyl/kY/1kFuqs9M3RlkrFfQ
rBtQ==
  masterconfig:
    internal_ips:
    - 192.168.0.251
    subnet: 192.168.0.0/24
  name: DemoCluster
  repoid: ICPIsolatedvm
```

Notes:

- The generated directories and files must not be deleted because the uninstallation procedure needs to validate those files when you reset the environment. It is recommended that you back them up after cluster installation.
- The private key `ssh_key` must be protected because it provides SSH access to all of the cluster nodes. Only the system administrator who will install the IBM Cloud Private can have the access privilege to the **ssh_key** file.
- If you run the command line tool again with a same cluster name but different configurations, you must delete the `config/<ClusterName>` directory first.
- The `ipsec.secrets` file contains the IPSec key that is generated by the command line tool, and is used when creating the network configuration for each node. You can use LUKS (Linux Unified Key Setup) hardware encryption to protect the key from access other than the root user on Linux on Z server.

Verify the containers status by using the following instructions. The containers for IBM Cloud Private cluster will be displayed and you will be able to see the status of each container.

a. Create a **get_containers.sh** file under your installation folder (`/opt/blockchain`) with the content that is listed in Example 3-33.

Example 3-7 Sample of get_containers.sh script

```
#!/bin/bash
# <emarins@br.ibm.com>

if [ "$1" != "" ]; then
   USERNAME=$1
else
   echo 'What is the SSC user admin name? (ie. admin)'
   read option
   USERNAME="$option"
fi

if [ "$2" != "" ]; then
        PASSWORD=$2
else
   echo 'What is the SSC admin password?'
```

```
   read -s option
   PASSWORD="$option"
   printf "%s\n" "${PASSWORD//?/*}"
fi

if [ "$3" != "" ]; then
        server=$3
else
   echo 'What is the SSC IP address or FQDN?'
   read option
   server="$option"
fi

# Getting Token
token=$(curl --request POST --url
https://$server/api/com.ibm.zaci.system/api-tokens -H 'accept:
application/vnd.ibm.zaci.payload+json' -H 'cache-control: no-cache' -H
'content-type: application/vnd.ibm.zaci.payload+json;version=1.0' -H 'zaci-api:
com.ibm.zaci.system/1.0' --insecure --data '{ "kind" : "request", "parameters"
: { "user" : "'$USERNAME'", "password" : "'$PASSWORD'" } }' 2>/dev/null |
python -m json.tool |grep token | awk -F\: '{print $2}'|tr -d \" |tr -d
[:blank:])

if [ $(echo $token|wc -c) -eq 210 ]; then
   echo
   #echo "Token is valid"
else
   echo "Invalid Token. Exiting..."
   exit 1
fi

sleep 1s

output=$(curl --silent -k -X GET "https://$server/api/com.ibm.zaas/containers"
-H "accept: application/vnd.ibm.zaci.payload+json" -H "zACI-API:
com.ibm.zaci.system/1.0" -H "Content-Type:
application/vnd.ibm.zaci.payload+json;version=1.0" -H "Authorization: Bearer
$token")

#echo $output | python -m json.tool

containers=$(echo $output | python -m json.tool |grep -A1 Names |grep \/|tr -d
\" |tr -d \/)
#echo $containers

echo "-----------------------------------------------"
for container in $containers;
do
   echo "Container: $container"
   echo "-----------------------------------------------"
   output=$(curl --silent -k -X GET
"https://$server/api/com.ibm.zaas/containers/$container" -H  "accept:
application/vnd.ibm.zaci.payload+json" -H  "Authorization: Bearer $token" -H
"zACI-API: com.ibm.zaci.system/1.0" -H  "Content-Type:
application/vnd.ibm.zaci.payload+json;version=1.0")
```

```
   echo $output  | python -m json.tool | egrep "IPv4Address|Status" |grep -v
SecondaryIPAddresses|tr -d \" | sed -e 's/^[[:space:]]*//g' -e
's/[[:space:]]*\$//g' |tr -d \,
   echo "-----------------------------------------------"
   echo
   sleep 1s
done
```

b. Change permission for this script by using the following command to allow users to execute it: **chmod +x /opt/blockchain/get_containers.sh**
c. Now, issue the script to get the status of the containers and IP address information. See output in Example 3-34.

   To execute this script, you need to provide the information about the SSC admin name, SSC password and the SSC IP address. This script uses Secure Service Container system APIs to authenticate and get the information about the containers.

   Syntax of the script is `get_containers.sh <SSC_ADMIN> <SSC_PASSWORD> <SSC_IP>`. Example 3-8 is an example of the command.

Example 3-8 Getting information about the containers

```
[root@ssc4icp-master blockchain]# /opt/blockchain/get_containers.sh admin
password 9.16.27.18

Container: worker1-15001
-----------------------------------------------
IPv4Address: 192.168.0.253
Status: running
Status: Up 13 hours
-----------------------------------------------

Container: proxy1-16001
-----------------------------------------------
IPv4Address: 9.16.27.25
IPv4Address: 192.168.0.254
Status: running
Status: Up 13 hours
-----------------------------------------------

[root@ssc4icp-master blockchain]#
```

3.5.5 Configuring the network on the master node

You need to configure the network on Linux on Z server to ensure that the master node is connected with other cluster nodes on the LinuxONE system.

Before you deploy the SSC containers, ensure that the network interfaces (NIC) and IP addresses are correctly configured and that IPs can be pinged from your master node. If you do not know how to set it up, refer to https://www.ibm.com/support/knowledgecenter/en/SSUPZ7_1.1.0.3/topics/configure_network_master.html.

The IPSec must be installed and configured to allow the communication with the worker and proxy nodes that are running on the SSC partition. The IPSec ensures that the data traffic within the network is encrypted. IPSec can operate in two different modes: transport or tunnel. The transport mode is sufficient for encryption of the provided IP traffic. To configure IPSec,

you must ensure that the **strongSwan** daemon is installed. See **strongSwan** https://strongswan.org for more details.

To install **strongSwan** on our Redhat Linux on Z server (master node), follow these steps:

For Redhat on IBM Z

1. Download the **strongswan** source package and build the binary on the LinuxONE system.

Example 3-9 Compiling and installing strongswan package on Redhat

```
yum install gmp-devel
wget http://download.strongswan.org/strongswan-5.6.2.tar.bz2
tar xjvf strongswan-5.6.2.tar.bz2
cd strongswan-5.6.2/
./configure --prefix=/usr --sysconfdir=/etc
make
make install
ipsec version
ipsec start
```

Note: You might experience network connectivity problems when using the **strongswan** 5.6.2 with Redhat 7 as the master node on IBM Z because of this known issue Using /32 groups in ipsec causing leaks https://access.redhat.com/solutions/4251881.
To work around the problem, create a **cron** job to run the **ipsec restart** command every 30 minutes on the master node.

For information on how to build the **strongSwan** package, see *strongSwan Installation Documentation* at https://wiki.strongswan.org/projects/strongswan/wiki/InstallationDocumentation.

For other distributions

Refer to https://www.ibm.com/support/knowledgecenter/en/SSUPZ7_1.1.0.3/topics/configure_network_master.html or strongSwan's manual for additional details.

Assuming that network IP addresses and network settings are in place, execute the following steps to complete the pre-setup steps for ICP installation:

1. Copy the following two files into the /etc directory as shown in Example 3-10. Those two files are generated in the config/<*ClusterName*> directory after the Secure Service Container for IBM Cloud Private CLI tool is installed.
   - `config/<ClusterName>/ipsec.conf`
     This file contains the network topology of the cluster.
   - `config/<ClusterName>/ipsec.secret`
     This file contains a randomly generated Pre-Shared-Key (PSK) that is be used as an authorization token to the IPSec network.

Example 3-10 Copying the ipsec files to /etc/

```
cp -a /opt/blockchain/config/DemoCluster/ipsec* /etc/
```

2. Start the **strongSwan** daemon to apply the changes by using the **`systemctl strongswan restart`** command. Also, remember to make configuration persistent by issuing this command:
   `systemctl enable strongswan`

3. Test the internal and external connection to each cluster node on the LinuxONE system by using the **ping** command as can be seen in Example 3-11.

Example 3-11 Verifying if IPs are reachable from master node

```
[root@ssc4icp-master ~]# ping -c2 192.168.0.253
PING 192.168.0.253 (192.168.0.253) 56(84) bytes of data.
64 bytes from 192.168.0.253: icmp_seq=1 ttl=64 time=0.388 ms
64 bytes from 192.168.0.253: icmp_seq=2 ttl=64 time=0.516 ms

--- 192.168.0.253 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.388/0.452/0.516/0.064 ms
[root@ssc4icp-master ~]# ping -c2 192.168.0.254
PING 192.168.0.254 (192.168.0.254) 56(84) bytes of data.
64 bytes from 192.168.0.254: icmp_seq=1 ttl=64 time=0.399 ms
64 bytes from 192.168.0.254: icmp_seq=2 ttl=64 time=0.430 ms

--- 192.168.0.254 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.399/0.414/0.430/0.025 ms
[root@ssc4icp-master ~]# ping -c2 9.16.27.25
PING 9.16.27.25 (9.16.27.25) 56(84) bytes of data.
64 bytes from 9.16.27.25: icmp_seq=1 ttl=64 time=1.08 ms
64 bytes from 9.16.27.25: icmp_seq=2 ttl=64 time=0.478 ms

--- 9.16.27.25 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.478/0.783/1.089/0.306 ms
[root@ssc4icp-master ~]#
```

You can use **ipsec status** command to verify the status of ipsec connections for the internal IP addresses (Example 3-12).

Example 3-12 Checking ipsec status

```
[root@ssc4icp-master ~]# ipsec status
Routed Connections:
main192.168.0.251{1}:  ROUTED, TRANSPORT, reqid 1
main192.168.0.251{1}:   192.168.0.0/24 === 192.168.0.0/24
Security Associations (2 up, 0 connecting):
main192.168.0.251[40]: ESTABLISHED 26 minutes ago,
192.168.0.251[192.168.0.251]...192.168.0.254[192.168.0.254]
main192.168.0.251{90}:  INSTALLED, TRANSPORT, reqid 1, ESP SPIs: c06be3c1_i
c1951f88_o, IPCOMP CPIs: e8e4_i d980_o
main192.168.0.251{90}:   192.168.0.251/32 === 192.168.0.254/32
main192.168.0.251[38]: ESTABLISHED 47 minutes ago,
192.168.0.251[192.168.0.251]...192.168.0.253[192.168.0.253]
main192.168.0.251{91}:  INSTALLED, TRANSPORT, reqid 1, ESP SPIs: cbc344b7_i
ca4bb55f_o, IPCOMP CPIs: e04c_i 6998_o
main192.168.0.251{91}:   192.168.0.251/32 === 192.168.0.253/32
[root@ssc4icp-master ~]#
```

Additional information about the Linux on Z server

To allow the Linux server to communicate to the SSC containers (worker, proxy, and storage nodes), we defined the following configuration in the Linux on Z master node and z/VM user directory for the Linux instance. Remember that the device names and IP addresses might need to change to fit your environment.

The VM directory

As server is running on z/VM environment, we defined the VM user ID to have the required network devices for the cluster connections.

- 1F00, 1F01, and 1F02 virtual devices are using devices on chpid E2 (0B53, 0B54, and 0B55) that are in the same channel where the SSC partition was defined.
- 1000 is the virtual interface that is attached to the external network, and this device is configured to couple to the network using VLAN 2309.

In Figure 3-15, see the text that is highlighted in **bold** under the virtual network devices that are defined to the Linux on Z server.

```
USER LNXLG001 DIRCOPY 16G 16G G                                    08192211
   INCLUDE DFLTLNX                                                 08192211
   CPU 00 BASE                                                     08192211
   CPU 01                                                          08192211
   CPU 02                                                          08192211
   CPU 03                                                          08192211
   CPU 04                                                          08192211
   CPU 05                                                          08192211
   CPU 06                                                          08192211
   CPU 07                                                          08192211
   IPL CMS                                                         08192211
   POSIXINFO   UID 100754                                          08192211
   VMRELOCATE ON                                                   08192211
   DEDICATE 1F00 0B53                                              08192211
   DEDICATE 1F01 0B54                                              08192211
   DEDICATE 1F02 0B55                                              08192211
   NICDEF 1000 TYPE QDIO LAN SYSTEM VSWITCHM                       08192211
   NICDEF 1000 VLAN 2309                                           08192211
   MDISK 0100 3390 0001 30042 LXADE2 M                             08192211
   MDISK 0500 3390 0001 60101 LXADE3 M                             08192211
   MDISK 0300 3390 0001 60101 LXADE4 M                             08192211
   MDISK 0301 3390 0001 60101 LXADE5 M                             08192211
   MDISK 0302 3390 0001 60101 LXADDL M                             08192211
   MDISK 0303 3390 0001 60101 LXADDM M                             08192211
   MDISK 0501 3390 0001 60101 LXADDN M                             08192211
   MDISK 0502 3390 0001 60101 LXLNGJ M                             08192211
*DVHOPT LNKO LOG1 RCM1 SMSO NPW1 LNGAMENG PWC20190815 CRC N        08200002
```

Figure 3-15 Sample for the VM user ID directory for the Linux on Z master node

Linux network configuration files for the Linux on Z server

The Example 3-13 shows the Linux configuration files that are used to set up our environment.

Example 3-13 Linux configuration files

```
[root@ssc4icp-master ~]# cat /etc/sysconfig/network-scripts/ifcfg-enccw0.0.1000
DEVICE=enccw0.0.1000
UUID=135203ac-2871-4b0c-ae48-a5793380a5dd
ONBOOT=yes
```

```
BOOTPROTO=static
MTU=1500
SUBCHANNELS=0.0.1000,0.0.1001,0.0.1002
IPADDR=9.16.27.19
NETMASK=255.255.255.0
BROADCAST=9.16.27.255
GATEWAY=9.16.27.1
DNS1=9.0.130.50
DNS="9.0.130.50"
DOMAIN="boulder.ibm.com"
NETTYPE=qeth
PORTNAME=FOOBAR
OPTIONS=''
ZONE=public
[root@ssc4icp-master ~]# cat /etc/sysconfig/network-scripts/ifcfg-enccw0.0.1f00
DEVICE=enccw0.0.1f00
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.0.251
NETMASK=255.255.255.0
BROADCAST=192.168.0.255
#GATEWAY=192.168.0.1
DOMAIN="boulder.ibm.com"
```

To list the active network interfaces, issue the commands that are listed in Example 3-14. This output contains several network information details, including the chpid number that might be helpful to identify the physical port that is in use for the device.

Example 3-14 Listing Active network devices on the Master Node

```
[root@ssc4icp-master ~]# lsqeth
Device name                   : enccw0.0.1000
--------------------------------------------------------------------------------
        card_type             : VSWITCH: SYSTEM VSWITCHM (Type: QDIO)
        cdev0                 : 0.0.1000
        cdev1                 : 0.0.1001
        cdev2                 : 0.0.1002
        chpid                 : 00
        online                : 1
        portname              : FOOBAR
        portno                : 0
        state                 : UP (LAN ONLINE)
        priority_queueing     : always queue 2
        buffer_count          : 64
        layer2                : 1
        isolation             : none
        bridge_role           : none
        bridge_state          : inactive
        bridge_hostnotify     : 0
        bridge_reflect_promisc : none

Device name                   : enccw0.0.1f00
--------------------------------------------------------------------------------
        card_type             : OSD_10GIG
        cdev0                 : 0.0.1f00
```

```
cdev1                : 0.0.1f01
cdev2                : 0.0.1f02
chpid                : E2
online               : 1
portno               : 0
state                : UP (LAN ONLINE)
priority_queueing    : always queue 2
buffer_count         : 64
layer2               : 1
isolation            : none
bridge_role          : none
bridge_state         : inactive
bridge_hostnotify    : 0
bridge_reflect_promisc : none
switch_attrs         : unknown
```

3.6 Deploying IBM Cloud Private

This section describes deployment of the IBM Cloud Private cluster to the cluster nodes on Linux on Z server and Secure Service Container partitions. Notice that the Linux on Z server is the master node and boot node for the IBM Cloud Private cluster.

The following steps can be used to define an IBM Cloud Private environment to deploy and manage dockerized applications.

1. Create the required installation folder.

   ```
   mkdir /opt/icp320/
   ```

2. Copy the *ibm-cloud-private-s390x-3.2.0.gz* image file to **/opt/icp320**.

3. From the installation directory (**/opt/icp320**), load the IBM Cloud Private archive file as shown in Example 3-15.

Example 3-15 Loading Docker container images

```
[root@ssc4icp-master cluster]# tar xf ibm-cloud-private-s390x-3.2.0.tar.gz -O |
sudo docker load
Loaded image: ibmcom/icp-vip-manager-s390x:1.1
Loaded image: ibmcom/kafka-s390x:0.10.0.4
Loaded image: ibmcom/tiller-s390x:v2.12.3-icp-3.2.0
Loaded image: ibmcom/heketi-s390x:v8.0.0.1

                                Output snippet

Loaded image: ibmcom/istio-servicegraph-s390x:1.0.2
Loaded image: ibmcom/ma-file-wl-gen-s390x:3.2.0
Loaded image: ibmcom/metering-data-manager-s390x:3.2.0
Loaded image: ibmcom/compliance-annotator-s390x:3.2.0
Loaded image: ibmcom/cos-indexer-s390x:3.2.0
Loaded image: ibmcom/grafana-s390x:5.2.0-f3
Loaded image: ibmcom/kubectl-s390x:v1.13.5
[root@ssc4icp-master cluster]#
```

Note: If you find problems with the installation of ICP, follow instructions on Installing an IBM Cloud Private Enterprise environment. There, you find detailed information about the ICP installation.

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/installing/install_containers.html

4. Extract the configuration files from the installer image by using the command in Example 3-16.

Example 3-16 Extracting the configuration files from the installer image

```
sudo docker run -v $(pwd):/data -e LICENSE=accept \
ibmcom/icp-inception-s390x:3.2.0-ee \
cp -r cluster /data
```

5. Run the **ls cluster** command to confirm that the cluster folder was created.
6. Create the images directory under **/opt/icp320/cluster** folder:
mkdir /opt/icp320/cluster/images
7. Move the **/opt/icp320/ibm-cloud-private-s390x-3.2.0.tar.gz** image file to **/opt/icp320/cluster/images** directory:
mv /opt/icp320/ibm-cloud-private-s390x-3.2.0.tar.gz /opt/icp320/cluster/images/
8. Copy the SSH private key **ssh_key** file for the cluster to the IBM Cloud Private installation directory by running this command, where **/opt/<installation-directory>/config/<cluster_name>/ssh_key** is the location of the generated SSH private key for your cluster and /opt/icp320/cluster is the directory that is configured for IBM Cloud Private installation:
cp -p /opt/blockchain/config/DemoCluster/ssh_key* /opt/icp320/cluster
9. Ensure that the new **ssh_keys** can be used to access the Linux master node (192.168.0.251). We ran the command in Example 3-17 to authorize logins by using the SSH keys.

Example 3-17 Authorizing the use of SSH_keys on Linux Master node

```
[root@ssc4icp-master cluster]# ssh-copy-id -i
/opt/blockchain/config/DemoCluster/ssh_key.pub root@192.168.0.251
/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/opt/blockchain/config/DemoCluster/ssh_key.pub"
The authenticity of host '192.168.0.251 (192.168.0.251)' can't be established.
ECDSA key fingerprint is SHA256:hhlO+P8k6LaZ6EMRG5vwy/Ea9wGcZLJRtoi5r9hzu7Q.
ECDSA key fingerprint is MD5:75:3a:c2:12:39:ca:23:f7:5b:58:03:25:54:99:2a:13.
Are you sure you want to continue connecting (yes/no)? yes
/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out
any that are already installed
/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted
now it is to install the new keys

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@192.168.0.251'"
and check to make sure that only the key(s) you wanted were added.

[root@ssc4icp-master cluster]#
```

10. Before you start the installation, confirm communications between the Linux on Z master node and the SSC containers. We used ssh commands that are listed in Example 3-18.

Example 3-18 Checking if communication with SSC containers are OK

```
### Checking network connection with 192.168.0.254 (proxy node) ###

[root@ssc4icp-master ~]# ssh -i /opt/blockchain/config/DemoCluster/ssh_key
root@192.168.0.254 date
Sun Aug 18 14:37:08 UTC 2019

## Now checking network connection with 192.168.0.253 (worker node) ###

[root@ssc4icp-master ~]# ssh -i /opt/blockchain/config/DemoCluster/ssh_key
root@192.168.0.253 date
Sun Aug 18 14:37:14 UTC 2019
[root@ssc4icp-master ~]#
```

11. Use commands in Example 3-19 to get hostnames for the proxy and work nodes that were defined during the SSC installation. This information is required to perform the ICP installation. (Optionally, you can configure **skip_host_ip_check: true** value pair in the *config.yaml* file to skip checking IP addresses in the hosts file)

Example 3-19 Getting worker and proxy hostnames

```
[root@ssc4icp-master cluster]# ssh -i /opt/icp320/cluster/ssh_key
root@192.168.0.253 cat /etc/hostname
worker1-15001
[root@ssc4icp-master cluster]# ssh -i /opt/icp320/cluster/ssh_key
root@192.168.0.254 cat /etc/hostname
proxy1-16001
```

12. Connect through SSH to worker and proxy nodes by using the SSH key and update the /etc/hosts entries as shown Example 3-20 below.

Example 3-20 Sample of /etc/hosts entries

```
192.168.0.251 ssc4icp-master
192.168.0.253 worker1-15001
192.168.0.254 proxy1-16001
```

13. To make the /etc/hosts updates easier, we recommend that you update the /etc/hosts file on the master node server (192.168.0.251) first and then send it over the worker and proxy nodes by using the two commands in Example 3-21.

Example 3-21 Updating /etc/hosts files on the worker and proxy nodes

```
[root@ssc4icp-master ~]# scp -i /opt/icp320/cluster/ssh_key /etc/hosts
root@192.168.0.253:/etc/
hosts
100%  497   204.0KB/s   00:00

root@ssc4icp-master ~]# scp -i /opt/icp320/cluster/ssh_key /etc/hosts
root@192.168.0.254:/etc/
hosts
100%  497   155.0KB/s   00:00
```

14. Update **/opt/icp320/cluster/hosts** file with the internal IPs for the master, worker, and proxy node as shown in Example 3-22.

Example 3-22 Sample of /opt/icp320/cluster/hosts file

```
[master]
192.168.0.251

[worker]
192.168.0.253

[proxy]
192.168.0.254

#[management]
#4.4.4.4

#[va]
#5.5.5.5
```

15. Also, update **/opt/icp320/cluster/config.yaml** file with the modifications that are listed in Example 3-23.

Note: When you customize the IBM Cloud Private cluster, the value of the **cluster_lb_address** parameter in the ***<icp_installation_directory>*/cluster/config.yaml** file must be set to the public IP address of master node that will be accessed. And the **proxy_lb_address** parameter must be set to the public IP address of proxy node.

Example 3-23 config.yaml modifications

```
## Remove the restriction for complex passwords
password_rules:
- '(.*)'

## Advanced Settings
default_admin_user: admin
default_admin_password: abc12345

## External loadbalancer IP or domain
## Or floating IP in OpenStack environment
# cluster_lb_address: none
cluster_lb_address: 9.16.27.19

## External loadbalancer IP or domain
## Or floating IP in OpenStack environment
# proxy_lb_address: none
proxy_lb_address: 9.16.27.25

## Install in firewall enabled mode
# firewall_enabled: false
firewall_enabled: false

## Calico Network Settings
```

```
## Note that enccw0.0.1000 in the example is the device name of master node. You need to
replace ens7 with the actual name of the primary network device on the x86 or Linux on Z
server
calico_ipip_enabled: true
calico_tunnel_mtu: 1350
calico_ip_autodetection_method: interface=eth0,eth1,enccw0.0.1000
```

Note: You need to update the password, IP addresses, and Calico Network interface to match your environment.

16. Run the following command to install ICP.

```
sudo docker run --net=host -t -e LICENSE=accept \
  -v "$(pwd)":/installer/cluster ibmcom/icp-inception-s390x:3.2.0-ee install
```

17. The installation takes some minutes (around 45 minutes) to complete. Your installation might require more or less time, depending on the number of nodes and configurations you selected. You receive the following output when the installation succeeds.

```
TASK [kubectl-config : include_tasks]
*********************************************************************************

TASK [k8s-resource : Finding all resource files]
*********************************************************************************
ok: [192.168.0.251 -> localhost]

TASK [k8s-resource : Creating Kubernetes resources]
*********************************************************************************

TASK [archive-addon : include_tasks]
*********************************************************************************

PLAY RECAP
*********************************************************************************
*****************************************************
192.168.0.251              : ok=166  changed=94   unreachable=0    failed=0
192.168.0.253              : ok=92   changed=44   unreachable=0    failed=0
192.168.0.254              : ok=85   changed=38   unreachable=0    failed=0
localhost                  : ok=373  changed=195  unreachable=0    failed=0


POST DEPLOY MESSAGE
*********************************************************************************

The Dashboard URL: https://9.16.27.19:8443, please use credentials in
config.yaml to login.

Playbook run took 0 days, 0 hours, 44 minutes, 37 seconds
```

Figure 3-16 Sample of the Post Deploy message

18. Also, you can issue **`kubectl get nodes`** command to get information about the cluster nodes as shown in Example 3-24.

Example 3-24 Output of kubectl get nodes command

```
[root@ssc4icp-master cluster]# kubectl get nodes
NAME            STATUS   ROLES                    AGE   VERSION
192.168.0.251   Ready    etcd,management,master   53m   v1.13.5+icp-ee
192.168.0.253   Ready    worker                   37m   v1.13.5+icp-ee
192.168.0.254   Ready    proxy                    37m   v1.13.5+icp-ee
```

The *192.168.0.253* and *192.168.0.254* entries are container instances that run in the SSC partition.

19. Access the dashboard URL to deploy your own blockchain workloads to the IBM Cloud Private cluster. See dashboard screen in Figure 3-17.

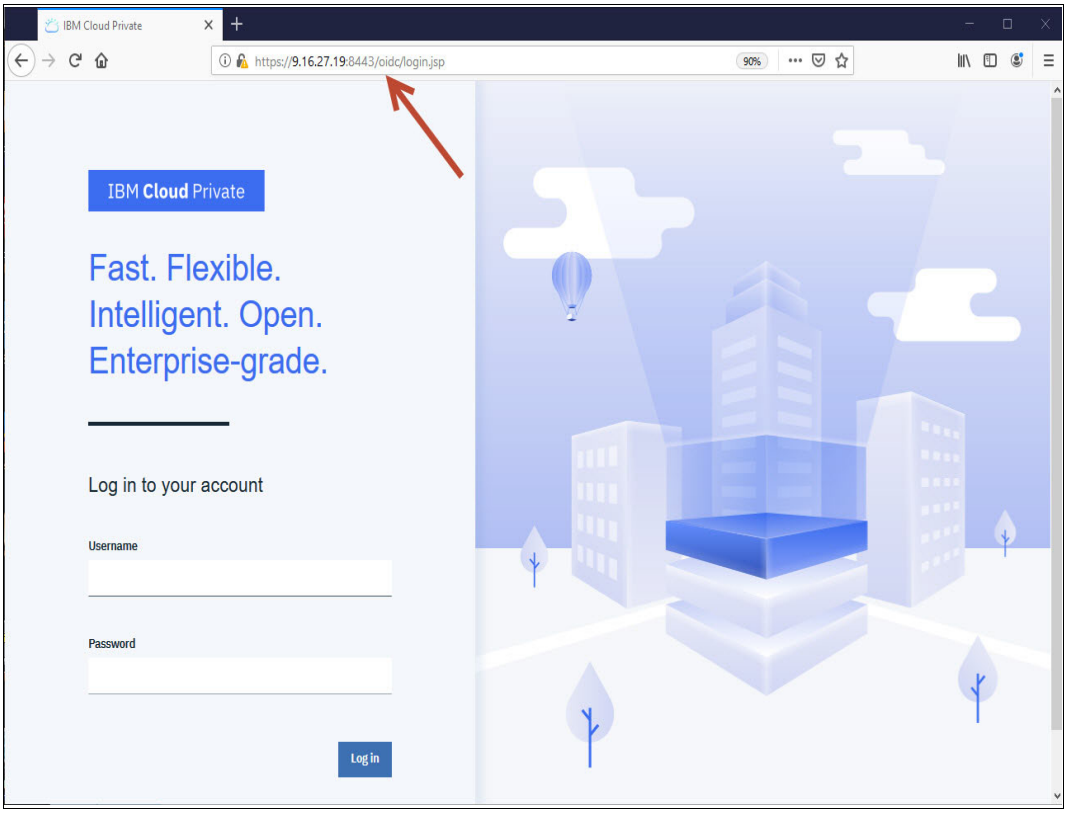


Figure 3-17 ICP dashboard

3.6.1 Deploying containerized applications

To deploy your own workloads to the IBM Cloud Private cluster on LinuxONE system, you need to install the bundled components into the IBM Cloud Private Catalog. Then, use Helm charts for your applications.

Install the bundled components into the IBM Cloud Private catalog

See *Installing bundled products* at https://www.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/installing/install_entitled_workloads.html for more details. Each of the following bundled components is supported by IBM Cloud Private on LinuxONE architecture, and is listed by its application category on the IBM Cloud Private Catalog.

- Data Services

- IBM Db2® Direct Advanced Edition 11.1 with Data Server Manager
- IBM Db2 Advanced Enterprise Server Ed. 11.1 with Data Server Manager
- IBM Db2 Warehouse Enterprise 2.0
- MongoDB
- PostgreSQL
- MariaDB
- Data Science and Business Analytics
- IBM Data Science Experience Local 1.1
- Toolchains & Runtimes
- IBM UrbanCode® Deploy
- Microclimate 18.03
- Jenkins
- IBM WebSphere® Liberty 17.0.0.4, 18.0.0.x
- IBM SDK for Node.js V6, V8
- Open Liberty
- Swift runtime
- Modernization Tools
- IBM Transformation Advisor 1.5.1
- Messaging
- IBM MQ Advanced 9.0 & v.next
- Rabbit MQ
- Digital Business Automation
- IBM Operational Decision Manager 8.9.2

Install your own components into the IBM Cloud Private Catalog by using Helm charts. For more information, see *IBM Cloud Private enablement guide for ISV and open source software:*
https://developer.ibm.com/linuxonpower/ibm-cloud-private-on-power/isv-guide/

3.7 Deploying GlusterFS on SSC ICP nodes

You can use GlusterFS storage either by deploying GlusterFS on your IBM Cloud Private cluster nodes, or by integrating a GlusterFS storage cluster that is deployed outside the IBM Cloud Private environment.

This section shows how to prepare dedicated nodes for a GlusterFS cluster in the Secure Service Container (SSC) for IBM Cloud Private environment. Then, you can configure GlusterFS during the IBM Cloud Private installation. For more information about how the GlusterFS works, see GlusterFS.
https://www.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/manage_cluster/glusterfs_land.html

Also, you have these configuration options for GlusterFS:

- Configure GlusterFS on worker nodes as described in *Configuring GlusterFS during IBM Cloud Private installation*:
  https://www.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/manage_cluster/configure_glusterfs_during.html
- Configure GlusterFS by using Helm charts or as an add-on service as in *Configuring GlusterFS after IBM Cloud Private installation*:
  https://www.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/manage_cluster/configure_glusterfs_after.html.

Note: You can get additional information about GlusterFS in “Persistent Storage providers” on page 23.

3.7.1 Preparing for deployment

Some or all of these steps might be required to prepare your environment for deployment of GlusterFS on SCC ICP nodes:

- “Adding disks through IBM Service Secure Container console”
- “Uninstalling ICP (only if SSC is installed or when an installation failed)” on page 107
- “Uninstalling the SSC containers (optional if not installed)” on page 108
- “Creating the containers in your SSC partition” on page 109

Adding disks through IBM Service Secure Container console

Before starting this procedure, we needed to add six additional disks through the IBM Service Secure Container console (**`https://9.16.27.18`**) to allow the configuration of the GlusterFS as described in this section.

On the Linux on Z master node server, update the **config/ssc4icp-config.yaml** file to configure the GlusterFS nodes on the Secure Service Container partition with the required CPU, memory, port range, and network specifications by using a specified template.

The following **ssc4icp-config.yaml** example file shows that template5 and template6 are defined for GlusterFS nodes in the DemoCluster with required resources. Template5 depicts the configuration of two storage quotagroups. Template6 depicts the configuration of single storage quotagroup. Ensure that the **ss4icp-config.yaml** file has the new settings.

Example 3-25 Sample of ss4icp-config.yaml file for GlusterFS

```
cluster:
  name: "DemoCluster"
  masterconfig:
    internal_ips: ['192.168.0.251']
    subnet: "192.168.0.0/24"

LPARS:
  - ipaddress: '9.16.27.18'
    containers:
      - template: "template1"
        count: 1
        internal_ips: ['192.168.0.253']

  - ipaddress: '9.16.27.18'
    containers:
      - template: "template2"
        count: 1
        internal_ips: ['192.168.0.254']
        proxy_external_ips: ['9.16.27.25']

  - ipaddress: '9.16.27.18'
    containers:
      - template: "template5"
        count: 1
        internal_ips: ['192.168.0.245']

  - ipaddress: '9.16.27.18'
    containers:
```

```
      - template: "template6"
        count: 2
        internal_ips: ['192.168.0.246','192.168.0.247']

template1:
 name: "worker1"
 type: "WORKER"
 # cpu defined by Number of threads
 cpu: "2"
 # Memory in MB
 memory: "4098"
 port_range: '15000'
 root_storage: "50G"
 icp_storage: "110G"
 internal_network:
   subnet: "192.168.0.0/24"
   gateway: "192.168.0.1"
   parent: "encb53"

template2:
 name: "proxy1"
 type: "PROXY"
 # cpu defined by Number of threads
 cpu: "2"
 # Memory in MB
 memory: "4096"
 port_range: '16000'
 root_storage: "50G"
 icp_storage: "110G"
 internal_network:
   subnet: "192.168.0.0/24"
   gateway: "192.168.0.1"
   parent: "encb53"
 proxy_external_network:
   subnet: "9.16.27.0/24"
   gateway: "9.16.27.1"
   parent: "vlan00b90.2309"

template5:
    name: "storage"
    type: "STORAGE"
    # Configure storage provisioner
    provisioner:
      - name: "glusterfs"
        size: "40G"
      - name: "glusterfs"
        size: "40G"
    # CPU defined by number of threads
    cpu: "4"
    # Memory in MB
    memory: "4098"
    port_range: '17000'
    root_storage: "50G"
    icp_storage: "110G"
    internal_network:
      subnet: "192.168.0.0/24"
      gateway: "192.168.0.1"
      parent: "encb53"
template6:
    name: "storage"
```

```
type: "STORAGE"
# Configure storage provisioner
provisioner:
  - name: "glusterfs"
    size: "80G"
# CPU defined by number of threads
cpu: "4"
# Memory in MB
memory: "4098"
port_range: '18000'
root_storage: "50G"
icp_storage: "110G"
internal_network:
  subnet: "192.168.0.0/24"
  gateway: "192.168.0.1"
  parent: "encb53"
```

Table 3-4 Additional parameter description for the ssc4icp-config.yaml file

Value	Description
internal_ips	Defines the IP addresses of GlusterFS node
provisioner	Indicates the template is used for the glusterfs node. The value of name under the provisioner section must be glusterfs.
size	Defines the disk size of glusterfs quotagroup
Note: Some parameters are detailed in the **ssc4icp-config.yaml** file on Table 3-2 on page 88.	

The previous sections instructed how to create the nodes without GlusterFS. This section details how to deploy the ICP with GlusterFS.

Consider a scenario where you have already deployed ICP and also the containers SSC containers for testing the deployment. In this case, you need to uninstall the ICP first and then uninstall the SCC containers. Otherwise, your installation might fail. (If it is a new installation, you can skip the following uninstallation steps.)

Uninstalling ICP (only if SSC is installed or when an installation failed)

1. Go to ICP installation folder (for example, **/opt/icp320/cluster/**)

Example 3-26 /opt/icp320/cluster/ folder

```
[root@ssc4icp-master /]# cd /opt/icp320/cluster/
[root@ssc4icp-master icp320]# pwd
/opt/icp320/cluster
[root@ssc4icp-master icp320]#
```

2. Issue the command that is listed in Example 3-27 to uninstalll ICP.

Example 3-27 Uninstall ICP command

```
sudo docker run --net=host -t -e LICENSE=accept \
  -v "$(pwd)":/installer/cluster ibmcom/icp-inception-s390x:3.2.0-ee uninstall
```

3. Wait for the command to complete, and confirm that you receive the following output.

```
TASK [uninstall : Restarting Containerd on cluster nodes]
*********************************************************************************
skipping: [192.168.0.245]
skipping: [192.168.0.246]
skipping: [192.168.0.247]
skipping: [192.168.0.251]
skipping: [192.168.0.253]
skipping: [192.168.0.254]

TASK [uninstall : Removing local files]
*********************************************************************************
changed: [192.168.0.245 -> localhost] => (item=/installer/cluster/cfc-certs)
ok: [192.168.0.245 -> localhost] => (item=/installer/cluster/cfc-keys)
changed: [192.168.0.245 -> localhost] => (item=/installer/cluster/cfc-components)
changed: [192.168.0.245 -> localhost] => (item=/installer/cluster/.install-3.2.0.lock)
ok: [192.168.0.245 -> localhost] => (item=/installer/cluster/conf)
changed: [192.168.0.245 -> localhost] => (item=/installer/cluster/.addon)
ok: [192.168.0.245 -> localhost] => (item=/installer/cluster/.misc)
ok: [192.168.0.245 -> localhost] =>
(item=/installer/cluster/misc/storage_class/none-sc.yaml)

PLAY RECAP
*********************************************************************************
192.168.0.245              : ok=20   changed=17   unreachable=0    failed=0
192.168.0.246              : ok=17   changed=14   unreachable=0    failed=0
192.168.0.247              : ok=17   changed=14   unreachable=0    failed=0
192.168.0.251              : ok=16   changed=13   unreachable=0    failed=0
192.168.0.253              : ok=17   changed=14   unreachable=0    failed=0
192.168.0.254              : ok=17   changed=14   unreachable=0    failed=0

Playbook run took 0 days, 0 hours, 26 minutes, 11 seconds
```

Figure 3-18 Output of the ICP uninstaller

Uninstalling the SSC containers (optional if not installed)

1. Go to SSC installation folder (for example, `/opt/blockchain/`). See Example 3-28.

Example 3-28 /opt/blockchain/ folder

```
[root@ssc4icp-master cluster]# cd /opt/blockchain/
[root@ssc4icp-master blockchain]# pwd
/opt/blockchain
[root@ssc4icp-master blockchain]#
```

2. Issue the command that is shown in Example 3-29.

Example 3-29 Uninstall command for the SSC containers

```
docker run --network=host --rm -it -v $(pwd)/config:/ssc4icp-cli-installer/config
ibmzcontainers/ssc4icp-cli-installer:1.1.0.3 uninstall
```

3. Wait for the command to complete, and confirm that your output is similar to this example:

```
Deleting cluster configuration
------------------------------------------------------------------------ 0.40s
cli-base : Delete Bridge Network
https://9.16.27.18/api/com.ibm.zBlockchain/networks/aaa_vlan00b90_2309_proxy_network
------------------------------------------------------------------------ 0.38s
cli-base : LPAR API token https://9.16.27.18/api/com.ibm.zaci.system/api-tokens
------------------------------------------------------------------------ 0.35s
Pre-UnInstallation Operations
------------------------------------------------------------------ 0.12s
Generate peer configuration
------------------------------------------------------------------ 0.11s
Update Peer configuration with master IP and IPSec shared key
------------------------------------------------------------------------ 0.10s
Set container configuration
------------------------------------------------------------------------ 0.10s
configuration : Checking container template
------------------------------------------------------------------------ 0.10s
Delete containers for Templates
------------------------------------------------------------------------ 0.09s
Delete containers for Templates
------------------------------------------------------------------------ 0.09s
```

Figure 3-19 Output of the SSC uninstaller

Creating the containers in your SSC partition

Note: Before you proceed with these steps, ensure that the internal IP address (**192.168.0.251**) is configured on the IBM Z master node server. You might want to use **ip addr show** command.

On the Linux on Z master node server, complete the following steps as a root user.

1. Run the command line tool to create the cluster nodes. Notice that the command must be run in the parent directory of the **config** directory, for example, `/opt/blockchain`. See Example 3-30.

Example 3-30 Installing the SSC containers

```
docker run --network=host --rm -it -v $(pwd)/config:/ssc4icp-cli-installer/config
ibmzcontainers/ssc4icp-cli-installer:1.1.0.3 install
```

2. Check the yaml files under `config/DemoCluster/`:

The following **cluster-configuration.yaml** example file is generated based on the cluster configuration that is specified in the **ssc4icp-config.yaml** file. Notice that new containers were created (**192.168.0.245**, **192.168.0.246**, and **192/168.0.247**).

Example 3-31 Sample of cluster-configuration.yaml file

```
LPARS:
- containers:
  - cpu: '2'
    icp_storage: 40000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.253
      parent: encb53
      subnet: 192.168.0.0/24
```

```
    memory: '4098'
    name: worker1-15001
    port: '15001'
    root_storage: 35000M
  ipaddress: 9.16.27.18
- containers:
  - cpu: '2'
    icp_storage: 40000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.254
      parent: encb53
      subnet: 192.168.0.0/24
    memory: '4096'
    name: proxy1-16001
    port: '16001'
    proxy_external_network:
      gateway: 9.16.27.1
      ip: 9.16.27.25
      parent: vlan00b90.2309
      subnet: 9.16.27.0/24
    root_storage: 35000M
  ipaddress: 9.16.27.18
- containers:
  - cpu: '4'
    icp_storage: 40000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.245
      parent: encb53
      subnet: 192.168.0.0/24
    memory: '4098'
    name: storage-17001
    port: '17001'
    provisioner:
    - name: glusterfs
      quotagroup: storage_17001_glusterfs1_qg
      size: 40000M
    - name: glusterfs
      quotagroup: storage_17001_glusterfs2_qg
      size: 40000M
    root_storage: 35000M
    storagenode: 'Yes'
  ipaddress: 9.16.27.18
- containers:
  - cpu: '4'
    icp_storage: 40000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.246
      parent: encb53
      subnet: 192.168.0.0/24
    memory: '4098'
    name: storage-18001
    port: '18001'
    provisioner:
    - name: glusterfs
      quotagroup: storage_18001_glusterfs1_qg
      size: 40000M
    root_storage: 35000M
```

```
    storagenode: 'Yes'
  - cpu: '4'
    icp_storage: 40000M
    internal_network:
      gateway: 192.168.0.1
      ip: 192.168.0.247
      parent: encb53
      subnet: 192.168.0.0/24
    memory: '4098'
    name: storage-18002
    port: '18002'
    provisioner:
    - name: glusterfs
      quotagroup: storage_18002_glusterfs1_qg
      size: 40000M
    root_storage: 35000M
    storagenode: 'Yes'
  ipaddress: 9.16.27.18
cluster:
  ipsecsecrets:
VqyLzJM829aY1Dpm3FA1iOJuH6h16CA8RksKx/nz7CCqmmSNdvreUoOzPbH632/ORkttTKI69oqbKEgOnBLTVg==
  masterconfig:
    internal_ips:
    - 192.168.0.251
    subnet: 192.168.0.0/24
  name: DemoCluster
  repoid: ICPIsolatedvm
```

Generation of the following **quotagroup-symlink.yaml** example file is based on the configuration of template5 and template6 in the **ssc4icp-config.yaml** file. In the example,

- two quotagroups **storage_17001_glusterfs1_qg** and **storage_17001_glusterfs1_qg** are attached to the GlusterFS node storage-17001
- **storage_18001_glusterfs1_qg** is attached to node storage-18001
- **storage_18002_glusterfs1_qg** is attached to node storage-18002

Example 3-32 Sample of quotagroup-symlink.yaml file

```
container = storage-17001, quotagroup = storage_17001_glusterfs1_qg,  symbolic_link =
/dev/disk/by-runq-id/storage_17001_glusterfs1_qg
container = storage-17001, quotagroup = storage_17001_glusterfs2_qg,  symbolic_link =
/dev/disk/by-runq-id/storage_17001_glusterfs2_qg
container = storage-18001, quotagroup = storage_18001_glusterfs1_qg,  symbolic_link =
/dev/disk/by-runq-id/storage_18001_glusterfs1_qg
container = storage-18002, quotagroup = storage_18002_glusterfs1_qg,  symbolic_link =
/dev/disk/by-runq-id/storage_18002_glusterfs1_qg
```

Make a note of these symlinks, because you are going to include them in the ICP **config.yaml** file during the ICP deployment later in this section.

Verify the cluster status by using the following instructions. The containers for IBM Cloud Private cluster are displayed and you can see the status of each container.

a. Create **get_containers.sh** file under your installation folder (`/opt/blockchain`) with the content that is listed in Example 3-33.

Example 3-33 Sample of get_containers.sh script

```
#!/bin/bash
# <emarins@br.ibm.com>
```

```
if [ "$1" != "" ]; then
   USERNAME=$1
else
   echo 'What is the SSC user admin name? (ie. admin)'
   read option
   USERNAME="$option"
fi

if [ "$2" != "" ]; then
        PASSWORD=$2
else
   echo 'What is the SSC admin password?'
   read -s option
   PASSWORD="$option"
   printf "%s\n" "${PASSWORD//?/*}"
fi

if [ "$3" != "" ]; then
        server=$3
else
   echo 'What is the SSC IP address or FQDN?'
   read option
   server="$option"
fi

# Getting Token
token=$(curl --request POST --url https://$server/api/com.ibm.zaci.system/api-tokens -H
'accept: application/vnd.ibm.zaci.payload+json' -H 'cache-control: no-cache' -H
'content-type: application/vnd.ibm.zaci.payload+json;version=1.0' -H 'zaci-api:
com.ibm.zaci.system/1.0' --insecure --data '{ "kind" : "request", "parameters" : {
"user" : "'$USERNAME'", "password" : "'$PASSWORD'" } }' 2>/dev/null | python -m
json.tool |grep token | awk -F\: '{print $2}'|tr -d \" |tr -d [:blank:])

if [ $(echo $token|wc -c) -eq 210 ]; then
   echo
   #echo "Token is valid"
else
   echo "Invalid Token. Exiting..."
   exit 1
fi

sleep 1s

output=$(curl --silent -k -X GET "https://$server/api/com.ibm.zaas/containers" -H
"accept: application/vnd.ibm.zaci.payload+json" -H "zACI-API: com.ibm.zaci.system/1.0"
-H "Content-Type: application/vnd.ibm.zaci.payload+json;version=1.0" -H "Authorization:
Bearer $token")

#echo $output | python -m json.tool

containers=$(echo $output | python -m json.tool |grep -A1 Names |grep \/|tr -d \" |tr -d
\/)
#echo $containers

echo "------------------------------------------------"
for container in $containers;
do
   echo "Container: $container"
   echo "------------------------------------------------"
```

```
   output=$(curl --silent -k -X GET
"https://$server/api/com.ibm.zaas/containers/$container" -H  "accept:
application/vnd.ibm.zaci.payload+json" -H  "Authorization: Bearer $token" -H  "zACI-API:
com.ibm.zaci.system/1.0" -H  "Content-Type:
application/vnd.ibm.zaci.payload+json;version=1.0")
   echo $output  | python -m json.tool | egrep "IPv4Address|Status" |grep -v
SecondaryIPAddresses|tr -d \" | sed -e 's/^[[:space:]]*//g' -e 's/[[:space:]]*\$//g' |tr
-d \,
   echo "-----------------------------------------------"
   echo
   sleep 1s
done
```

b. Run the following command to change permissions for this script so that users can execute it:
   chmod +x /opt/blockchain/get_containers.sh
c. Issue the script to get the status of the containers and IP address information.

   To execute this script, you need to provide information about the zAppliance user name, password, and its IP address. This script uses Secure Service Container system APIs to authenticate and get the information about the containers.

   Syntax of the script is `get_containers.sh` *<SSC_ADMIN> <SSC_PASSWORD> <SSC_IP>*. Typical output is shown in Example 3-34.

Example 3-34 Getting information about the containers

```
[root@ssc4icp-master blockchain]# /opt/blockchain/get_containers.sh admin password
9.16.27.18

-----------------------------------------------
Container: storage-17001
-----------------------------------------------
IPv4Address: 192.168.0.245
Status: running
Status: Up 13 hours
-----------------------------------------------

Container: storage-18002
-----------------------------------------------
IPv4Address: 192.168.0.247
Status: running
Status: Up 13 hours
-----------------------------------------------

Container: worker1-15001
-----------------------------------------------
IPv4Address: 192.168.0.253
Status: running
Status: Up 13 hours
-----------------------------------------------

Container: proxy1-16001
-----------------------------------------------
IPv4Address: 9.16.27.25
IPv4Address: 192.168.0.254
Status: running
Status: Up 13 hours
-----------------------------------------------

Container: storage-18001
```

```
-----------------------------------------------
IPv4Address: 192.168.0.246
Status: running
Status: Up 13 hours
-----------------------------------------------

[root@ssc4icp-master blockchain]#
```

Deploying ICP with GlusterFS

The deployment of ICP with GlusterFS is very similar of installing it without GlusterFS. You need to follow the steps listed in 3.6, “Deploying IBM Cloud Private” on page 98 and pay attention of the updates on the **config.yaml** file for GlusterFS. See the instructions here. If you have done it before, you can skip these steps.

1. Download the **strongswan** source package and build the binary on the LinuxONE system.

Example 3-35 Compiling and installing strongswan package on Redhat

```
# Run as root
yum install gmp-devel
wget http://download.strongswan.org/strongswan-5.6.2.tar.bz2
tar xjvf strongswan-5.6.2.tar.bz2
cd strongswan-5.6.2/
./configure --prefix=/usr --sysconfdir=/etc
make
make install
ipsec version
ipsec start
```

Note: You might experience network connectivity problems when you use strongswan 5.6.2 with Redhat 7 as the master node on IBM Z because of this known issue Using /32 groups in ipsec causing leaks https://access.redhat.com/solutions/4251881. To work around the problem, create a cron job to run ipsec restart command every 30 minutes on the master node.

For information on how to build the strongSwan package, see strongSwan Installation Documentation

2. Create the required installation folder:

```
mkdir /opt/icp320
```

3. Copy the *ibm-cloud-private-s390x-3.2.0.gz* image file to **/opt/icp320**
4. From the installation directory (**/opt/icp320**), load the IBM Cloud Private archive file as shown in Example 3-15.

Example 3-36 Loading Docker container images

```
[root@ssc4icp-master cluster]# tar xf ibm-cloud-private-s390x-3.2.0.tar.gz -O | sudo
docker load
Loaded image: ibmcom/icp-vip-manager-s390x:1.1
Loaded image: ibmcom/kafka-s390x:0.10.0.4
Loaded image: ibmcom/tiller-s390x:v2.12.3-icp-3.2.0
Loaded image: ibmcom/heketi-s390x:v8.0.0.1
```

Output snippet

```
Loaded image: ibmcom/istio-servicegraph-s390x:1.0.2
Loaded image: ibmcom/ma-file-wl-gen-s390x:3.2.0
Loaded image: ibmcom/metering-data-manager-s390x:3.2.0
Loaded image: ibmcom/compliance-annotator-s390x:3.2.0
Loaded image: ibmcom/cos-indexer-s390x:3.2.0
Loaded image: ibmcom/grafana-s390x:5.2.0-f3
Loaded image: ibmcom/kubectl-s390x:v1.13.5
[root@ssc4icp-master cluster]#
```

Note: If you find problems to install ICP, follow instructions on Installing an IBM Cloud Private Enterprise environment to get detailed information about the ICP installation.

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/installing/install_containers.html

5. Extract the configuration files from the installer image by using the command in this example:

   ```
   sudo docker run -v $(pwd):/data -e LICENSE=accept \
   ibmcom/icp-inception-s390x:3.2.0-ee \
   cp -r cluster /data
   ```

6. Confirm that the cluster folder was created by using **`ls cluster`** command.
7. Create the images directory under **`/opt/icp320/cluster`** folder.

   ```
   mkdir /opt/icp320/cluster/images
   ```

8. Move the **/opt/icp320/ibm-cloud-private-s390x-3.2.0.tar.gz** image file to the **`/opt/icp320/cluster/images`** directory.

   ```
   mv /opt/icp320/ibm-cloud-private-s390x-3.2.0.tar.gz /opt/icp320/cluster/images/
   ```

9. Copy the following two files into the **`/etc`** directory. Those two files are generated in the config/<*ClusterName*> directory after the Secure Service Container for IBM Cloud Private CLI tool is installed.
   - `config/<ClusterName>/ipsec.conf`
     This file contains the network topology of the cluster.
   - `config/<ClusterName>/ipsec.secret`
     This file contains a randomly generated Pre-Shared-Key (PSK) for use as an authorization token to the IPSec network.

Example 3-37 Copying the ipsec files to /etc/

```
cp -a /opt/blockchain/config/DemoCluster/ipsec* /etc/
```

10. Restart the **strongSwan** daemon to apply the changes by using **`systemctl restart strongswan`** command. Also, remember to make configuration persistent by issuing **`systemctl enable strongswan`**.

    Note: You might see the following message:
    "Warning: strongswan.service changed on disk. Run 'systemctl daemon-reload' to reload units."

    In this case, run **`systemctl daemon-reload`** and then **`systemctl restart strongswan`**.

11. Run **`ipsec stop`**, wait for 1 minute, and then run **`ipsec start.`**

12. Wait for about 5 minutes, and test the internal and external connection to each cluster node on the LinuxONE system by using the ping command as shown in Example 3-38.

Example 3-38 Verifying if IPs are reachable from master node

```
[root@ssc4icp-master ~]# ping -c2 192.168.0.245
PING 192.168.0.245 (192.168.0.245) 56(84) bytes of data.
64 bytes from 192.168.0.245: icmp_seq=1 ttl=64 time=0.241 ms
64 bytes from 192.168.0.245: icmp_seq=2 ttl=64 time=0.363 ms

--- 192.168.0.245 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1006ms
rtt min/avg/max/mdev = 0.241/0.302/0.363/0.061 ms

[root@ssc4icp-master ~]# ping -c2 192.168.0.246
PING 192.168.0.246 (192.168.0.246) 56(84) bytes of data.
64 bytes from 192.168.0.246: icmp_seq=1 ttl=64 time=0.371 ms
64 bytes from 192.168.0.246: icmp_seq=2 ttl=64 time=0.358 ms

--- 192.168.0.246 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.358/0.364/0.371/0.020 ms
[root@ssc4icp-master ~]# ping -c2 192.168.0.247
PING 192.168.0.247 (192.168.0.247) 56(84) bytes of data.
64 bytes from 192.168.0.247: icmp_seq=1 ttl=64 time=0.418 ms
64 bytes from 192.168.0.247: icmp_seq=2 ttl=64 time=0.335 ms

--- 192.168.0.247 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.335/0.376/0.418/0.045 ms
[root@ssc4icp-master ~]#

[root@ssc4icp-master ~]# ping -c2 192.168.0.253
PING 192.168.0.253 (192.168.0.253) 56(84) bytes of data.
64 bytes from 192.168.0.253: icmp_seq=1 ttl=64 time=0.388 ms
64 bytes from 192.168.0.253: icmp_seq=2 ttl=64 time=0.516 ms

--- 192.168.0.253 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.388/0.452/0.516/0.064 ms
[root@ssc4icp-master ~]# ping -c2 192.168.0.254
PING 192.168.0.254 (192.168.0.254) 56(84) bytes of data.
64 bytes from 192.168.0.254: icmp_seq=1 ttl=64 time=0.399 ms
64 bytes from 192.168.0.254: icmp_seq=2 ttl=64 time=0.430 ms

--- 192.168.0.254 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.399/0.414/0.430/0.025 ms
[root@ssc4icp-master ~]# ping -c2 9.16.27.25
PING 9.16.27.25 (9.16.27.25) 56(84) bytes of data.
64 bytes from 9.16.27.25: icmp_seq=1 ttl=64 time=1.08 ms
64 bytes from 9.16.27.25: icmp_seq=2 ttl=64 time=0.478 ms

--- 9.16.27.25 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.478/0.783/1.089/0.306 ms
[root@ssc4icp-master ~]#
```

13. Make sure all IP addresses are reachable you proceed.

14. You can use **ipsec status** command to verify the status of ipsec connections for the internal IP addresses (Example 3-39).

Example 3-39 Checking ipsec status

```
[root@ssc4icp-master ~]# ipsec status
Routed Connections:
main192.168.0.251{1}:  ROUTED, TRANSPORT, reqid 1
main192.168.0.251{1}:   192.168.0.0/24 === 192.168.0.0/24
Security Associations (5 up, 0 connecting):
main192.168.0.251[32]: ESTABLISHED 17 minutes ago,
192.168.0.251[192.168.0.251]...192.168.0.246[192.168.0.246]
main192.168.0.251{48}:  INSTALLED, TRANSPORT, reqid 1, ESP SPIs: c1c93ae1_i c91bbd64_o,
IPCOMP CPIs: d1e0_i 64e2_o
main192.168.0.251{48}:   192.168.0.251/32 === 192.168.0.246/32
main192.168.0.251[30]: ESTABLISHED 18 minutes ago,
192.168.0.251[192.168.0.251]...192.168.0.253[192.168.0.253]
main192.168.0.251{47}:  INSTALLED, TRANSPORT, reqid 1, ESP SPIs: c7b4d46e_i c0a66d49_o,
IPCOMP CPIs: e499_i 4959_o
main192.168.0.251{47}:   192.168.0.251/32 === 192.168.0.253/32
main192.168.0.251[28]: ESTABLISHED 19 minutes ago,
192.168.0.251[192.168.0.251]...192.168.0.245[192.168.0.245]
main192.168.0.251{46}:  INSTALLED, TRANSPORT, reqid 1, ESP SPIs: c91dcaef_i cec302f3_o,
IPCOMP CPIs: 55dd_i 2a2a_o
main192.168.0.251{46}:   192.168.0.251/32 === 192.168.0.245/32
main192.168.0.251[26]: ESTABLISHED 25 minutes ago,
192.168.0.251[192.168.0.251]...192.168.0.254[192.168.0.254]
main192.168.0.251{45}:  INSTALLED, TRANSPORT, reqid 1, ESP SPIs: c15dd72e_i cdfd5229_o,
IPCOMP CPIs: ba29_i 9765_o
main192.168.0.251{45}:   192.168.0.251/32 === 192.168.0.254/32
main192.168.0.251[25]: ESTABLISHED 26 minutes ago,
192.168.0.251[192.168.0.251]...192.168.0.247[192.168.0.247]
main192.168.0.251{44}:  INSTALLED, TRANSPORT, reqid 1, ESP SPIs: cdd8a738_i c58a9788_o,
IPCOMP CPIs: 5e6a_i ca09_o
main192.168.0.251{44}:   192.168.0.251/32 === 192.168.0.247/32
[root@ssc4icp-master ~]#
```

15. In the **/opt/icp320/cluster/config.yaml** file, add a 'nodes' section under 'storage-glusterfs' section to specify quotagroups for each GlusterFS node configured in the **ssc4icp-config.yaml** file.
    For the details of each GlusterFS device, refer to the **config/quotagroup-symlink.yaml** file information listed in Example 3-32 on page 111.

Example 3-40 Sample of new sections for config.yaml

```
no_taint_group: ["hostgroup-glusterfs"]

## GlusterFS Storage Settings
storage-glusterfs:
  nodes:
    - ip: 192.168.0.245
      devices:
        - /dev/disk/by-runq-id/storage_17001_glusterfs1_qg
        - /dev/disk/by-runq-id/storage_17001_glusterfs2_qg
    - ip: 192.168.0.246
      devices:
        - /dev/disk/by-runq-id/storage_18001_glusterfs1_qg
    - ip: 192.168.0.247
      devices:
        - /dev/disk/by-runq-id/storage_18002_glusterfs1_qg
  storageClass:
```

```
    create: true
    name: glusterfs
    isDefault: false
    volumeType: replicate:3
    reclaimPolicy: Delete
    volumeBindingMode: Immediate
    volumeNamePrefix: icp
    additionalProvisionerParams: {}
    allowVolumeExpansion: true
  gluster:
    resources:
      requests:
        cpu: 500m
        memory: 512Mi
      limits:
        cpu: 1000m
        memory: 1Gi
  heketi:
    backupDbSecret: heketi-db-backup
    authSecret: "heketi-secret"
    maxInFlightOperations: "20"
    dbSyncupDelay: "10"
    tls:
      generate: true
      issuer: "icp-ca-issuer"
      issuerKind: "ClusterIssuer"
      secretName: ""
    resources:
      requests:
        cpu: 500m
        memory: 512Mi
      limits:
        cpu: 1000m
        memory: 1Gi
  nodeSelector:
    key: hostgroup
    value: glusterfs
  prometheus:
    enabled: true
    path: "/metrics"
    port: 8080
  tolerations: []
  podPriorityClass: "system-cluster-critical"
```

16. Also, update **/opt/icp320/cluster/config.yaml** file with the modifications that are listed in Example 3-23.

Note: When you customize the IBM Cloud Private cluster, you must set the value of the **cluster_lb_address** parameter in the `<icp_installation_directory>/cluster/config.yaml` file to the public IP address of master node that will be accessed. And the **proxy_lb_address** parameter must be set to the public IP address of proxy node.

While you deploy ICP in your environment, ensure that the IP addresses, password, and devices are updated accordingly to fit your infrastructure.

Example 3-41 config.yaml modifications

```
## Remove the restriction for complex passwords
password_rules:
- '(.*)'

## Advanced Settings
default_admin_user: admin
default_admin_password: abc12345

## External loadbalancer IP or domain
## Or floating IP in OpenStack environment
# cluster_lb_address: none
cluster_lb_address: 9.16.27.19

## External loadbalancer IP or domain
## Or floating IP in OpenStack environment
# proxy_lb_address: none
proxy_lb_address: 9.16.27.25

## Install in firewall enabled mode
# firewall_enabled: false
firewall_enabled: false

## Calico Network Settings
## Note that enccw0.0.1000 in the example is the device name of master node. You need to replace
ens7 with the actual name of the primary network device on the x86 or Linux on Z server
calico_ipip_enabled: true
calico_tunnel_mtu: 1350
calico_ip_autodetection_method: interface=eth0,eth1,enccw0.0.1000

## Note that these following settings already exist on the config.yaml file.
## Ensure you do not add all parameters below. Intead of, update storage-glusterfs
## from disabled to enabled.
management_services:
  istio: disabled
  vulnerability-advisor: disabled
  storage-glusterfs: enabled
  #storage-glusterfs: disabled
  storage-minio: disabled
  platform-security-netpols: disabled
  node-problem-detector-draino: disabled
  multicluster-endpoint: disabled
```

17. In the hosts file (`/opt/icp320/cluster/hosts`), add **hostgroup-glusterfs** section with the IP address of each GlusterFS node. Also, ensure that the master, storage, worker, and proxy nodes have the internal IP addresses. See Example 3-42.

Example 3-42 Sample of hostgroup-glusterfs section

```
[master]
192.168.0.251

[worker]
192.168.0.253

[proxy]
192.168.0.254

[hostgroup-glusterfs]
192.168.0.245
```

```
192.168.0.246
192.168.0.247

#[management]
#4.4.4.4

#[va]
#5.5.5.5
```

18. Update `/etc/hosts` file on Linux on Z master node server to include all nodes. See Example 3-43.

Example 3-43 Sample of /etc/hosts file

```
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

#For ICP and SSC
192.168.0.251 ssc4icp-master
192.168.0.253 worker1-15001
192.168.0.254 proxy1-16001
192.168.0.245 storage-17001
192.168.0.246 storage-18001
192.169.0.247 storage-18002
```

19. Copy `config/DemoCluster/ssh_key*` files to `/opt/icp320/cluster` folder to allow the master node to access the SSC container images that was created previously.

    cp -a /opt/blockchain/config/DemoCluster/ssh_key* /opt/icp320/cluster/

20. Upload `/etc/hosts` files from master to all nodes by using the scp commands that are listed in Example 3-44.

Example 3-44 Uploading /etc/hosts files from master to storage, worker and proxy nodes.

```
[root@ssc4icp-master ~]# scp -i /opt/icp320/cluster/ssh_key /etc/hosts
root@192.168.0.245:/etc/
hosts
100%  497    51.9KB/s   00:00
[root@ssc4icp-master ~]# scp -i /opt/icp320/cluster/ssh_key /etc/hosts
root@192.168.0.246:/etc/
hosts
100%  497    68.2KB/s   00:00
[root@ssc4icp-master ~]# scp -i /opt/icp320/cluster/ssh_key /etc/hosts
root@192.168.0.247:/etc/
hosts
100%  497   204.2KB/s   00:00
[root@ssc4icp-master ~]# scp -i /opt/icp320/cluster/ssh_key /etc/hosts
root@192.168.0.253:/etc/
hosts
100%  497   204.0KB/s   00:00
root@ssc4icp-master ~]# scp -i /opt/icp320/cluster/ssh_key /etc/hosts
root@192.168.0.254:/etc/
hosts
100%  497   155.0KB/s   00:00
```

Now, ICP deployment is complete. You complete the deployment in the following steps.

21. Ensure the new **ssh_key** can be used to access the Linux master node (**192.168.0.251**). Run the command in Example 3-45 to authorize logins with the SSH keys.

Example 3-45 Authorizing the use of SSH_keys on Linux Master node

```
[root@ssc4icp-master cluster]# ssh-copy-id -i
/opt/blockchain/config/DemoCluster/ssh_key.pub root@192.168.0.251
/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/opt/blockchain/config/DemoCluster/ssh_key.pub"
The authenticity of host '192.168.0.251 (192.168.0.251)' can't be established.
ECDSA key fingerprint is SHA256:hhlO+P8k6LaZ6EMRG5vwy/Ea9wGcZLJRtoi5r9hzu7Q.
ECDSA key fingerprint is MD5:75:3a:c2:12:39:ca:23:f7:5b:58:03:25:54:99:2a:13.
Are you sure you want to continue connecting (yes/no)? yes
/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
to install the new keys

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@192.168.0.251'"
and check to make sure that only the key(s) you wanted were added.
```

22. Update and confirm that the **/opt/icp320/cluster/hosts** file contains the internal IPs for the master, storage, worker, and proxy nodes as shown in Example 3-22.

Example 3-46 Sample of /opt/icp320/cluster/hosts file

```
[master]
192.168.0.251

[worker]
192.168.0.253

[proxy]
192.168.0.254

[hostgroup-glusterfs]
192.168.0.245
192.168.0.246
192.168.0.247

#[management]
#4.4.4.4

#[va]
#5.5.5.5
```

23. Run the following command to install IBM Cloud Private:

```
cd /opt/icp320/cluster && sudo docker run --net=host -t -e LICENSE=accept \
  -v "$(pwd)":/installer/cluster ibmcom/icp-inception-s390x:3.2.0-ee install
```

24. Wait for approximately 45 minutes or more for completion. Remember that the duration depends on the number of nodes and settings that your installation requires. Therefore, more or less time might be required. After successful completion, you see output that is similar to Figure 3-20 on page 122.

```
TASK [kubectl-config : include_tasks]
*********************************************************************************

TASK [k8s-resource : Finding all resource files]
*********************************************************************************
ok: [192.168.0.251 -> localhost]

TASK [k8s-resource : Creating Kubernetes resources]
*********************************************************************************

TASK [archive-addon : include_tasks]
*********************************************************************************

PLAY RECAP
*********************************************************************************
192.168.0.245              : ok=110  changed=62   unreachable=0    failed=0
192.168.0.246              : ok=103  changed=56   unreachable=0    failed=0
192.168.0.247              : ok=103  changed=56   unreachable=0    failed=0
192.168.0.251              : ok=188  changed=113  unreachable=0    failed=0
192.168.0.253              : ok=104  changed=56   unreachable=0    failed=0
192.168.0.254              : ok=104  changed=56   unreachable=0    failed=0
localhost                  : ok=383  changed=201  unreachable=0    failed=0

POST DEPLOY MESSAGE
*********************************************************************************

The Dashboard URL: https://9.16.27.19:8443, please use credentials in config.yaml to
login.

Playbook run took 0 days, 1 hours, 14 minutes, 20 seconds
```

Figure 3-20 Output of a successful ICP installation

25. Confirm that the cluster is up.
    a. Check the IBM Cloud Private console at this URL:
       https://*<cluster_lb_address>*:8443.
       In our example, this value is **https://9.16.27.19:8443**.
    b. Run the following command to check the GlusterFS nodes:
       kubectl get po -n kube-system | grep gluster.
       Example 3-47 show typical output.

Example 3-47 Getting information about the gluster resources

```
[root@ssc4icp-master blockchain]# kubectl get po -n kube-system | grep gluster
storage-glusterfs-glusterfs-daemonset-6lfq7                      1/1     Running
0          18h
storage-glusterfs-glusterfs-daemonset-jc2w5                      1/1     Running
0          18h
storage-glusterfs-glusterfs-daemonset-xxkv8                      1/1     Running
0          18h
storage-glusterfs-glusterfs-heketi-deployment-7755689cf4-8r6zd   1/1     Running
0          18h
storage-glusterfs-glusterfs-heketicert-job-9hjkm                 0/1     Completed
0          18h
[root@ssc4icp-master blockchain]#
```

26. To check the **storageclass** component, run this command:
    kubectl get storageclass -n kube-system
    Example 3-48 shows sample output for this command.

Example 3-48 Getting storageclass resources

```
[root@ssc4icp-master blockchain]# kubectl get storageclass -n kube-system
NAME                        PROVISIONER                    AGE
glusterfs                   kubernetes.io/glusterfs        18h
image-manager-storage       kubernetes.io/no-provisioner   19h
logging-storage-datanode    kubernetes.io/no-provisioner   18h
mongodb-storage             kubernetes.io/no-provisioner   18h
[root@ssc4icp-master blockchain]#
```

27. Run the command in Example 3-49 to see information about the cluster nodes.

Example 3-49 Getting cluster nodes information

```
[root@ssc4icp-master ~]# kubectl get nodes
NAME            STATUS   ROLES                      AGE   VERSION
192.168.0.245   Ready    glusterfs                  21h   v1.13.5+icp-ee
192.168.0.246   Ready    glusterfs                  21h   v1.13.5+icp-ee
192.168.0.247   Ready    glusterfs                  21h   v1.13.5+icp-ee
192.168.0.251   Ready    etcd,management,master     22h   v1.13.5+icp-ee
192.168.0.253   Ready    worker                     21h   v1.13.5+icp-ee
192.168.0.254   Ready    proxy                      21h   v1.13.5+icp-ee
[root@ssc4icp-master ~]#
```

Now, you are ready to deploy blockchain through ICP.

3.8 Uninstalling ICP and SSC

This section describes how to uninstall ICP and SSC in case you need to perform this activity.

3.8.1 Uninstalling SSC for IBM Cloud Private

Uinstall the IBM Cloud Private as follows:

1. Log in as a root user.
2. Run the ***docker save*** command to back up all the images for your workloads.
3. Uninstall the IBM Cloud Private runtime environment by following the instructions in one of these resources:

- *Uninstalling IBM Cloud Private Community Edition* https://www.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/installing/uninstall_ce.html or
- *Uninstalling IBM Cloud Private Enterprise Edition* https://www.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/installing/uninstall.html.

3.8.2 Uninstalling the Secure Service Container for IBM Cloud Private CLI tool

Uninstall the Secure Service Container for IBM Cloud Private CLI from the x86 or Linux on Z master node server as follows:

1. Log in as a root user
2. Run the following command under the **config** folder to delete all the nodes:

```
docker run --network=host --rm -it -v \
$(pwd)/config:/ssc4icp-cli-installer/config \
ibmzcontainers/ssc4icp-cli-installer:1.1.0.3 uninstall
```

The uninstallation references the configuration details in the **ssc4icp-config.yaml** file to delete all the cluster nodes and configurations that are created on the Secure Service Container partitions. The command also removes the SSH key files from your Secure Service Container for IBM Cloud Private **config** directory.

3. Run one of the following commands to perform the unregistration process on the Secure Service Container partitions.
   - Perform the basic unregistration process:
     ```
     docker run --network=host --rm -it -v \
     $(pwd)/config:/ssc4icp-cli-installer/config \
     ibmzcontainers/ssc4icp-cli-installer:1.1.0.3 cleanup
     ```
   - Perform the unregistration process, with removal of active nodes based on the use of the `--force` option:
     ```
     docker run --network=host --rm -it -v \
     $(pwd)/config:/ssc4icp-cli-installer/config \
     ibmzcontainers/ssc4icp-cli-installer:1.1.0.3 cleanup --force
     ```

   Note: The data pool of the active nodes is not deleted after a forced cleanup.

4. (*Optional*) Delete the SSH key pair that was generated for the cluster, and remove the associated SSH access from your master node. If you do not delete the SSH key pair, the existing SSH keys are used if you install the IBM Cloud Private again with the same cluster installation directory.
   a. Run the following command to find the public SSH key in your **authorized_keys** file, including the line number.
      You must modify the path of **~/config/ssh_key.pub** file to match the public key location inside your Secure Service Container for IBM Cloud Private **config** directory when you run the command.
      ```
      grep -n "$(cat ~/config/ssh_key.pub)" /root/.ssh/authorized_keys
      ```
   b. Use a text editor such as vi to remove the line that you see here from your **authorized_keys** file:
      ```
      vi /root/.ssh/authorized_keys
      ```
   c. Run the following command to remove the SSH private key from the IBM Cloud Private installation directory.
      You must modify the path of **~/cluster/ssh_key** file to match the SSH private key location in your IBM Cloud Private installation directory when you run the command.
      ```
      rm -rf ~/cluster/ssh_key
      ```

3.8.3 Uninstalling Secure Service Container partitions

You can stop, deactivate, or delete the Secure Service Container partitions on the IBM Z or LinuxONE machine. These steps refer to chapters in the *IBM Z Secure Service Container User's Guide:*
https://www-01.ibm.com/support/docview.wss?uid=isg26ab9a15cbe12a81d85258194006e4a89.

1. (*Optional*) Export the Secure Service Container configuration as described in the *Exporting or importing appliance configuration data* section of Chapter 14, "Using the Secure Service Container user interface."
2. Stop/deactivate or delete the Secure Service Container partition:
   - On a standard mode system, Chapter 7, “Deactivating or deleting a Secure Service Container partition on a standard mode system.”
   - On a DPM-enabled system, Chapter 12, “Stopping or deleting a Secure Service Container partition on a DPM-enabled system.”

3.9 Updating the cluster resources dynamically

IBM Cloud Private allows you to change the CPU or memory resources on the cluster nodes while the cluster is still running. To do this on the x86 or Linux on Z server, you complete the following steps as the root user.

1. Run the following command to identify the node IP addresses for which you want to update the resources. (You could also get the information from the **ssc4icp-config.yaml** file by checking the node IP addresses that use the template to be updated.)

   ```
   kubectl get nodes
   ```

2. Configure each of those to-be-updated cluster nodes to the maintenance mode by following the instructions on Node maintenance.

   ```
   kubectl cordon <node_IP_address>
   kubectl drain <node_IP_address> --grace-period=300 --ignore-daemonsets=true
   ```

3. Update the **config/ssc4icp-config-update.yaml** file to include the nodes to be updated and resource settings that you want to apply.
   For example, to change the CPU number to 5 for a worker node **192.168.0.252**, the **ssc4icp-config-update.yaml** needs to specify only the following values:

   – The configuration that is related to this node **192.168.0.252**, such as LPAR configuration.
   – The template for this worker node, **template1** in this example.

   ```
   cluster:
    name: "temp"
    datapool: "exists"
    masterconfig:
     internal_ips: ['192.168.0.251']
     subnet: "192.168.0.0/24"
   LPARS:
   -ipaddress: '10.152.151.105'
     containers:
      -template: "template1"
       count: 1
       internal_ips: ['192.168.0.252']
   template1:
     name: "worker"
     type: "WORKER"
     cpu: "5"
     memory: "4098"
     port_range: '15000'
     root_storage: "60G"
     icp_storage: "140G"
     internal_network:
       subnet: "192.168.0.0/24"
       gateway: "192.168.0.1"
       parent: "encf700"
   ```

Note: Only the changes to the CPU and Memory settings are applied. Any other changes in the **ssc4icp-config-update.yaml** file are ignored. Ensure that the total CPU or memory of your cluster does not exceed the assigned resources on the Secure Service Container partitions.

   – The cluster name in the **ssc4icp-config-update.yaml** file must be **temp**.
   – The value of **count** setting under the container configuration is **1** because only one worker node will be updated.

For more information of the existing cluster settings in the **ssc4icp-config.yaml** file, see “Configuring the cluster resources” on page 86.

4. Run the following command to apply the changes to the cluster node.

```
docker run --rm -it --net host -v $(pwd)/config:/ssc4icp-cli-installer/config
ibmzcontainers/ssc4icp-cli-installer:1.1.0.3 update
```

Note: The **cluster-configuration.yaml** file of the cluster will be refreshed with the new configuration. For more information of the **cluster-configuration.yaml** file, see “Creating the cluster nodes” on page 89.

5. After the command completes, the updated cluster nodes are in the **NotReady** state. Run the following commands to reactivate those cluster nodes.

   Remove those cluster nodes by following the instructions on *Removing an IBM Cloud Private cluster node*:
   https://www-03preprod.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/installing/remove_node.html

   For example, remove the updated worker node **192.168.0.252** from the existing cluster by running the following command on the Linux on Z server.

```
docker run -e LICENSE=accept --net=host -t -e LICENSE=accept -v
"$(pwd)":/installer/cluster ibmcom/icp-inception-s390x:3.2.0-ee uninstall -l
192.168.0.252
```

6. Configure the **/cluster/hosts** file under the IBM Cloud Private installation directory to include the cluster node information. For example, for a cluster with two worker nodes and one proxy node, the **cluster/hosts** file resembles the following settings.

```
[master]
<master_node_IP_address> ansible_user="root"
[worker]
<worker_node_1_IP_address> ansible_user="root"
<worker_node_2_IP_address> ansible_user="root"
[proxy]
<proxy_node_IP_address> ansible_user="root"
```

7. Log in to the updated cluster nodes by using the **ssh** utility, and configure the **/etc/hosts** file by adding the IP address and node name of each cluster node. You can get the IP address and node name information from the **cluster-configuration.yaml** file. For example, the **/etc/hosts** file for a cluster with two worker nodes and one proxy node resembles the following settings.

```
...
<master_node_IP_address> <master_node_host_name>
<worker_node_1_IP_address> <worker_node_1_host_name>
<worker_node_2_IP_address> <worker_node_2_IP_host_name>
<proxy_node_IP_address> <proxy_node_host_name>
```

8. Add those cluster nodes back into the cluster by following the instructions on *Adding an IBM Cloud Private cluster node*:
   https://www-03preprod.ibm.com/support/knowledgecenter/SSBS6K_3.2.0/installing/add_node.html

9. For example, add the updated worker node **192.168.0.252** back into the existing cluster by running the following command on the Linux on Z server.

```
docker run -e LICENSE=accept --net=host -t -e LICENSE=accept -v
"$(pwd)":/installer/cluster ibmcom/icp-inception-s390x:3.2.0-ee worker -l 192.168.0.252
```

4

IBM Blockchain Platform installation and configuration

In this chapter, we describe the installation and configuration process of IBM Blockchain Platform (IBP), including the IBP Console.

This chapter includes the following topics:

4.1 Console installation

To install the IBM Blockchain Platform for Multicloud Console, a Kubernetes administrator must complete the general process that is described in this section:

1. “Loading Helm chart” on page 128
2. “Setting up role-based access control (RBAC) roles for blockchain [1x per cluster only]” on page 131
3. Console installation:
   – “Scripted console installation” on page 136 or
   – “Manual console installation” on page 143

Before beginning this section, you must have configured an IBM Cloud Private cluster and created a storage class for use with the blockchain platform such as local storage, nfs, or glusterfs.

You can deploy only one console per namespace. If you need to create multiple blockchain networks (for example development, staging, and production), you must create a unique namespace for each environment. You can do this by setting a higher *team_number* in the procedure “Scripted console installation” on page 136 or manually by repeating the steps in “Manual console installation” on page 143.

Note: To complete console installation, the Kubernetes administrator must have cluster administrator access and cloudctl, helm (client), and kubectl must be installed. If the administrator is not cluster administrator, they must get this access or must ask a cluster administrator to perform these steps.

If you need to install cloudctl, helm, or kubectl,

- cloudctl instructions are here:
  https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/manage_cluster/install_cli.html
- helm (client) instructions are here:
  https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/app_center/create_helm_cli.html
- kubectl instructions are here:
  https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/manage_cluster/install_kubectl.html

4.1.1 Loading Helm chart

After you set up the security policy, download the Helm chart of IBM Blockchain Platform for IBM Cloud Private from Passport Advantage Online. Then, follow these steps for installation:

1. Log in to your IBM Cloud Private cluster to any namespace:

   ```
   cloudctl login -a https://<cluster_CA_domain>:8443 --skip-ssl-validation
   ```

2. Create a namespace for your Helm chart:

   ```
   kubectl create ns <namespace_name>
   ```

3. Set this namespace as your current namespace:

   ```
   kubectl config set-context --current --namespace=<namespace>
   ```

4. Ensure that the Docker CLI is configured. After you configure the Docker CLI, access the image registry on your cluster by using the following command:
   `docker login <cluster_CA_domain>:8500`
5. List current helm repositories:
   `cloudctl catalog repos`
6. Import the Helm chart by using the command line. Access the directory where you stored the downloaded Helm chart package from PPA. Then, run the following command in the IBM Cloud Private CLI to import the Helm chart into your IBM Cloud Private cluster:
   `cloudctl catalog load-archive --archive <archive-name> --registry <cluster_CA_domain>:8500/<namespace> --repo <repo-name>`

   Note: If you are using macOS, add `--username value` and `--password value`, or set in the environment as DOCKER_USER and DOCKER_PASSWORD.

7. Replace the following values:
   - `<archive-name>` with the name of the downloaded.`tar.gz` file.
   - <*`cluster_CA_domain`*>`:8500` with the domain that you use to log in to your IBM Cloud Private cluster.
   - <*namespace*> with the namespace where you plan to upload your chart.
   - `<repo-name>` with the Helm repository where you want to upload the chart.

   Note: You can upload the chart to your local repo (local-charts) by first running `cloudctl catalog repos` and selecting one of your existing repos such as local-charts and uploading there with `--repo local-charts`. Choose this repository when you look for the chart in the catalog to install the Helm chart.

   When this command completes successfully, you can see something that is similar to the following information:

Example 4-1 Command successful

```
Uploading Helm chart(s)
Processing chart: charts/ibm-blockchain-platform-prod-1.1.0.tgz
Updating chart values.yaml
Uploading chart
Loaded Helm chart
OK

Synch charts
Archive finished processing
```

8. Click **Catalog** in the IBM Cloud Private console, and then click **Blockchain** in the left navigation panel. If the import was successful, the `ibm-blockchain-platform-prod` tile is displayed on the IBM Cloud Private Catalog page.

Example 4-2 shows an example.

Example 4-2 Loading Helm chart to cluster

```
cloudctl login -a https://9.56.28.25:8443 --skip-ssl-validation -n default

Username> admin

Password>
Authenticating...
```

```
OK

Targeted account mycluster Account (id-mycluster-account)

Targeted namespace default

Configuring kubectl ...
Property "clusters.mycluster" unset.
Property "users.mycluster-user" unset.
Property "contexts.mycluster-context" unset.
Cluster "mycluster" set.
User "mycluster-user" set.
Context "mycluster-context" created.
Switched to context "mycluster-context".
OK

Configuring helm: /Users/garrettwoodworth/.helm
OK
```

kubectl create ns blockchain-time

```
namespace/blockchain-time created
```

kubectl config set-context --current --namespace=blockchain-time

```
Context "mycluster-context" modified.
```

docker login mycluster.icp:8500

```
Username: admin
Password:
Login Succeeded
```

cloudctl catalog repos

```
Name                   URL
Local
ibm-charts            https://raw.githubusercontent.com/IBM/charts/master/repo/stable/
false
local-charts           https://mycluster.icp:8443/helm-repo/charts
true
mgmt-charts            https://mycluster.icp:8443/mgmt-repo/charts
true
ibm-charts-public      https://registry.bluemix.net/helm/ibm/
false
ibm-community-charts  https://raw.githubusercontent.com/IBM/charts/master/repo/community/
false
ibm-entitled-charts   https://raw.githubusercontent.com/IBM/charts/master/repo/entitled/
false
```

cloudctl catalog load-archive --archive IBM_BLOCKCHAIN_PLATFORM_FOR_IBM_C.tar.gz --registry mycluster.icp:8500/blockchain-time --repo local-charts --username admin --password mypassword

> **Tip: --username** and **--password** are needed for MacOS only. These values are not needed on Linux.

```
Expanding archive
OK

Importing docker image(s)
  Processing image: op-tools/op-tools:2.0.0-amd64
    Loading Image
```

```
    Tagging Image
    Pushing image as: mycluster.icp:8500/blockchain-time/op-tools/op-tools:2.0.0-amd64
  Processing image: op-tools/op-tools:2.0.0-s390x
    Tagging Image
    Pushing image as:
```

.....<**Output truncated for brevity>**

```
mycluster.icp:8500/blockchain-time/fluentd:v1.4-2-amd64
    Annotating manifest list: mycluster.icp:8500/blockchain-time/fluentd:v1.4-2-s390x
    Pushing manifest list: mycluster.icp:8500/blockchain-time/fluentd:v1.4-2
Digest: sha256:5f15171e80504fb6d5811885eeee42c6ce034852c470cff8bf25b5bb8e5dcc4d 743
OK

Uploading Helm chart(s)
  Processing chart: charts/ibm-blockchain-platform-prod-2.0.0.tgz
  Updating chart values.yaml
  Uploading chart
Loaded Helm chart
OK

Synch charts
OK

Archive finished processing
```

4.1.2 Setting up role-based access control (RBAC) roles for blockchain [1x per cluster only]

Note: This setup needs to be done only once per cluster. After the initial creation, the cluster administrator merely creates more rolebindings to the cluster roles that are created in this section. (The administrator can use either the script or manual methods.)

Kubernetes grants roles that give users or service accounts the ability to perform a set of specified actions against specified resources. Running applications have the roles that are bound to their service account. The IBM Blockchain Platform for Multicloud Console requires specific roles at both the namespace and cluster level to control the deployment and monitoring of your blockchain networks.

For the maximum amount of security, using least-privilege is important (in other words, giving as little access as necessary). This means defining *rolebindings* rather than *clusterrolebindings* wherever possible to get these advantages:

- You limit the privileges that are granted to the namespace scope.
- You always grant privileges to a non-default service account in a non-default namespace so that only pods explicitly designated to gain these extra privileges will receive them.
- Other users cannot claim cluster-admin-like privileges for themselves.

The IBM Blockchain Platform Helm chart requires that the cluster administrator bind specific security and access policies to the target namespace before installation. We provide YAML files that define the policies in the following steps. Save these files to your local system, and then bind them your namespace using the IBM Cloud Private CLI. Follow the steps below before deploying the IBM Blockchain Platform Helm chart.

You have two ways to apply these policies:

- Clone a github repository and apply the files from there. OR

- Copy and paste the files directly onto your local machine and apply the files from there.

Scripted: Files on GitHub

Perform the following steps to for scripted installation:

1. Log in to your IBM Cloud Private cluster and select the target namespace of your deployment.

   ```
   cloudctl login -a https://mycluster.icp:8443 --skip-ssl-validation -n
   default
   ```

2. Clone the script to the computer that you use to interact with your cluster. Use the command line tools (kubectl and cloudctl), and move the cluster into its main directory and apply all cluster roles and pod security policy to your cluster.

   ```
   git clone --recursive -j8
   https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform
   -for-Multicloud.git ibpbook-scripts && cd
   ibpbook-scripts/IBM_Blockchain_Platform_for_Multicloud_Automation && kubectl apply -f .
   ```

   Example 4-3 shows an example GitHub session.

Example 4-3 Apply all cluster roles and pod security policy to your cluster

```
cloudctl login -a https://mycluster.icp:8443 --skip-ssl-validation -n default

   Username> admin

   Password>
   Authenticating...
   OK

   Targeted account mycluster Account (id-mycluster-account)

   Targeted namespace default

   Configuring kubectl ...
   Property "clusters.mycluster" unset.
   Property "users.mycluster-user" unset.
   Property "contexts.mycluster-context" unset.
   Cluster "mycluster" set.
   User "mycluster-user" set.
   Context "mycluster-context" created.
   Switched to context "mycluster-context".
   OK

   Configuring helm: /home/support/.helm
   OK

   git clone --recursive -j8
   https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform
   -for-Multicloud.git ibpbook-scripts && cd
   ibpbook-scripts/IBM_Blockchain_Platform_for_Multicloud_Automation && kubectl apply -f .
   Cloning into 'ibpbook-scripts'...
   remote: Enumerating objects: 6, done.
   remote: Counting objects: 100% (6/6), done.
   remote: Compressing objects: 100% (5/5), done.
   remote: Total 6 (delta 1), reused 6 (delta 1), pack-reused 0
   Unpacking objects: 100% (6/6), done.
   Submodule 'IBM_Blockchain_Platform_for_Multicloud_Automation'
   (https://github.com/siler23/IBM_Blockchain_Platform_for_Multicloud_Automation)
   registered for path 'IBM_Blockchain_Platform_for_Multicloud_Automation'
```

```
Submodule 'nfs-helm-dynamic-multiarch'
(https://github.com/siler23/nfs-helm-dynamic-multiarch) registered for path
'nfs-helm-dynamic-multiarch'
Cloning into
'/Users/garrettwoodworth/Documents/Redbook_IBP4Multicloud/ibpbook-scripts/IBM_Blockchain
_Platform_for_Multicloud_Automation'...
remote: Enumerating objects: 46, done.
remote: Counting objects: 100% (46/46), done.
remote: Compressing objects: 100% (33/33), done.
remote: Total 46 (delta 24), reused 34 (delta 12), pack-reused 0
Cloning into
'/Users/garrettwoodworth/Documents/Redbook_IBP4Multicloud/ibpbook-scripts/nfs-helm-dynam
ic-multiarch'...
remote: Enumerating objects: 122, done.
remote: Counting objects: 100% (122/122), done.
remote: Compressing objects: 100% (70/70), done.
remote: Total 122 (delta 53), reused 105 (delta 36), pack-reused 0
Receiving objects: 100% (122/122), 180.74 KiB | 3.23 MiB/s, done.
Resolving deltas: 100% (53/53), done.
Submodule path 'IBM_Blockchain_Platform_for_Multicloud_Automation': checked out
'd04f7d3f9cd911179bb66d75cd1426a08140c800'
Submodule path 'nfs-helm-dynamic-multiarch': checked out
'c879b3d17c3e8ef105e6be9818365c25c6e23215'
clusterrole.rbac.authorization.k8s.io/crd-clusterrole created
clusterrole.rbac.authorization.k8s.io/ibm-blockchain-platform-clusterrole created
clusterrole.rbac.authorization.k8s.io/ibm-blockchain-platform-psp-clusterrole created
podsecuritypolicy.extensions/ibm-blockchain-platform-psp created
```

Note: Pod Security Policies (PSPs) and ClusterRoles (CRs) are at the Cluster Scope, so you set them only one time per cluster. Thus, the preceding PSPs and CRs should be applied only one time per cluster.

Manual: Copy and paste piles

Perform the following steps to install the console manually:

1. Define pod security policy as described here.
   IBP Helm charts require specific security and access policies be bound to the target namespace prior to installation. The file that is shown in Example 4-4 defines the PodSecurityPolicy. Save it as `ibm-blockchain-platform-psp.yaml` on your local system.

Example 4-4 ibm-blockchain-platform-psp.yaml

```
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: ibm-blockchain-platform-psp
spec:
  hostIPC: false
  hostNetwork: false
  hostPID: false
  privileged: true
  allowPrivilegeEscalation: true
  readOnlyRootFilesystem: false
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
```

```
    rule: RunAsAny
  fsGroup:
    rule: RunAsAny
  allowedCapabilities:
  - NET_BIND_SERVICE
  - CHOWN
  - DAC_OVERRIDE
  - SETGID
  - SETUID
  - FOWNER
  volumes:
  - '*'
```

2. Save the file shown in Example 4-5 that defines the required ClusterRole for the PodSecurityPolicy as `ibm-blockchain-platform-psp-clusterrole.yaml`.

Example 4-5 ibm-blockchain-platform-psp-clusterrole.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
   annotations:
   name: ibm-blockchain-platform-psp-clusterrole
rules:
 - apiGroups:
   - extensions
   resourceNames:
   - ibm-blockchain-platform-psp
   resources:
   - podsecuritypolicies
   verbs:
   - use
```

3. Save the file in Example 4-6 that defines the required ClusterRole for the blockchain console as `ibm-blockchain-platform-clusterrole.yaml`.

Example 4-6 ibm-blockchain-platform-clusterrole.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
  name: ibm-blockchain-platform-clusterrole
rules:
- apiGroups:
  - '*'
  resources:
  - pods
  - services
  - endpoints
  - persistentvolumeclaims
  - persistentvolumes
  - events
  - configmaps
  - secrets
  - ingresses
  - roles
```

```
  - rolebindings
  - serviceaccounts
  verbs:
  - '*'
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
  - '*'
- apiGroups:
  - ibp.com
  resources:
  - '*'
  verbs:
  - '*'
- apiGroups:
  - apps
  resources:
  - deployments
  - daemonsets
  - replicasets
  - statefulsets
  verbs:
  - '*'
```

4. Save the file in Example 4-7, which defines the **clusterrole** for custom resource definitions at a cluster scope as `crd-clusterrole.yaml`.

Example 4-7 crd-clusterrole.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: crd-clusterrole
rules:
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
  - get
  - watch
  - list
  - create
```

5. After the PodSecurityPolicy, ClusterRole, and ClusterRoleBinding YAML files are saved to your local system, a cluster administrator must use the IBM Cloud Private CLI to bind the policies to your namespace.
   a. Log in to your IBM Cloud Private cluster, and select the target namespace of your deployment.

      ```
      cloudctl login -a https://<cluster_CA_domain>:8443 --skip-ssl-validation
      ```

   b. Use the following commands to apply the policies to your cluster:

      ```
      kubectl apply -f ibm-blockchain-platform-psp.yaml
      ```

```
kubectl apply -f ibm-blockchain-platform-psp-clusterrole.yaml
kubectl apply -f ibm-blockchain-platform-clusterrole.yaml
kubectl apply -f crd-clusterrole.yaml
```

Note: Pod Security Policies (PSPs) and ClusterRoles (CRs) are at the Cluster Scope, so you set them only one time per cluster. Thus, the preceding PSPs and CRs should be applied only one time per cluster.

4.1.3 Scripted console installation

A console installation script is available in the GitHub page for this Redbooks document: https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform-for-Multicloud

The script automates the IBM Blockchain Platform for Multicloud console installation. This script is in the **IBM_Blockchain_Platform_for_Multicloud_Automation** directory of the main GitHub page.

(Instead of the script, you can follow the manual steps for this operation in the section 4.1.4, “Manual console installation” on page 143.)

This section describes the basic steps to set up and run the script. For more information, see the README on the main page of GitHub repository.

Note: This script exists for educational purposes and is not supported by IBM. Nonetheless, the script might provide a valid way for you to quickly deploy consoles, and also delete and redeploy consoles.

Script setup instructions

1. Clone the script to the computer that you use to interact with your cluster by using the command line tools (kubectl and cloudctl), if you have not already done so.

   ```
   git clone --recursive -j8
   https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform
   -for-Multicloud.git ibpbook-scripts && cd
   ibpbook-scripts/IBM_Blockchain_Platform_for_Multicloud_Automation
   ```

2. Log in to your IBM Cloud Private Cluster with a cluster admin user account:

   ```
   cloudctl login -a https://<cluster_hostname>:8443 -n default
   ```

3. Find the chart repo to which you uploaded the blockchain chart, and get its address in your repo:

   ```
   cloudctl catalog charts | grep ibm-blockchain-platform-prod
   cloudctl catalog repos | grep <repo_blockchain-result_of_command_above>
   ```

4. Configure a local blockchain-charts helm repo (by using this address) if you do not already have one:

   ```
   helm repo add blockchain-charts --ca-file "${HOME}"/.helm/ca.pem --cert-file
   "${HOME}"/.helm/cert.pem --key-file "${HOME}"/.helm/key.pem <repo_url_from_end_of_step3>

   helm repo update
   ```

Note: You might get the following error from the **helm repo add** command:

```
Error: Couldn't load repositories file
(/home/support/.helm/repository/repositories.yaml).
```

In this case, you might need to run **'helm init'** (or **'helm init --client-only'** if **tiller** is already installed)

Run this command:

```
helm init --client-only
```

Then, rerun the previous commands for your values:

```
helm repo add blockchain-charts
https://<cluster_hostname>:8443/helm-repo/charts
helm repo update
```

5. Run the script with your designated values.

   TEAM_NUMBER=<*number_of_consoles_to_deploy*> PREFIX=<*prefix_deploy*>
   DOCKER_NAMESPACE=<*helm_chart_loaded_namespace*> STORAGE_CLASS=<*storage_class_setup*>
   CLUSTER_HOSTNAME=<*cluster_hostname*> ARCH=<*hardware_architecture*>
   ADMIN_EMAIL=<*admin_username*> ./Blockchain_Setup.sh

   An explanation of each value in the script is listed with the default value in brackets ([]). A **[Must Set]** flag means that the values must be set by the users during installation. If you do not set a value, the default is used. If a [**Must Set]** variable is not set, the script stops and prompts the user to set the value for that variable.

   – [Must Set] **TEAM_NUMBER**
   The number of consoles that the user wants to deploy. This must be set for chart to run.

   – [Must Set] **PREFIX**
   The prefix for the console installation script. The prefix adds uniqueness so that multiple users can coexist. Do the following search to confirm that your prefix is not being used by a namespace already:

   kubectl get ns | grep *prefix*

   where *prefix* is the name of your prefix.

   – [blockchain-time] **DOCKER_NAMESPACE**
   Holds the blockchain images in its section of the private docker repository.

   – [managed-nfs-storage] **STORAGE_CLASS**
   The storage class name that is used for your console helm release and the subsequent blockchain component deployments. Typically, you set this up by following the instructions in the previous section on storage classes (**Hint:** either **glusterfs** or **nfs**).

Note: If your cluster does not have the storage class yet, complete this operation now and then complete the current step. Run the kubectl command to confirm creation of your storage classes.

The options to use in the cluster for this document have been *italicized* in the following example. These options might be either of the ones with dynamic provisioners (**glusterfs** and **managed-nfs-storage**). You can use the one that you set up earlier, based on your preference. [**managed-nfs-storage**]

```
kubectl get sc
   NAME                        PROVISIONER                    AGE
   glusterfs                   kubernetes.io/glusterfs        7d
   image-manager-storage       kubernetes.io/no-provisioner   58d
   logging-storage-datanode    kubernetes.io/no-provisioner   58d
   managed-nfs-storage         nfs-provisioner                8d
   mongodb-storage             kubernetes.io/no-provisioner   58d
```

- [s390x] **ARCH**
  The hardware architecture used for the helm release. LinuxONE uses **s390x** while x86 uses **amd64**
- [optional] **ADMIN_EMAIL**
  An optional field that gives all consoles this admin username for login. The default is to not set this value. As a result, team numbers 0 to x @ibm.com serve as usernames.

6. After the script runs to completion, find the username and initial password that was set for each console. Also, find the two URLs that access and accept certificates (the console itself and the proxy):

   ```
   cat portList.txt
   ```

7. Visit the URLs that are provided in the **portList** document. If this operation fails at this stage or earlier, look at more details on the README page for this script at this website: https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform-for-Multicloud/blob/master/automation-README.md
8. When you are done with the consoles (if you are using it for development or testing), run the clean-up script that is provided at the end of the **portList.txt** document. When you plan for development deployments, you typically use that as a prefix for easy deletion of all consoles at once. Alternatively, you can adjust the START_NUMBER parameters as part of your cleanup to see which console number to start at, beginning from 0. If 10 consoles were deployed, run the following cleanup operation to delete teams 6-10, as in this example:

   ```
   TEAM_NUMBER=11 PREFIX=garrett PRESTART_NUMBER=6 ./cleanupNamespaces.sh
   ```

Script example that runs with setup

A script example that runs with setup is shown in Example 4-8 on page 138.

Example 4-8 Script example run with setup

```
git clone --recursive -j8
https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform-fo
r-Multicloud.git ibpbook-scripts && cd
ibpbook-scripts/IBM_Blockchain_Platform_for_Multicloud_Automation
      Cloning into 'ibpbook-scripts'...
      remote: Enumerating objects: 6, done.
      remote: Counting objects: 100% (6/6), done.
      remote: Compressing objects: 100% (5/5), done.
      remote: Total 6 (delta 1), reused 6 (delta 1), pack-reused 0
      Unpacking objects: 100% (6/6), done.
      Submodule 'IBM_Blockchain_Platform_for_Multicloud_Automation'
      (https://github.com/siler23/IBM_Blockchain_Platform_for_Multicloud_Automation)
      registered for path 'IBM_Blockchain_Platform_for_Multicloud_Automation'
      Submodule 'nfs-helm-dynamic-multiarch'
      (https://github.com/siler23/nfs-helm-dynamic-multiarch) registered for path
      'nfs-helm-dynamic-multiarch'
      Cloning into
      '/Users/garrettwoodworth/Documents/Redbook_IBP4Multicloud/ibpbook-scripts/IBM_Blockc
      hain_Platform_for_Multicloud_Automation'...
```

```
       remote: Enumerating objects: 46, done.
       remote: Counting objects: 100% (46/46), done.
       remote: Compressing objects: 100% (33/33), done.
       remote: Total 46 (delta 24), reused 34 (delta 12), pack-reused 0
       Cloning into
       '/Users/garrettwoodworth/Documents/Redbook_IBP4Multicloud/ibpbook-scripts/nfs-helm-d
       ynamic-multiarch'...
       remote: Enumerating objects: 122, done.
       remote: Counting objects: 100% (122/122), done.
       remote: Compressing objects: 100% (70/70), done.
       remote: Total 122 (delta 53), reused 105 (delta 36), pack-reused 0
       Receiving objects: 100% (122/122), 180.74 KiB | 2.70 MiB/s, done.
       Resolving deltas: 100% (53/53), done.
       Submodule path 'IBM_Blockchain_Platform_for_Multicloud_Automation': checked out
       'd04f7d3f9cd911179bb66d75cd1426a08140c800'
       Submodule path 'nfs-helm-dynamic-multiarch': checked out
       'c879b3d17c3e8ef105e6be9818365c25c6e23215'

cloudctl login -a https://mycluster.icp:8443 -n default

   Username> admin

   Password>
   Authenticating...
   OK

   Targeted account mycluster Account (id-mycluster-account)

   Targeted namespace default

   Configuring kubectl ...
   Property "clusters.mycluster" unset.
   Property "users.mycluster-user" unset.
   Property "contexts.mycluster-context" unset.
   Cluster "mycluster" set.
   User "mycluster-user" set.
   Context "mycluster-context" created.
   Switched to context "mycluster-context".
   OK

   Configuring helm: /home/support/.helm
   OK

cloudctl catalog charts | grep ibm-blockchain-platform-prod
ibm-blockchain-platform-prod         2.0.0      local-charts           IBM Blockchain
Platform for IBM Cloud Private

cloudctl catalog repos | grep local-charts
   local-charts          https://mycluster.icp:8443/helm-repo/charts
   true

helm repo add blockchain-charts https://mycluster.icp:8443/helm-repo/charts
   Error: Couldn't load repositories file
   (/home/support/.helm/repository/repositories.yaml).
   You might need to run `helm init` (or `helm init --client-only` if tiller is already
   installed)

helm init --client-only
   Creating /home/support/.helm/repository
   Creating /home/support/.helm/repository/cache
```

```
   Creating /home/support/.helm/repository/local
   Creating /home/support/.helm/plugins
   Creating /home/support/.helm/starters
   Creating /home/support/.helm/cache/archive
   Creating /home/support/.helm/repository/repositories.yaml
   Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
   Adding local repo with URL: http://127.0.0.1:8879/charts
   $HELM_HOME has been configured at /home/support/.helm.
   Not installing Tiller due to 'client-only' flag having been set
   Happy Helming!

helm repo add blockchain-charts --ca-file "${HOME}"/.helm/ca.pem --cert-file
"${HOME}"/.helm/cert.pem --key-file "${HOME}"/.helm/key.pem
https://mycluster.icp:8443/helm-repo/charts
   "blockchain-charts" has been added to your repositories

helm repo update
   Hang tight while we grab the latest from your chart repositories...
   ...Skip local chart repository
   ...Successfully got an update from the "blockchain-charts" chart repository
   ...Successfully got an update from the "stable" chart repository
   Update Complete. Happy Helming!

TEAM_NUMBER=1 PREFIX=ibp-auto-deploy DOCKER_NAMESPACE=blockchain-time
STORAGE_CLASS=managed-nfs-storage ARCH=s390x CLUSTER_HOSTNAME=mycluster.icp
ADMIN_EMAIL=siler23@ibm.com ./Blockchain_Setup.sh
```

Note: Because the command in Example 4-8 uses many of the defaults (for demonstration purposes) this script would be equivalent to running the following command:

TEAM_NUMBER=1 PREFIX=ibp-auto-deploy ADMIN_EMAIL=siler23@ibm.com ./Blockchain_Setup.sh

Example 4-9 shows the output for this command, which results in the application of all default values.

Example 4-9 Blockchain setup script

```
_____________ _____________________ _____________  _____________________  __
___  __ )__  / __  __ \_  ____/__  //_/_  ____/__  / / /__    |___  _/__  | / /
__  __  |_  /  _  / / /  /    __  ,<   _  /    __  /_/ /__  /| |__  / __   |/ /
_  /_/ /_  /___/ /_/ // /___  _  /| | / /___  _  __  / _  ___ |_/ /  _  /|  /
/_____/ /_____/\____/ \____/  /_/ |_| \____/  /_/ /_/  /_/  |_/___/  /_/ |_/

_____________________  _______________
__  ___/_  __ \_  / / /__    |__  __ \
_____ \_  / / /  / / /__  /| |_  / / /
____/ // /_/ // /_/ / _  ___ |  /_/ /
/____/ \___\_\____/  /_/  |_/_____/


 ---- Creating 1 of Optools Instances ----

Setting up namespace for team00
+ kubectl create ns ibp-auto-deploy-team00
namespace/ibp-auto-deploy-team00 created
+ kubectl config set-context --current --namespace=ibp-auto-deploy-team00
Context "mycluster-context" modified.
```

```
+ kubectl create secret generic team00-ibp-ui-secret --from-literal=password=team00pw12438
secret/team00-ibp-ui-secret created
+ kubectl create sa ibp
serviceaccount/ibp created
+ kubectl create rolebinding ibp-admin --serviceaccount ibp-auto-deploy-team00:ibp
--clusterrole=ibm-blockchain-platform-clusterrole
rolebinding.rbac.authorization.k8s.io/ibp-admin created
+ kubectl create rolebinding ibp-psp --group system:serviceaccounts:ibp-auto-deploy-team00
--clusterrole=ibm-blockchain-platform-psp-clusterrole
rolebinding.rbac.authorization.k8s.io/ibp-psp created
+ kubectl create clusterrolebinding ibp-auto-deploy-team00-ibp-crd --serviceaccount
ibp-auto-deploy-team00:ibp --clusterrole=crd-clusterrole
clusterrolebinding.rbac.authorization.k8s.io/ibp-auto-deploy-team00-ibp-crd created
++ kubectl get secret -n blockchain-time sa-blockchain-time -o 'jsonpath={.data.\.dockerconfigjson}'
++ base64 --decode
+ kubectl create secret generic blockchain-docker-registry
'--from-literal=.dockerconfigjson={"auths":{"mycluster.icp:8500":{"username":"sa-blockchain-time","password
":"9jbd5hsmf6","auth":"c2EtYmxvY2tjaGFpbi10aW1lOjhrYmQ0aHhtZjY="}}}' --type=kubernetes.io/dockerconfigjson
secret/blockchain-docker-registry created
+ set +x
+ helm install --tls --namespace ibp-auto-deploy-team00 blockchain-charts/ibm-blockchain-platform-prod
--version 2.0.0 -n ibp-auto-deploy-team00-ibp-console --set app.email=siler23@ibm.com --set
app.multiArch=true --set app.passwordSecretName=team00-ibp-ui-secret --set app.proxyIP=9.56.28.25 --set
app.serviceAccountName=ibp --set arch=s390x --set dataPVC.storageClassName=managed-nfs-storage --set
image.caImage=mycluster.icp:8500/blockchain-time/ibp2/hlfabric-ca --set image.caTag=1.4.1 --set
image.configtxlatorImage=mycluster.icp:8500/blockchain-time/ibp2/hlfabric-configtxlator --set
image.configtxlatorTag=1.4.1 --set
image.couchdbImage=mycluster.icp:8500/blockchain-time/ibp2/hlfabric-couchdb --set image.couchdbTag=1.4.1
--set image.deployerImage=mycluster.icp:8500/blockchain-time/ibp2/deployer-to-go --set
image.deployerTag=2.0.0 --set image.dindImage=mycluster.icp:8500/blockchain-time/ibp2/hlfabric-dind --set
image.dindTag=1.4.1 --set image.fluentdImage=mycluster.icp:8500/blockchain-time/fluentd --set
image.fluentdTag=v1.4-2 --set image.grpcwebImage=mycluster.icp:8500/blockchain-time/ibp2/hlfabric-grpcweb
--set image.grpcwebTag=1.4.1 --set image.imagePullSecret=blockchain-docker-registry --set
image.initImage=mycluster.icp:8500/blockchain-time/ibp2/hlfabric-init --set image.initTag=1.4.1 --set
image.operatorImage=mycluster.icp:8500/blockchain-time/ibp2/ibp-operator --set image.operatorTag=2.0.0
--set image.optoolsImage=mycluster.icp:8500/blockchain-time/op-tools/op-tools --set image.optoolsTag=2.0.0
--set image.ordererImage=mycluster.icp:8500/blockchain-time/ibp2/hlfabric-orderer --set
image.ordererTag=1.4.1 --set image.peerImage=mycluster.icp:8500/blockchain-time/ibp2/hlfabric-peer --set
image.peerTag=1.4.1 --set ingress.optools.hostname=9.56.28.25 --set ingress.optools.port=30003 --set
ingress.proxy.hostname=9.56.28.25 --set ingress.proxy.port=30004 --set license=accept --set
resources.configtxlator.limits.cpu=25m --set resources.configtxlator.limits.memory=100Mi --set
resources.configtxlator.requests.cpu=25m --set resources.configtxlator.requests.memory=10Mi --set
resources.couchdb.limits.cpu=250m --set resources.couchdb.limits.memory=250Mi --set
resources.couchdb.requests.cpu=50m --set resources.couchdb.requests.memory=100Mi --set
resources.deployer.limits.cpu=100m --set resources.deployer.limits.memory=100Mi --set
resources.deployer.requests.cpu=25m --set resources.deployer.requests.memory=10Mi --set
resources.operator.limits.cpu=100m --set resources.operator.limits.memory=100Mi --set
resources.operator.requests.cpu=25m --set resources.operator.requests.memory=20Mi --set
resources.optools.limits.cpu=250m --set resources.optools.limits.memory=250Mi --set
resources.optools.requests.cpu=50m --set resources.optools.requests.memory=100Mi
NAME:   ibp-auto-deploy-team00-ibp-console
LAST DEPLOYED: Fri Aug 23 08:32:51 2019
NAMESPACE: ibp-auto-deploy-team00
STATUS: DEPLOYED

RESOURCES:
==> v1/ConfigMap
NAME                                              DATA  AGE
ibp-auto-deploy-team00-ibp-console-env            9     2s
ibp-auto-deploy-team00-ibp-console-template       1     2s
```

```
ibp-auto-deploy-team00-ibp-console-deployer-template  1     2s

==> v1/PersistentVolumeClaim
NAME                                STATUS   VOLUME               CAPACITY  ACCESS MODES  STORAGECLASS  AGE
ibp-auto-deploy-team00-ibp-console  Pending  managed-nfs-storage  2s

==> v1/Service
NAME                                        TYPE      CLUSTER-IP  EXTERNAL-IP  PORT(S) AGE
ibp-auto-deploy-team00-ibp-console-optools  NodePort  10.0.152.0  <none>
3000:30003/TCP,3001:30004/TCP  2s

==> v1/Deployment
NAME                                DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
ibp-auto-deploy-team00-ibp-console  1        1        1           0          2s

==> v1/Pod(related)
NAME                                                 READY  STATUS   RESTARTS  AGE
ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5  0/5    Pending  0         0s


NOTES:
Open IBM Blockchain Console:
https://9.56.28.25:30003

Note: Please go to the following URL and accept the certificate:
https://9.56.28.25:30004

+ RC=0
+ set +x
Checking deploy for team00
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Pending, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Pending, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Pending, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Pending, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = ContainerCreating, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = ContainerCreating, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = ContainerCreating, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 3/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-2qjp5 to start completion. Status = Running, Readiness = 4/5

@@@@@@@@  @@@  @@@  @@@  @@@   @@@@@@   @@@  @@@  @@@@@@@@  @@@@@@@
@@@@@@@@  @@@  @@@@ @@@  @@@  @@@@@@@   @@@  @@@  @@@@@@@@  @@@@@@@@
@@!       @@!  @@!@!@@@  @@!  !@@       @@!  @@@  @@!       @@!  @@@
!@!       !@!  !@!!@!@!  !@!  !@!       !@!  @!@  !@!       !@!  @!@
@!!!:!    !!@  @!@ !!@!  !!@  !!@@!!    @!@!@!@!  @!!!:!    @!@  !@!
!!!!!:    !!!  !@!  !!!  !!!   !!@!!!   !!!@!!!!  !!!!!:    !@!  !!!
!!:       !!:  !!:  !!!  !!:       !:!  !!:  !!!  !!:       !!:  !!!
:!:       :!:  :!:  !:!  :!:      !:!   :!:  !:!  :!:       :!:  !:!
 ::        ::   ::   ::   ::  :::: ::   ::   :::   :: ::::   :::: ::
 :        :    ::    :   :    :: : :     :   : :  : :: ::   :: :  :

It took 1 minutes and 16 seconds to setup 1 optools instances each in an unique namespace
cat portList.txt
List of ports for each team for quick reference
****************   TEAM0  ****************
```

```
team00 optools URL: https://9.56.28.25:30003
team00 proxy URL: https://9.56.28.25:30004
team00 USERNAME: siler23@ibm.com
team00 PASSWORD: team00pw12438

Full Cleanup Command: TEAM_NUMBER=1 PREFIX=ibp-auto-deploy ./cleanupNamespaces.sh
```

4.1.4 Manual console installation

In this section, we describe manual installation of the console.

Setting up namespace for installation

Note: Do all of these steps one time per namespace. You must deploy each console in its own namespace. In other words, these steps must be performed for each new console that you create.

1. If you deploy more than one console, create subsequent namespaces for other consoles:

   ```
   kubectl create ns <namespace>
   ```

2. Set kubectl to use this new namespace.

   ```
   kubectl config set-context --current --namespace=<namespace>
   ```

3. Create a secret to store the default console password. The user resets this password upon login:

   ```
   kubectl create secret generic ibp-ui-secret
   --from-literal=password=<chosen_password>
   ```

4. After you apply the policies, run the following classes to create a new service account:

   ```
   kubectl create sa <sa_name>
   ```

5. Run the following commands to grant your service account the required level of permissions to deploy your console:

   The commands specify the name of your target namespace and newly created service account. This action creates bindings at the namespace level (**rolebinding**) and cluster level (**clusterrolebinding**), which grants the permissions that are necessary for the IBM Blockchain Platform to run.

   ```
   kubectl create rolebinding <rolebinding_name> --serviceaccount <namespace>:<sa_name>
   --clusterrole=ibm-blockchain-platform-clusterrole

   kubectl create rolebinding <rolebinding_name> --group system:serviceaccounts:<namespace>
   --clusterrole=ibm-blockchain-platform-psp-clusterrole

   kubectl create clusterrolebinding <namespace>-ibp-crd --serviceaccount
   <namespace>:<sa_name> --clusterrole=crd-clusterrole
   ```

6. Create a Docker Registry Secret. You use the secret name to access the images in the unique namespace that you pushed them to. To do this you copy the docker secret name while granting pull-only access to these images in the namespace to which you pushed the chart.

   ```
   kubectl create secret generic <docker_secret_name>
   --from-literal=.dockerconfigjson=$(kubectl get secret -n <helm_chart_loaded_namespace>
   sa-<helm_chart_loaded_namespace> -o jsonpath='{.data.\.dockerconfigjson}' | base64
   --decode) --type=kubernetes.io/dockerconfigjson
   ```

7. (*Optional*) [Only if necessary for your organization] Create a secret for using your organization's own TLS certificates instead of making these certificates self-signed. If you do this, create your **tls** private key and have your organization's certificate authority create a certificate for it first. Then, create a Kubernetes secret using the certificate and private key by providing paths to them by running the following command:

   ```
   kubectl create secret tls <tls_secret_name> --cert=<"path-to-tls-cert-file">
   --key=<"path-to-tls-key-file">
   ```

   Example 4-10 shows an example of namespace setup.

Example 4-10 Example of namespace setup

```
kubectl create ns blockchain-console1
namespace/blockchain-console1 created

kubectl config set-context --current --namespace=blockchain-console1
Context "mycluster-context" modified.

kubectl create secret generic ibp-ui-secret
--from-literal=password=I-Live-4-SECurity-OhYEAH
secret/ibp-ui-secret created

kubectl create sa ibp
serviceaccount/ibp created

kubectl create rolebinding ibp-admin --serviceaccount blockchain-console1:ibp
--clusterrole=ibm-blockchain-platform-clusterrole
rolebinding.rbac.authorization.k8s.io/ibp-admin created

kubectl create rolebinding ibp-psp --group system:serviceaccounts:blockchain-console1
--clusterrole=ibm-blockchain-platform-psp-clusterrole
rolebinding.rbac.authorization.k8s.io/ibp-psp created

kubectl create clusterrolebinding blockchain-console1-ibp-crd --serviceaccount
blockchain-console1:ibp --clusterrole=crd-clusterrole
clusterrolebinding.rbac.authorization.k8s.io/blockchain-console1-ibp-crd created

kubectl create secret generic blockchain-docker-registry
--from-literal=.dockerconfigjson=$(kubectl get secret -n blockchain-time sa-blockchain-time
-o jsonpath='{.data.\.dockerconfigjson}' | base64 --decode)
--type=kubernetes.io/dockerconfigjson
secret/blockchain-docker-registry created

(Optional) [Only if necessary for your organization]
kubectl create secret tls blockchain-tls --cert="tls.crt" --key="tls.key"
secret/blockchain-tls created
```

Installing console

First, go to the catalog and look in *local-charts* for the blockchain chart and click it. Figure 4-1 on page 145 shows it highlighted in blue. Click the Repository drop-down and select **local-charts**.

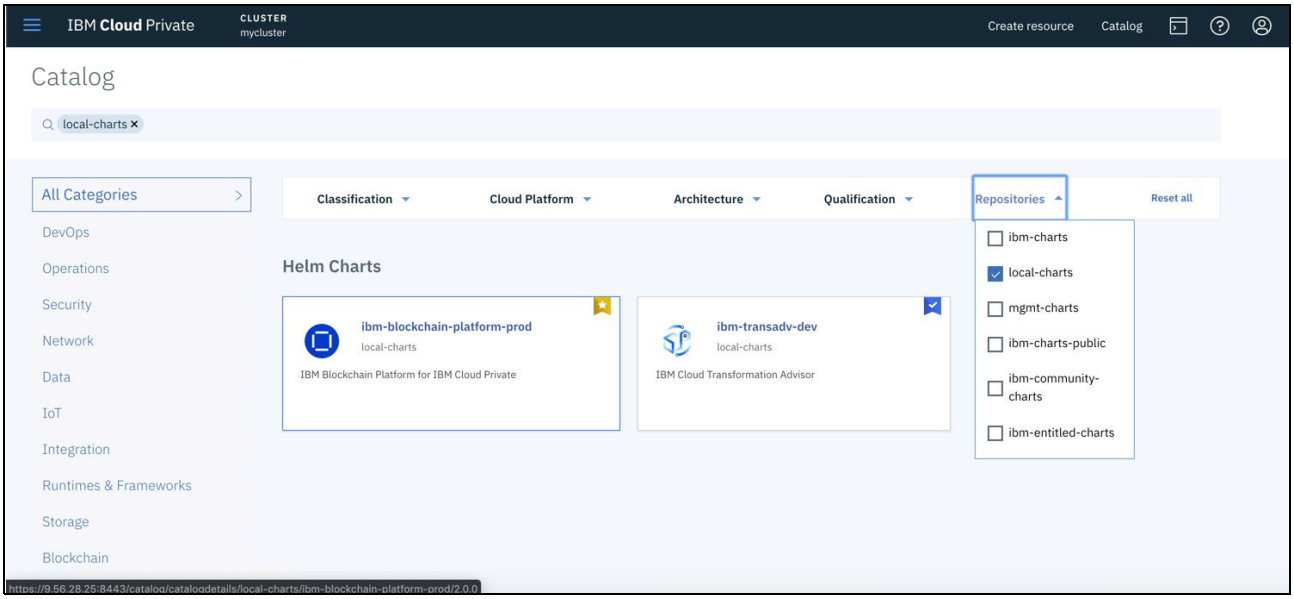


Figure 4-1 Installing console -1

8. Configure the chart with the following values for your cluster. We have labeled the pictures with numbers that match each value. The value that is required for this installation is shown in brackets ([ ]).

The first three values apply to Figure 4-2 on page 146.

Table 4-1 Values for callout numbers 1 to 3

1. **Helm release name** is the name you choose for your helm release. Make sure to use a unique name for each helm release in the cluster. [blockchain-console-1] 2. **Target namespace** is the namespace you set up in the previous sections. [blockchain-console1] 3. **Target Cluster** should be local-cluster unless installing on a different cluster. [local-cluster]

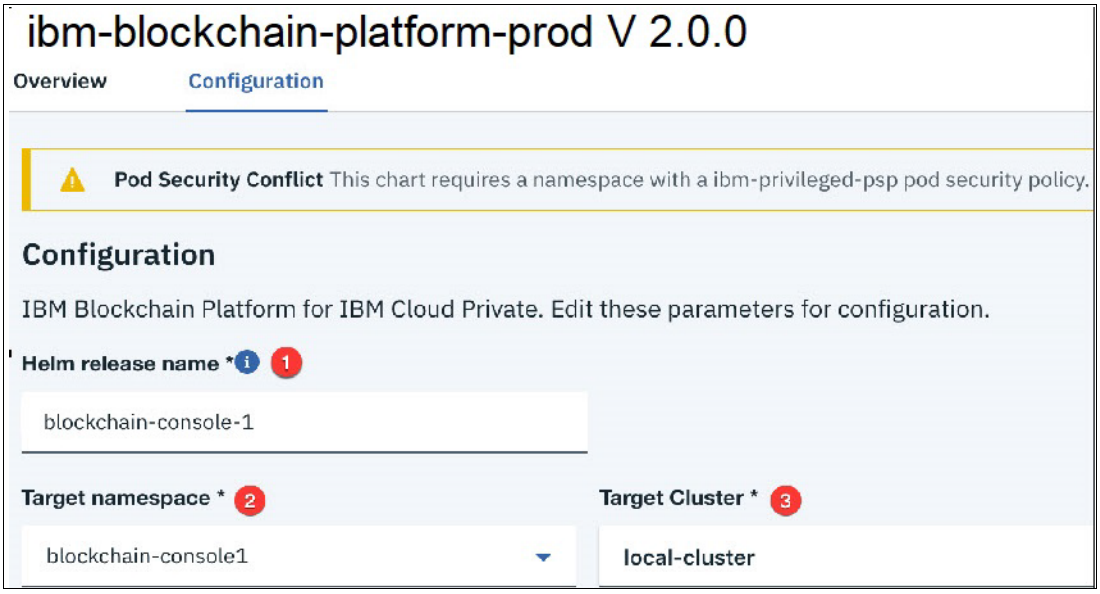


Figure 4-2 Installing console -2

Values 4 to 6 apply to Figure 4-3.

Table 4-2 Values for callout numbers 4 to 6

4. **Target Namespace Policies** displays the policies that are configured to all service accounts in the namespace that you selected in field 2, **Target namespace**. Following the earlier setup, *ibm-blockchain-platform-psp* should be available in your list. If you do not have this policy, go back to the section, “Setting up namespace for installation” on page 143. [ibm-blockchain-platform-psp, ibm-restricted-psp] 5. Click the drop-down arrow to the left of **All Parameters** to see all the parameters that you can configure. Some of the parameter settings are essential enablement of the console on IBM LinuxONE, as described in the procedure. 6. **Architecture** of the hardware where you run blockchain. **s390x** applies to IBM LinuxONE and **amd64** applies to x86 servers. [s390x]

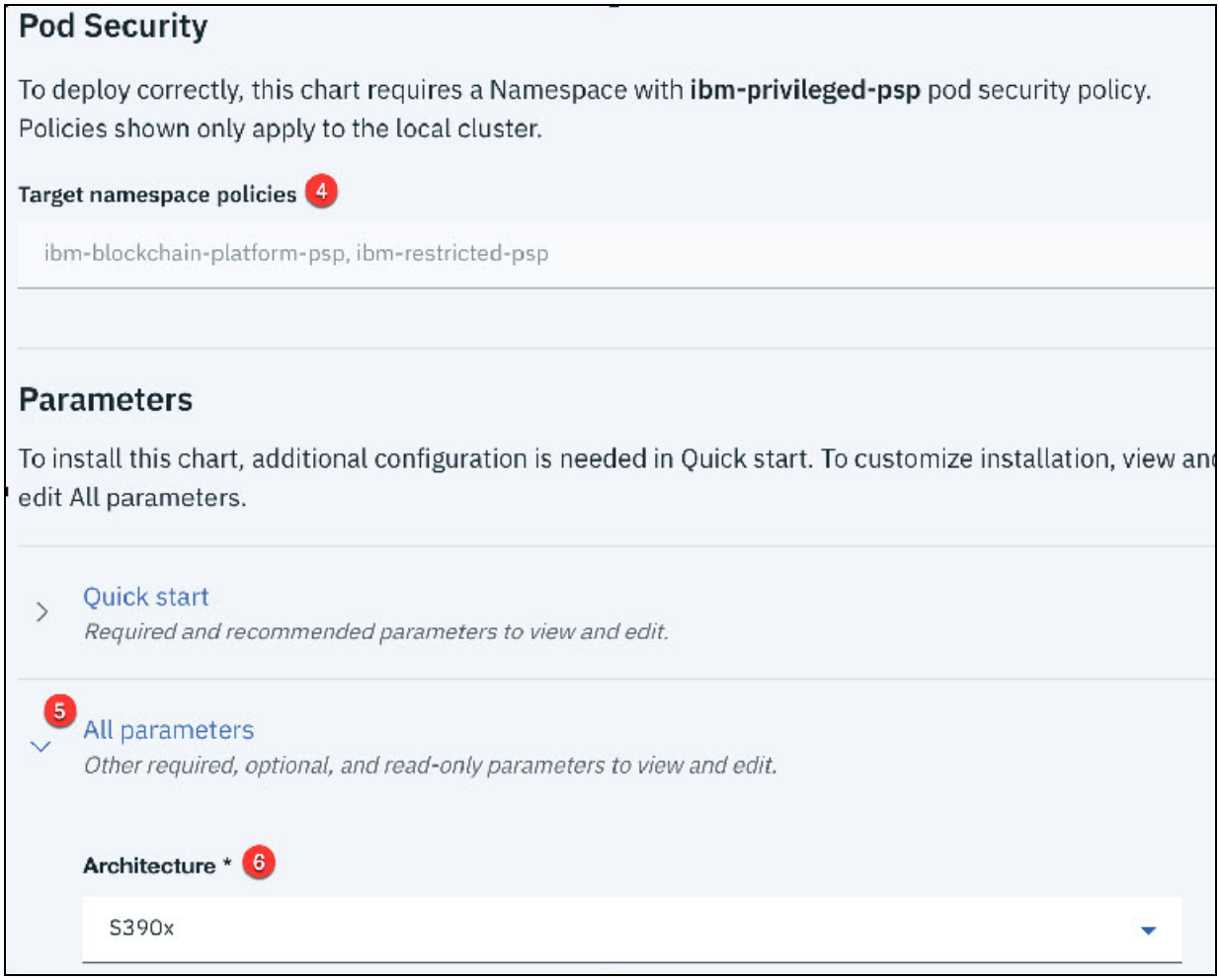


Figure 4-3 Installing console -3

Values 7 to 11 apply to Figure 4-4 on page 148.

Table 4-3 Values for callout numbers 7 to 11

7. **serviceAccountname** is the name of the service account setup in “Setting up namespace for installation” on page 143. If you forget, you can check with kubectl. [ibp]

```
kubectl get sa
NAME      SECRETS   AGE
default   1         36h
ibp       1         36h
```

8. **Proxy IP** is the IP of your proxy node for IBM Cloud Private. (It might also be any worker node that is accessible from your web browser.) You can find this if you do not already know it. [9.56.28.25]

```
kubectl get nodes -l proxy -o
jsonpath='{.items[0].metadata.labels.kubernetes\.io\/hostname}' && echo
9.56.28.25
```

9. **Console administrator email** will be the initial user’s username. Enter a username that you want. [book@ibm.com]

10. **Console administrator password secret name** is the secret that holds the initial password for console users setup in **Setting up Namespaces for Installation** section. You can use kubectl to find this secret. [ibp-ui-secret]

```
kubectl get secrets --field-selector=type=Opaque
NAME            TYPE     DATA   AGE
ibp-ui-secret   Opaque   1      37h
```

11. **imagePullSecret name** is the name for the imagePullSecret used by all pods to pull the blockchain images. Set this to the one that you created in **Setting up Namespaces for Installation.** You can use kubectl to find this secret. [blockchain-docker-registry]

```
kubectl get secrets --field-selector=type=kubernetes.io/dockerconfigjson
NAME                         TYPE                                  DATA   AGE
blockchain-docker-registry   kubernetes.io/dockerconfigjson   1      37h
```

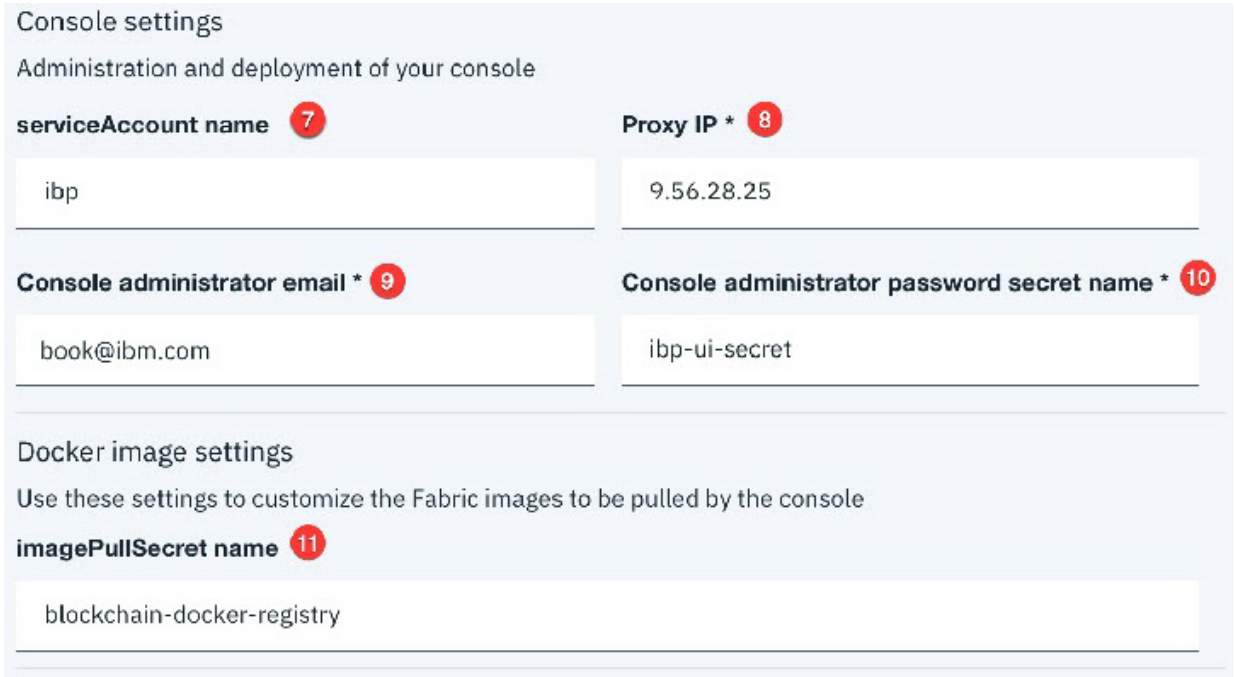


Figure 4-4 Installing console -4

Values 12 to 19 apply to Figure 4-5 on page 150.

Table 4-4 Values for callout numbers 12 to 19

Callout numbers 12 to 19

12. **Console Hostname** is the IP of your proxy node for IBM Cloud Private. (It could also be any worker node that is accessible from your web browser.) This value is the same as callout number 8 (Proxy IP). Again, you can use kubectl to find this value. [9.56.28.25]
`kubectl get nodes -l proxy -o jsonpath='{.items[0].metadata.labels.kubernetes\.io\/hostname}' && echo`
`9.56.28.25`

13. **Console port** is the port you will use to externally access your IBM Blockchain Platform console. Choose an open NodePort from ports 30000-32767. Find used NodePorts with kubectl and pick the first open one. [30003]
`kubectl get svc --all-namespaces -o go-template='{{range .items}}{{range.spec.ports}}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}{{end}}' | sort`
`30000`
`30001`
`30002`
`30170`
Note: The notes in the chart state that you should pick an open port from 31310-31220. These notes should read 'an open port from 30000-32767'. Additionally, these ports do not need to be successive. For example, 30003 and 300005 would also work here.

14. **Proxy Hostname** is the IP of your proxy node for IBM Cloud Private. (It could also be any worker node that is accessible from your web browser.) This value is the same as callout number 8 (Proxy IP) and callout number 12 (console hostname). Again, you can use kubectl to find this value. [9.56.28.25]
`kubectl get nodes -l proxy -o jsonpath='{.items[0].metadata.labels.kubernetes\.io\/hostname}' && echo`
`9.56.28.25`

15. **Proxy port** is the port used as a proxy for your IBM Blockchain Platform console. Choose an open NodePort from ports 30000-32767. Find used NodePorts with kubectl and pick the second open one (the first one should have been picked for your **Console port** above). [30004]
`kubectl get svc --all-namespaces -o go-template='{{range .items}}{{range.spec.ports}}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}{{end}}' | sort`
`30000`
`30001`
`30002`
`30170`
Note: The notes in the chart state that you should pick an open port from 31310-31220. These notes should read 'an open port from 30000-32767'. Additionally, these ports do not need to be successive. For example, 30003 and 300005 would also work here.

16. **TLS Secret [Optional]** is the TLS secret that is required if you are using your own (for example, bring your own) TLS certificate for the console. If you wanted this option, you would first create the secret in "Setting up namespace for installation" on page 143. Typically, you want your certificates to be issued by your organization's CA. If this is not a requirement for you, do not enter a secret. And in that case, a TLS certificate and secret is created for you during the console installation process. To confirm your TLS secret or lack of one, you can use kubectl. [blank]
`kubectl get secrets --field-selector=type=kubernetes.io/tls`
`No resources found.`
Note: If your organization wanted to use its own certificate authority for the IBM Blockchain Console TLS certificate, you would first create this secret in "Setting up namespace for installation" on page 143. In this case, the preceding **get secrets** command would give output similar to the following:
`kubectl get secrets --field-selector=type=kubernetes.io/tls`
`NAME TYPE DATA AGE`
`blockchain-tls kubernetes.io/tls 2 2m45s`
In that case, enter **blockchain-tls** (in your case, whatever the name of the secret from the command above) in the TLS Secret field (which is number 16 in the image).

17. **Volume Claim Size** is the size of the persistent volume that is created for use by the IBM Blockchain Platform for Multicloud console to persist its database. Leave as the default. [10Gi]

Callout numbers 12 to 19

18. **Storage class name** is the name of the storage class that you have set up. Typically, you set this up by following the instructions in the previous section on storage classes (Hint: either **glusterfs** or **nfs**). Run the following kubectl command to see your created storage classes. The options to use are italicized here, which could be either of the ones that have dynamic provisioners (**glusterfs** and **managed-nfs-storage**). You would use the one that you set up earlier based on your preference. [managed-nfs-storage]
`kubectl get sc`
`NAME PROVISIONER AGE`
`glusterfs kubernetes.io/glusterfs 7d`
`image-manager-storage kubernetes.io/no-provisioner 58d`
`logging-storage-datanode kubernetes.io/no-provisioner 58d`
`managed-nfs-storage nfs-provisioner 8d`
`mongodb-storage kubernetes.io/no-provisioner 58d`

19. **Storage access mode** describes the access mode to use for your persistent volume/persistent volume claim. This dictates whether one or many nodes can mount the volume concurrently. If other pods in Kubernetes want to mount the blockchain volume to read from its data, **`ReadWriteMany`** is necessary (because we cannot guarantee that this pod will land on the same node if it is following best practices). This approach can also prevent lock problems if a node goes down. **`ReadWriteOnce`** means that only one node can mount the persistent volume claim as a volume at a time. [ReadWriteMany]

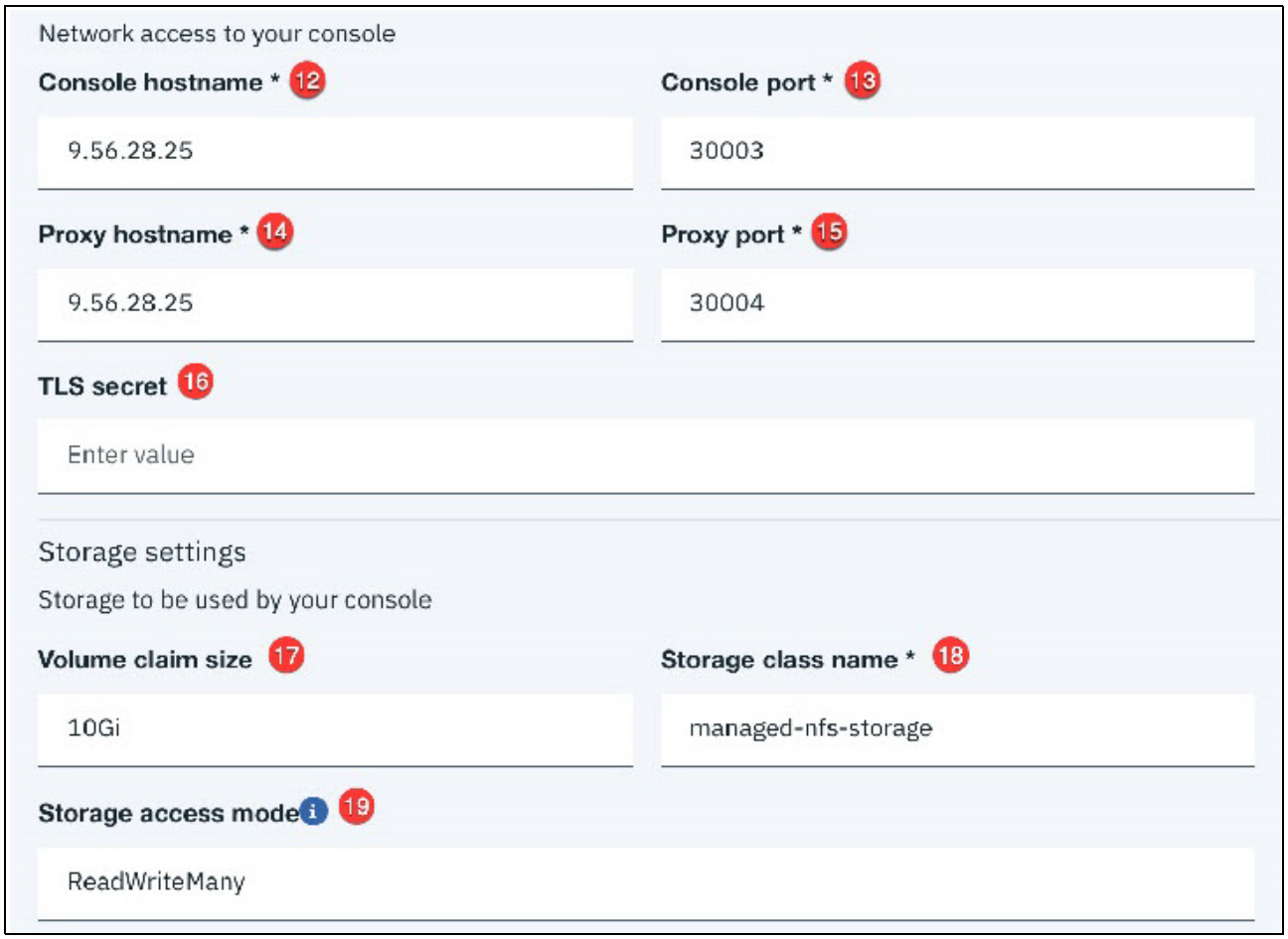


Figure 4-5 Installing console -5

Minimum Resource Configurations for the chart are in Figure 4-6 on page 151 and Figure 4-7 on page 159. (They are spread over two figures due to the length of the pictures.) These values should enable you to use the console with a lower resource

burden for the minimum configuration. If you have enough resources for the default values in the chart, this approach is advised for production usage. [Listed values are for minimum configuration. Default values in the chart are for production configuration].

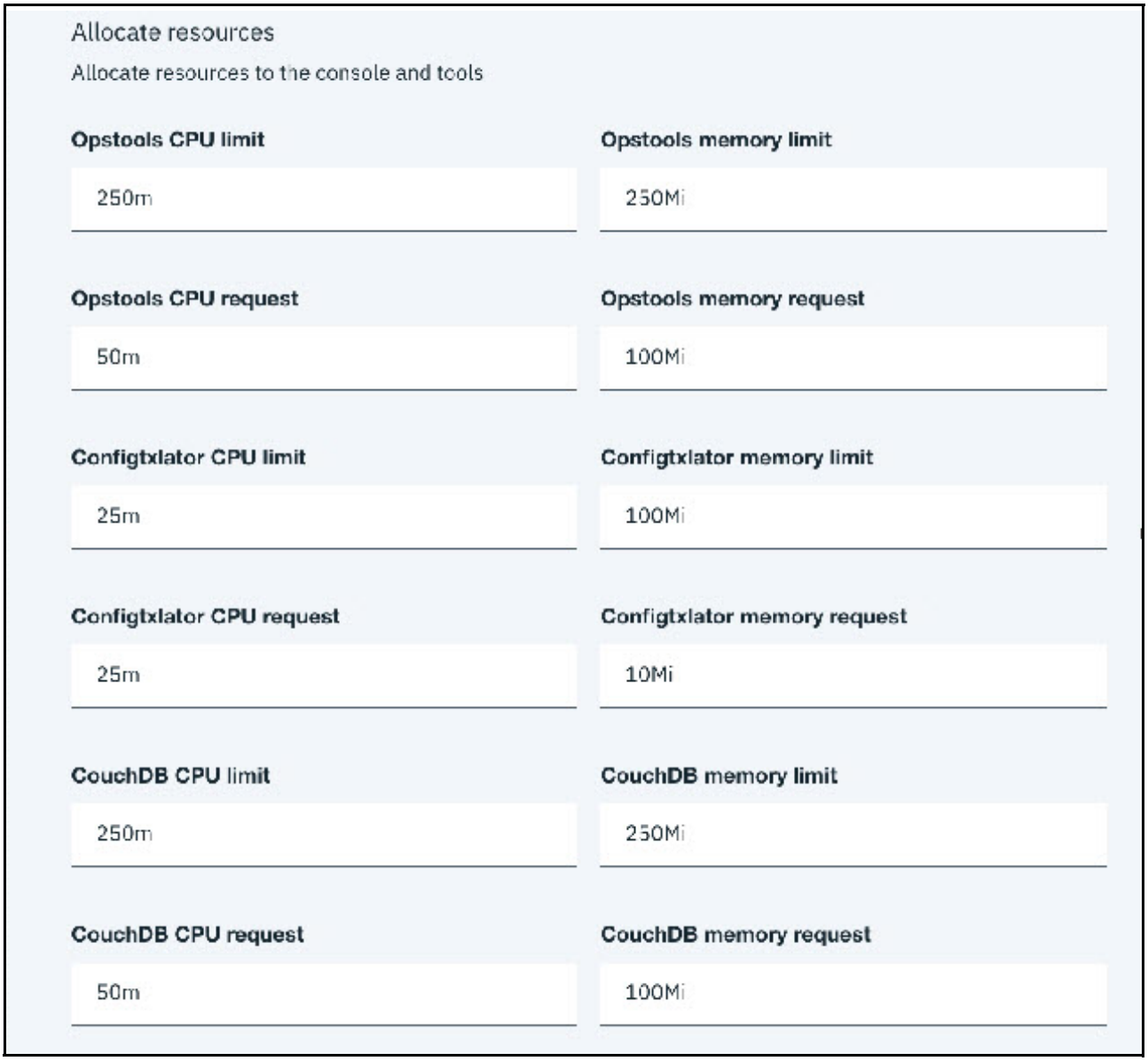


Figure 4-6 Installing console -6

Click **Install** to install the helm release after you enter all the values above.

To learn how to verify your installation, see section 4.2.1, “Verifying installation of the blockchain console” on page 152 section.

Note: If you get any errors on the installation step in the UI, read the error and apply the necessary change. For help, visit section 4.5, “Troubleshooting the installation” on page 194.

4.2 Verifying console installation and initializing console with users

This section describes these tasks for the IBM Blockchain Platform for Multicloud web console:

- 4.2.1, “Verifying installation of the blockchain console”
- 4.2.2, “Initializing blockchain console for other users” on page 167

4.2.1 Verifying installation of the blockchain console

Before you access the console through its URL, you can confirm its installation by using one of the options that is described in this section:

- “Verifying the console of a scripted installation”
- “Verifying the console with Kubectl (command line)” on page 153
- “Verifying the console with the IBM Cloud Private user interface (UI)” on page 158

Verifying the console of a scripted installation

When you use the scripted console installation that is described in section 4.1.3, “Scripted console installation” on page 136, the script performs verification itself. Here is the output of a successful completion:

Example 4-11 Output of a successful completion

```
Checking deploy for team00
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-qn8xc to start
completion. Status = ContainerCreating, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-qn8xc to start
completion. Status = ContainerCreating, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-qn8xc to start
completion. Status = Running, Readiness = 0/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-qn8xc to start
completion. Status = Running, Readiness = 1/5
Waiting for pod ibp-auto-deploy-team00-ibp-console-6d45d47868-qn8xc to start
completion. Status = Running, Readiness = 4/5

@@@@@@@@  @@@  @@@  @@@  @@@   @@@@@@   @@@  @@@  @@@@@@@@  @@@@@@@
@@@@@@@@  @@@  @@@@ @@@  @@@  @@@@@@@   @@@  @@@  @@@@@@@@  @@@@@@@@
@@!       @@!  @@!@!@@@  @@!  !@@       @@!  @@@  @@!       @@!  @@@
!@!       !@!  !@!!@!@!  !@!  !@!       !@!  @!@  !@!       !@!  @!@
@!!!:!    !!@  @!@ !!@!  !!@  !!@@!!    @!@!@!@!@  @!!!:!    @!@  !@!
!!!!!:    !!!  !@!  !!!  !!!   !!@!!!   !!!@!!!!  !!!!!:    !@!  !!!
!!:       !!:  !!:  !!!  !!:       !:!  !!:  !!!  !!:       !!:  !!!
:!:       :!:  :!:  !:!  :!:      !:!   :!:  !:!  :!:       :!:  !:!
 ::        ::   ::   ::   ::  :::: ::   ::   :::   :: ::::   :::: ::
 :        :    ::    :   :    :: : :     :   : :  : :: ::   :: :  :

It took 1 minutes and 19 seconds to setup 1 optools instances each in an unique
namespace
```

Note: At this point, you know that the console installation was successful. Typically, you then view the **portList.txt** file to get the URL of the console, as shown here.

```
cat portList.txt

List of ports for each team for quick reference

***************   TEAM0  ***************
team00 optools URL: https://9.56.28.25:30003
team00 proxy URL: https://9.56.28.25:30004
team00 USERNAME: siler23@ibm.com
team00 PASSWORD: team00pw2266

Full Cleanup Command: TEAM_NUMBER=1 PREFIX=ibp-auto-deploy
./cleanupNamespaces.sh
```

You use these details to log in and initialize the console as described in 4.2.2, “Initializing blockchain console for other users” on page 167.

If the script times out instead of reaching this message, follow the steps in the “Verifying the console with Kubectl (command line)” section. If that verification fails, seek further help in section 4.5, “Troubleshooting the installation” on page 194.

Verifying the console with Kubectl (command line)

Perform the following steps:

1. Choose one of the options to set up kubectl to point to the namespace for your blockchain console. (Choose only one of the two options.)

Kubectl option	Details
cloudctl	Use cloudctl to log in to the namespace that you installed the blockchain console helm release into. This action generates two results: ► Configures kubectl with a new authentication token for the Kubernetes API server. ► Sets kubectl to point to the correct namespaces for commands. You don’t need to use the **-n namespace** (the alternative) flag. The syntax is as follows: `cloudctl login -a https://<cluster_hostname>:8443 --skip-ssl-validation -n <blockchain_console_namespace>` Example 4-12 shows a sample **cloudctl** login.
kubectl config	Set the namespace of a previously authenticated user to the namespace of the blockchain console helm release by using the kubectl config command. (Previous authentication would have been done through cloudctl or a kubectl token that was received from the IBM Cloud Private UI.) `kubectl config set-context --current --namespace=<blockchain-console-namespace>` Example 4-13 on page 154 shows a sample command for setting the namespace for **kubectl config**.

Example 4-12 Sample cloudctl login

```
cloudctl login -a https://mycluster.icp:8443 --skip-ssl-validation -n
ibp-auto-deploy-team00

Username> admin

Password>
Authenticating...
OK

Targeted account mycluster Account (id-mycluster-account)
```

```
Targeted namespace ibp-auto-deploy-team00

Configuring kubectl ...
Property "clusters.mycluster" unset.
Property "users.mycluster-user" unset.
Property "contexts.mycluster-context" unset.
Cluster "mycluster" set.
User "mycluster-user" set.
Context "mycluster-context" created.
Switched to context "mycluster-context".
OK

Configuring helm: /Users/garrettwoodworth/.helm
```

Example 4-13 Setting the namespace for kubectl

```
kubectl config set-context --current --namespace=ibp-auto-deploy-team00
Context "mycluster-context" modified.
```

2. Run the following command to confirm the availability of the console:

   kubectl get deployments -l app=ibm-blockchain-platform-prod

   When AVAILABLE =1 as shown in **bold** in Example 4-14, the console is available. When AVAILABLE=0 as shown in **bold** in Example 4-15, the console is not available.

   Example 4-14 The deployment is available

```
kubectl get deploy -l app=ibm-blockchain-platform-prod
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
ibp-auto-deploy-team00-ibp-console   1/1     1            1           4h39m
```

   Example 4-15 The deployment is not available

```
kubectl get deploy -l app=ibm-blockchain-platform-prod
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
ibp-auto-deploy-team00-ibp-console   0/1     0            0           4h50m
```

   If AVAILABLE=1, go to 4.2.2, “Initializing blockchain console for other users” on page 167. If AVAILABLE=0, it is possible that the console is still loading. In the meantime, you can proceed with the other verification tests in the following steps.

3. Run the following commands to identify pods of the deployment to crosscheck.

   kubectl get pods -l app=ibm-blockchain-platform-prod

   – If pods are found, you see a list like the one in Example 4-16. ***Go to the next Step.***

   Example 4-16 There is a pod

```
kubectl get pods -l app=ibm-blockchain-platform-prod
NAME                                                READY   STATUS
RESTARTS   AGE
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   0/5
ContainerCreating   0          6s
```

   – If no pods are found, you see the following message: `No resources found.` Run through this troubleshooting logic to seek a solution:

a. Did the deployment show up in step 2 above, and now you get the no pods (`No resources found.`) message?
b. If yes, check for problems with the pod security policy as follows:

   i. Run the command that is shown in Example 4-17. You might see a similar error.

Example 4-17 kubectl get events

```
kubectl get events
2m51s       Warning   FailedCreate   ReplicaSet   Error creating: pods
"ibp-auto-deploy-team00-ibp-console-6d45d47868-" is forbidden: unable to
validate against any pod security policy
```

   This error means that the pod security policy is not giving enough permissions.

   ii. Review the pod security policy, clusterrole, and rolebinding steps from these topics:

- 4.1.2, “Setting up role-based access control (RBAC) roles for blockchain [1x per cluster only]” on page 131
- “Setting up namespace for installation” on page 143” from Section 4.1.4, “Manual console installation” on page 143.

   iii. Consider whether this your scenario:
   Your deployment exists, your pod does not exist, and `kubectl get events` does not show a pod security policy error like the one in Example 4-17

   iv. If this state is your scenario, make sure that you have accessed the correct namespace. Also, run the `kubectl get pods` command alone, with no parameters. If the problems persist, visit Section 4.5, “Troubleshooting the installation” on page 194 for additional help.

4. Run the following command to monitor the status of the pod:

   kubectl get pods -l app=ibm-blockchain-platform-prod --watch

   Example 4-18 shows pod status from the beginning. Depending on when you started the **watch** command, you might not see every stage that is listed in Example 4-18.

Example 4-18 watch command

```
kubectl get pods -l app=ibm-blockchain-platform-prod --watch
NAME                                                READY   STATUS    RESTARTS   AGE
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   0/5     Pending   0          0s
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   0/5     Pending   0          0s
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   0/5 ContainerCreating 0      0s
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   0/5     Running   0          18s
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   1/5     Running   0          26s
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   2/5     Running   0          29s
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   3/5     Running   0          29s
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   4/5     Running   0          30s
ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   5/5     Running   0          35s
```

Table 4-5 on page 156 describes the status messages.

Table 4-5 Pod status messages

Status	Description
Pending	The containers are waiting for something to start. This is most likely due to a persistent volume claim waiting to be bound to a persistent volume (i.e. dynamic provisioner has to go to work to do this). Depending on how fast storage is provisioned, this could take seconds or minutes. Generally, the GlusterFS storage can take up to a few minutes while the nfs storage should be done in a few seconds. **Note:** A performance update for GlusterFS has been released that should improve GlusterFS performance. Currently, it can take a few minutes without the update.
ContainerCreating	Kubelet is interfacing with the container runtime (through the container runtime interface [CRI]) to cause creation of the containers for the pod. This operation can take up to a few minutes while images are pulled from the appropriate docker repository (especially if the images are not on the node).
Running	All containers in the pod are in the running state. This is what we like to see.
Error	One or more of the containers in the pod have encountered an error state. This might be a momentary lapse that is fixed by restarting the container, but usually this is something that the user must debug.

a. Wait for the blockchain console pod to move through its phases (STATUS) until the **Running** status. Then, wait for all the containers to enter the ready state (5/5):

- If successful, go to Section 4.2.2, “Initializing blockchain console for other users” on page 167.
- If you see the **Error** status, or it has been over 2 minutes on a given phase, go to the next step.

b. Run the following command to check the details of events in your namespace:

`kubectl get events`

This command displays the events for the resources that are present in the namespace. The output has a time stamp, so look for events that are recent. **Normal** events are normal for the cluster. **Warning** events indicate that something might require corrective action. To look at warning events in particular, run this command:

```
kubectl get events --field-selector type=Warning
```

The warnings from our failed (and later fixed) deployment are shown in Example 4-19.

Example 4-19 The warnings from our failed (and later fixed) deployment

```
kubectl get events --field-selector type=Warning
LAST SEEN   TYPE      REASON         KIND         MESSAGE
52m         Warning   Unhealthy      Pod          Readiness probe failed: dial tcp 10.1.153.123:3000:
connect: connection refused
52m         Warning   Unhealthy      Pod          Readiness probe failed: dial tcp 10.1.153.123:8383:
connect: connection refused
52m         Warning   Unhealthy      Pod          Liveness probe failed: dial tcp 10.1.153.123:8383:
connect: connection refused
53m         Warning   Unhealthy      Pod          Readiness probe failed: dial tcp 10.1.153.117:3000:
connect: connection refused
53m         Warning   Unhealthy      Pod          Liveness probe failed: dial tcp 10.1.153.117:3000:
connect: connection refused
51m         Warning   Unhealthy      Pod          Readiness probe failed: dial tcp 10.1.153.104:8383:
connect: connection refused
```

```
51m         Warning   Unhealthy      Pod          Readiness probe failed: dial tcp 10.1.153.104:3000:
connect: connection refused
51m         Warning   Unhealthy      Pod          Liveness probe failed: dial tcp 10.1.153.104:8383:
connect: connection refused
48m         Warning   Unhealthy      Pod          Readiness probe failed: dial tcp 10.1.153.92:3000:
connect: connection refused
58m         Warning   FailedCreate   ReplicaSet   Error creating: pods
"ibp-auto-deploy-team00-ibp-console-6d45d47868-" is forbidden: unable to validate against any pod security
policy: [spec.containers[0].securityContext.capabilities.add: Invalid value: "NET_BIND_SERVICE": capability
may not be added spec.containers[0].securityContext.allowPrivilegeEscalation: Invalid value: true: Allowing
privilege escalation for containers is not allowed spec.containers[1].securityContext.capabilities.add:
Invalid value: "NET_BIND_SERVICE": capability may not be added
spec.containers[1].securityContext.allowPrivilegeEscalation: Invalid value: true: Allowing privilege
escalation for containers is not allowed spec.containers[2].securityContext.capabilities.add: Invalid
value: "NET_BIND_SERVICE": capability may not be added
spec.containers[3].securityContext.allowPrivilegeEscalation: Invalid value: true: Allowing privilege
escalation for containers is not allowed spec.containers[4].securityContext.capabilities.add: Invalid
value: "NET_BIND_SERVICE": capability may not be added spec.containers[4].securityContext.capabilities.add:
Invalid value: "CHOWN": capability may not be added spec.containers[4].securityContext.capabilities.add:
Invalid value: "DAC_OVERRIDE": capability may not be added
spec.containers[4].securityContext.capabilities.add: Invalid value: "SETGID": capability may not be added
spec.containers[4].securityContext.capabilities.add: Invalid value: "SETUID": capability may not be added]
```

The readiness and Liveness Probe failures show a normal status, because the pod came up after being deleted. However, we needed to fix the **FailedCreate** warning because of lack of access to a Pod Security Policy with the special permissions that the container requires to run.

Note: This command might display old events that are no longer relevant, so make sure to consider the time stamp of all events.

If you see a particular event that causes problems, refer to Section 4.5, “Troubleshooting the installation” on page 194.

5. Check status for all resources and describe particular resources:
   a. See all resources by running this command:

      kubectl get all -l app=ibm-blockchain-platform-prod

      The output should look similar to Example 4-20.

Example 4-20 See all resources

```
kubectl get all -l app=ibm-blockchain-platform-prod

NAME                                                  READY   STATUS    RESTARTS   AGE
pod/ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn   5/5     Running   0          74m

NAME                                                TYPE       CLUSTER-IP    EXTERNAL-IP  AGE     PORT(S)
service/ibp-auto-deploy-team00-ibp-console-optools   NodePort   10.0.67.247   <none>       7h
3000:30003/TCP,3001:30004/TCP

NAME                                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ibp-auto-deploy-team00-ibp-console   1/1     1            1           7h

NAME                                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/ibp-auto-deploy-team00-ibp-console-6d45d47868   1         1         1       7h
```

6. You can describe particular resources with this command:

   ```
   kubectl describe <resource> -l <selector>
   ```

   For example, to describe the blockchain console to see how each container is doing use this command:

   kubectl describe pod -l app=ibm-blockchain-platform-prod

   You can also look at a resource by using either of these commands:

   ```
   kubectl describe <resource_full_name>
   ```

   or:

   ```
   kubectl describe <resource> <resource_name>
   ```

   Using the output of Example 4-20, the command syntax would look like these examples:

   ```
   kubectl describe pod/ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn
   ```

   or:

   ```
   kubectl describe pod ibp-auto-deploy-team00-ibp-console-6d45d47868-mvcwn
   ```

   depending on your preference.

7. To see logs of all blockchain console containers run this command:

   ```
   kubectl logs --all-containers -l app=ibm-blockchain-platform-prod
   ```

   To see logs of a particular container use:

   ```
   kubectl logs -c <container_name> -l app=ibm-blockchain-platform-prod
   ```

   as in this example:

   ```
   kubectl logs -c optools -l app=ibm-blockchain-platform-prod
   ```

   To see logs for the prior 1-minute period (use m for minutes, h for hours, s for seconds)

   ```
   kubectl logs --all-containers -l app=ibm-blockchain-platform-prod --since=1m
   ```

8. After 5 minutes or more, you might be unable to find out why some resources in your deployment are still unavailable. In this case, visit the section 4.5, “Troubleshooting the installation” on page 194.
   Also, it might help to learn more about kubectl, by using the summary at this website:
   https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Verifying the console with the IBM Cloud Private user interface (UI)

Perform the following steps to verify the console installation by using the UI.

1. Check the helm release. Through the user interface, find the helm release for the blockchain console. You click that release in the dropdown menu that is shown in Figure 4-7.

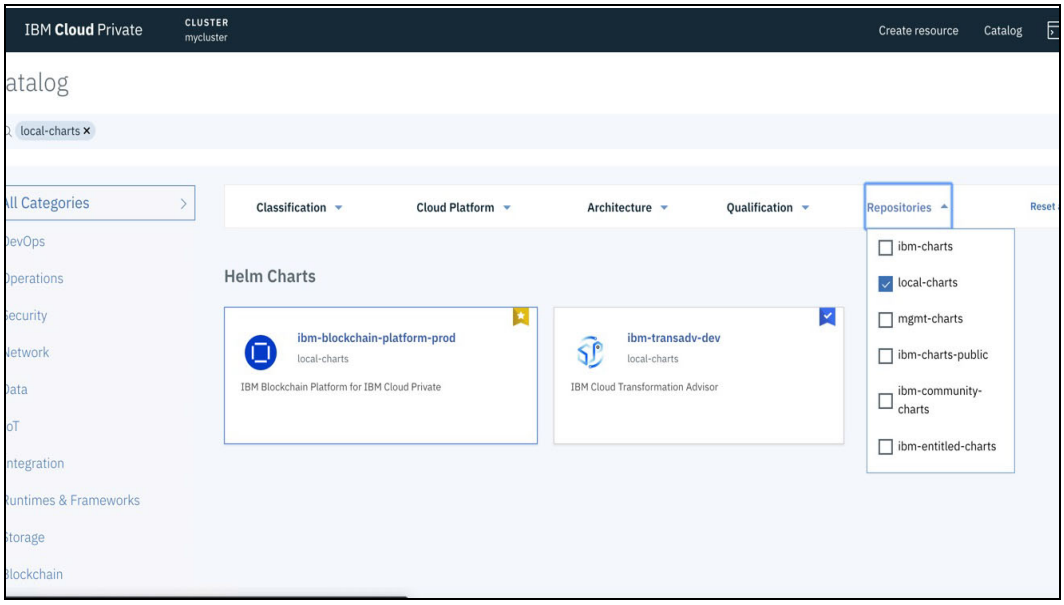


Figure 4-7 Verify the console installation using the UI -1

The main helm release page is displayed, with information about all of the components of the helm release. The main page should look similar to Figure 4-8 on page 159.

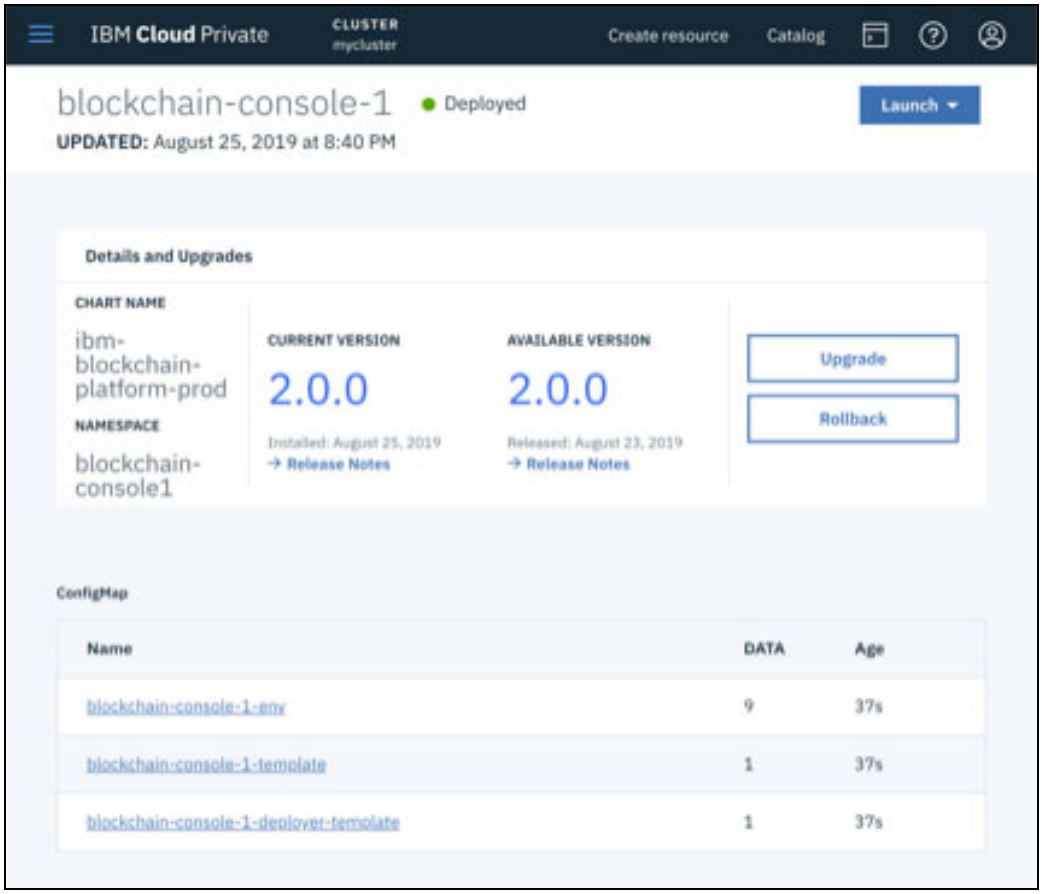


Figure 4-8 Verify the console installation using the UI -2

From this page, the user can see the release notes and navigate to the individual components of the helm release. (Navigate between components by clicking the blue links under each section).

2. Get the access details for blockchain. At the bottom of the helm release page are notes. These notes include the access details for the blockchain console and proxy when it is up and running (in other words, the deployment is available). The notes for a `blockchain-console-1` helm release are shown in Figure 4-9.

Figure 4-9 Verify the console installation using the UI -3

Make note of these values to use in the section 4.2.2, “Initializing blockchain console for other users” on page 167 after the blockchain console deployment is available. For now, follow on to step 3.

3. Check the deployment.
To check on the deployment, first look at the deployment section of the helm release. In Figure 4-10 on page 161, the deployment is already available so it would make sense to move on to 4.2.2, “Initializing blockchain console for other users” on page 167. However, in this case the screen capture was taken long after successful installation of the helm release. In most cases, the deployment will not be available yet. If this is the case for you, the best next step is to visit the deployment by clicking the link for the blockchain console deployment under the **Deployment** section.

Tip: Use **Right-click** → **Open** new tab (instead of the typical left click), so that the current helm releases tab remains open for future reference.

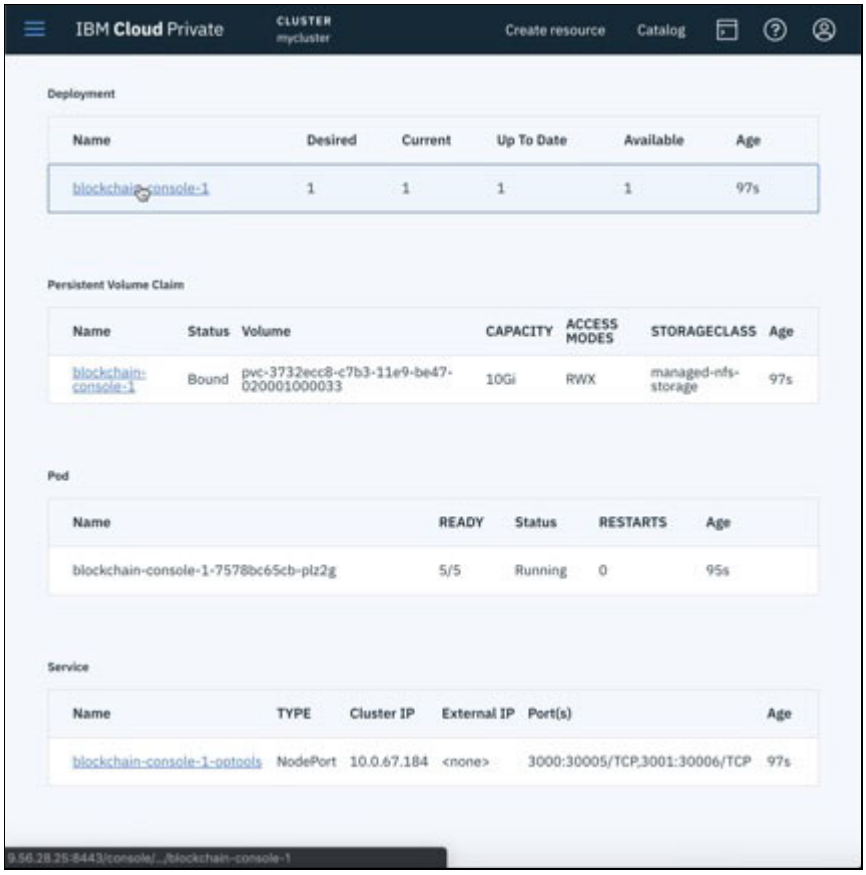


Figure 4-10 Verify the console installation using the UI -4

The deployment console illustrates a number of the different deployment values including its availability, which Figure 4-11 points out with an arrow.

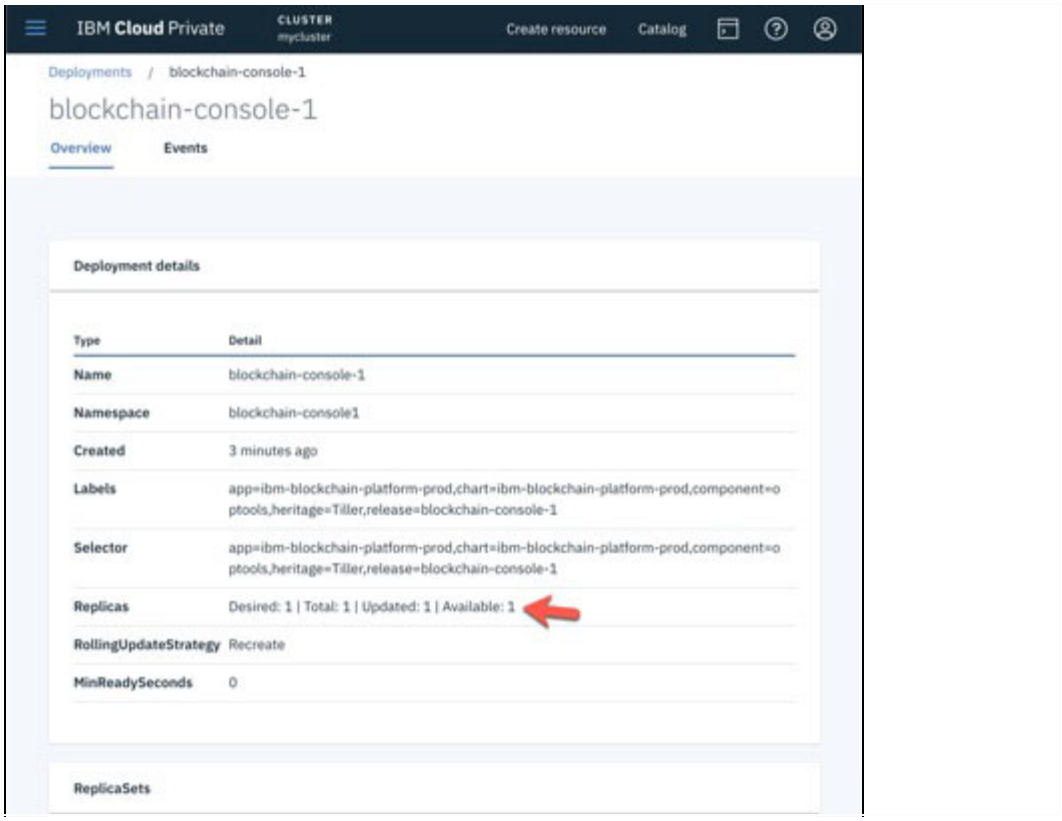


Figure 4-11 Verify the console installation using the UI -5

In the **Replicas** section of the deployment details, the red arrow points to the display that shows that the deployment is available (Available=1). If you see this, go to Section 4.2.2, “Initializing blockchain console for other users” on page 167. If your deployment is still not available, it is time to check the pod.

4. Scroll to the bottom of the deployment page until you see the pod. In Figure 4-12 on page 163, all the containers in the pod are running (in other words, the pod is running) and all five containers are ready (5/5). Click the pod name to see more about the individual pods including an overview, a container breakdown, and events.

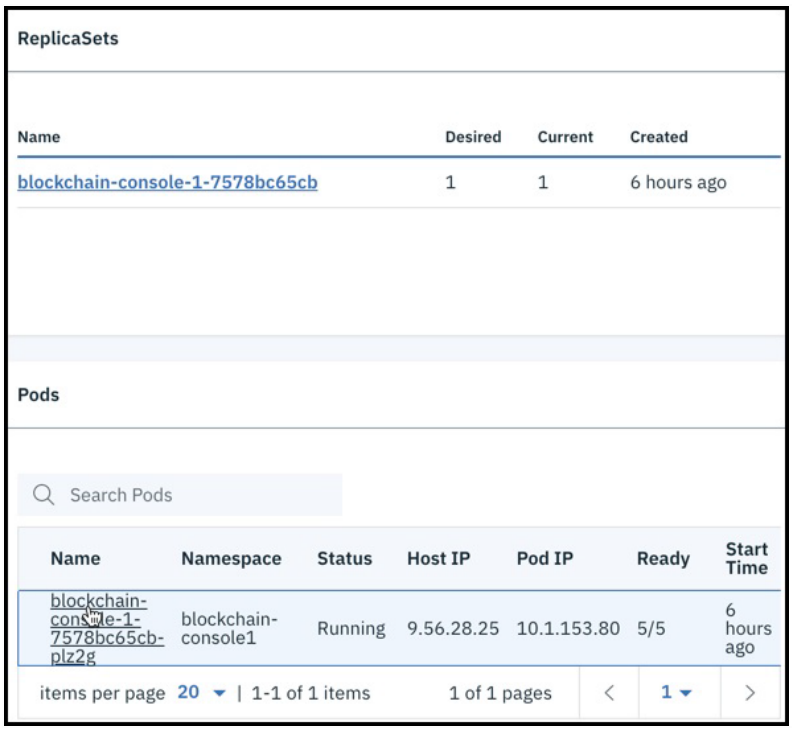


Figure 4-12 Verify the console installation using the UI -6

5. Review pod details as illustrated in Figure 4-13.
   A newly available data point is the `PodSecurityPolicy`. This setting should either match the created pod security policy of `ibm-blockchain-platform-psp` or be `ibm-privileged-psp`. If it is neither of these options, and you did not change names, make sure that everything in the deployment has gone according to plan. In this case, your pod has been provisioned so you are probably fine in terms of PodSecurityPolicy itself.

However, other changes might affect you later. So, it is good to confirm that you have completed all necessary steps in support of your organization's plan.

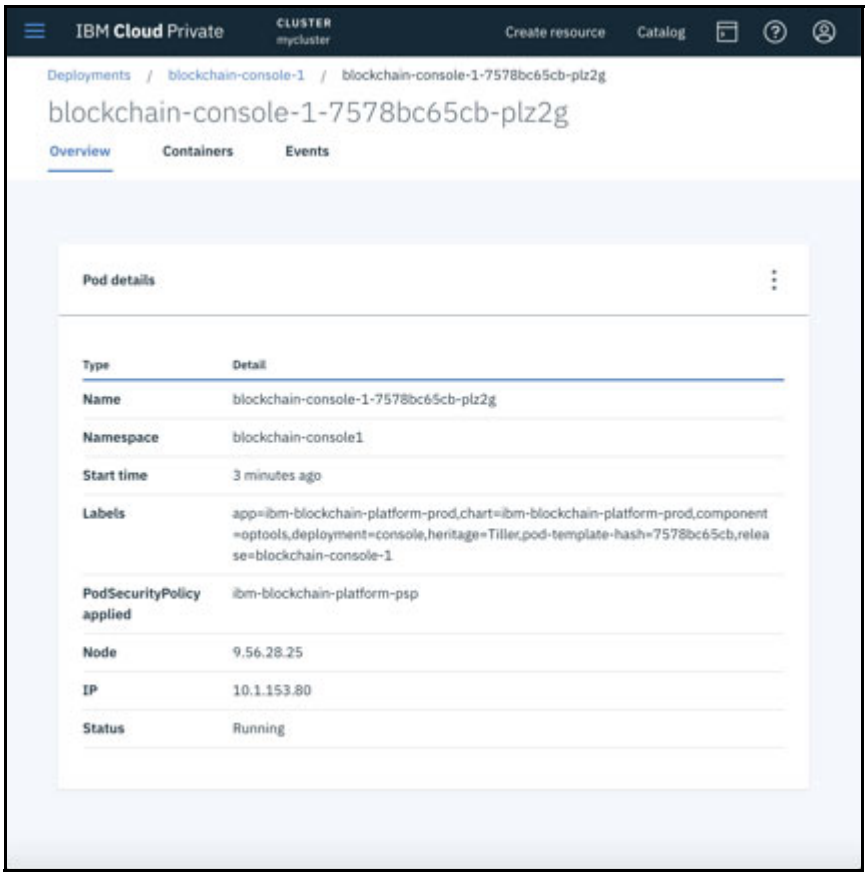


Figure 4-13 Verify the console installation using the UI -7

Containers tab: Another good information source is the Containers tab. This tab is especially useful if you see an error and want to know immediately the status of containers. Click the **Containers** (to the left of the highlighted overview) tab in the upper left to see a window similar to Figure 4-14.

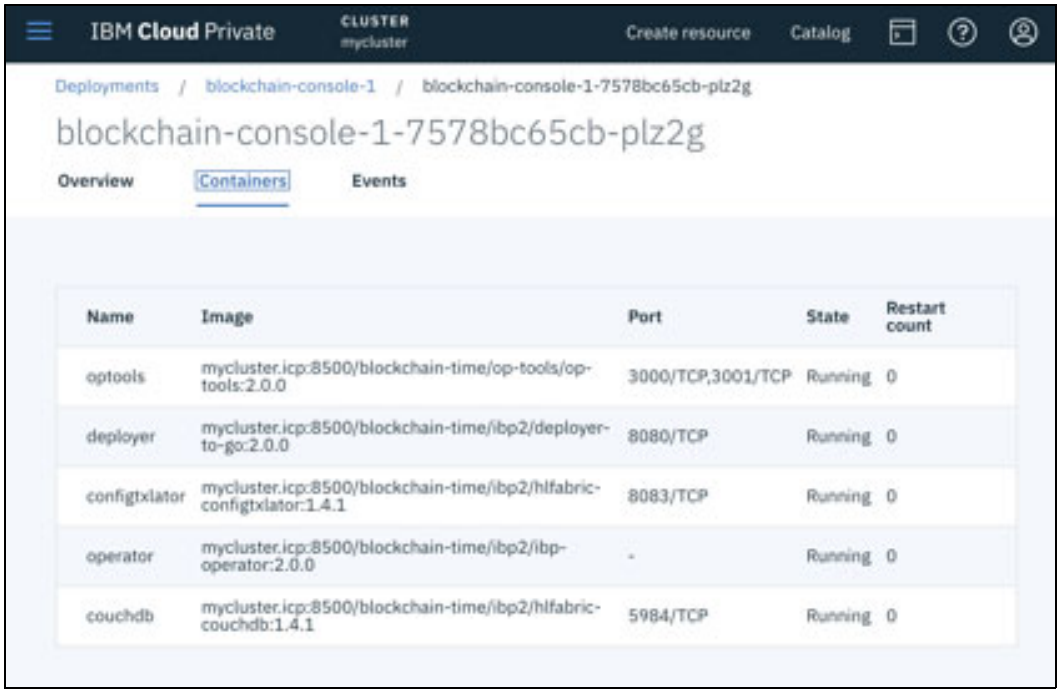


Figure 4-14 Verify the console installation using the UI -8

Events tab: Finally, pod events tell us what is happening with our pods. Events in general help us easily discover problems that are emerging in our cluster, with pod events specifically focusing on the pods. This Events section is the third section that is accessible from the pod view. Figure 4-15 on page 165 shows the events view.

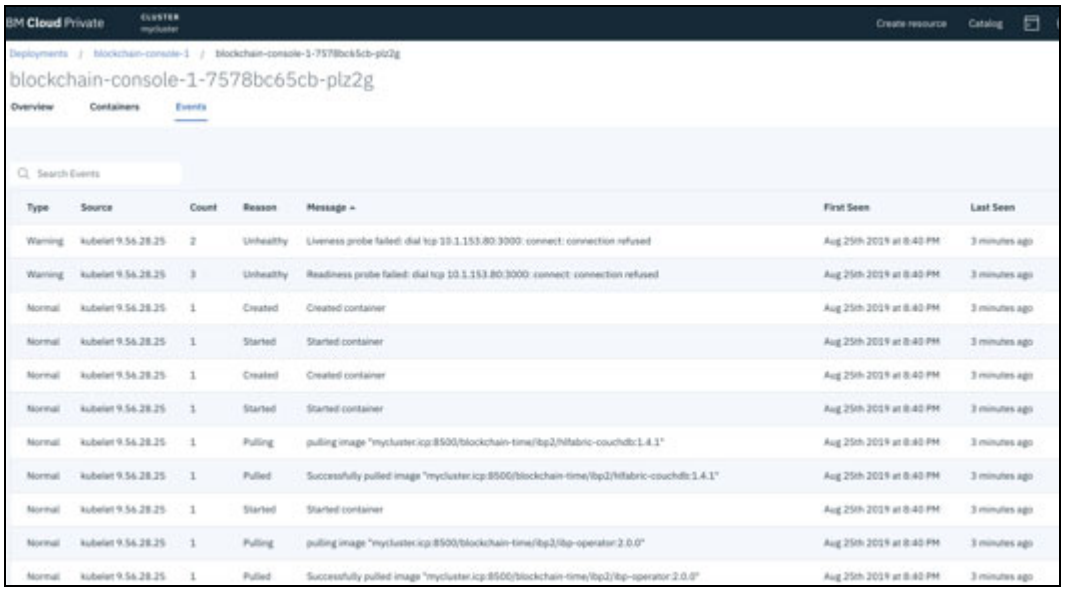


Figure 4-15 Verify the console installation using the UI -9

You see `Warning` and `Normal` events. `Normal` events document the activities of the pods and `Warning` events flag something potentially problematic. However, many `Warning` events — such as `Liveness and Readiness check` failures — are normal at first. A large number of such events in a small period of time is a cause for concern indicating, because

that indicates downtime of the application. For this reason, Kubernetes documents these as **Warnings**, along with many other events.

6. Check the PersistentVolumeClaims (storage) status. Blockchain is a persistent data base, so you must keep track of backing storage. To see the persistent volume claim for the helm release, click the name of the PersistentVolumeClaim item in the **Persistent Volume Claim** section. In Figure 4-16 it is named `blockchain-console-1`. The PersistentVolumeClaim details are displayed.

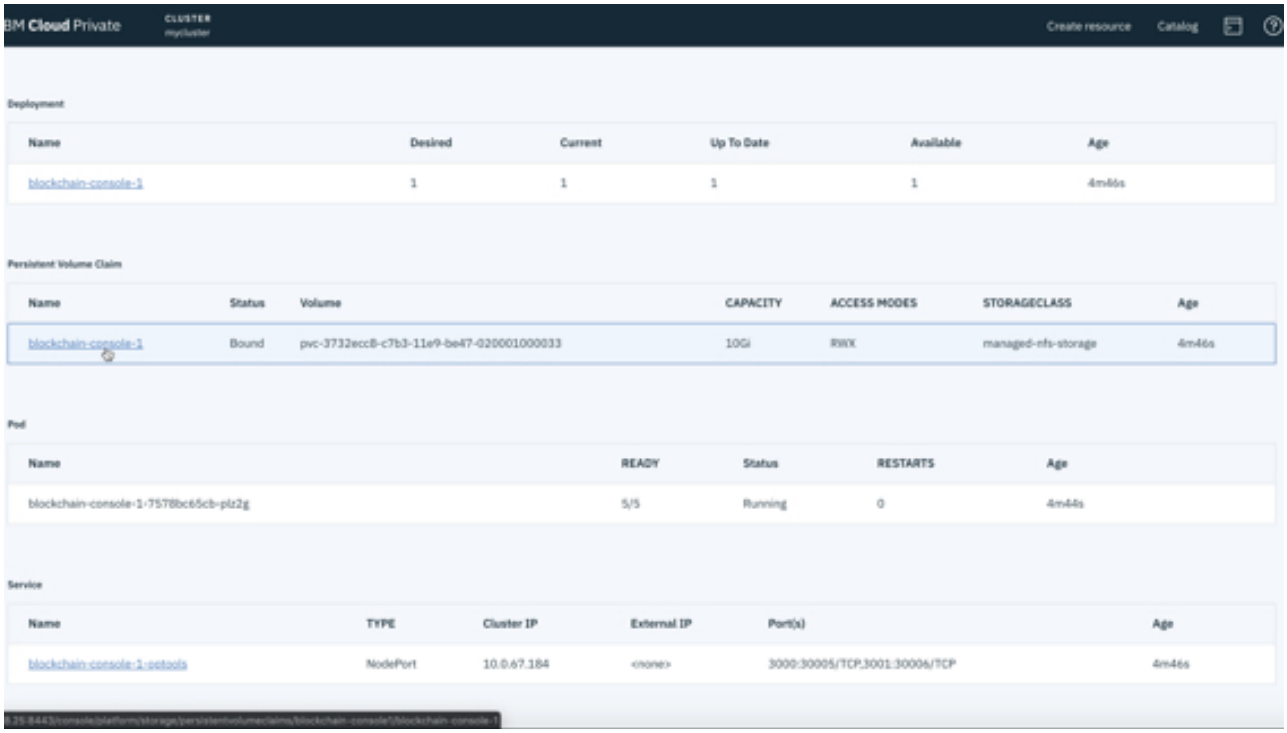


Figure 4-16 Verify the console installation using the UI -10

7. Click the **Events** tab to see the automated activities of the dynamic provisioner, as it creates the storage that is necessary for the blockchain console. The events are captured in Figure 4-17.

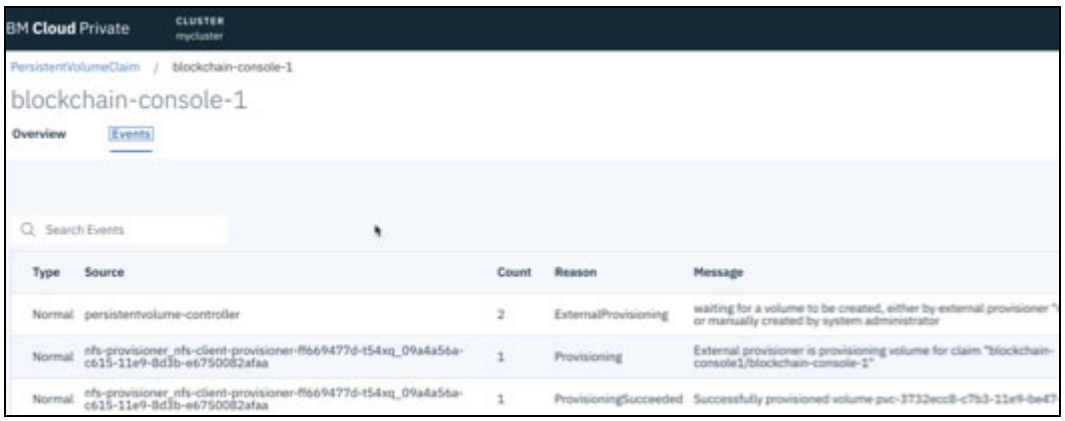


Figure 4-17 Verify the console installation using the UI -11

Often, a pod has a **Pending** status because it is waiting for storage provisioning. You can consult the events view in Figure 4-17 to see what is happening with this storage

provisioning and begin to diagnose an error. For further help with errors, see Section 4.5, “Troubleshooting the installation” on page 194.

4.2.2 Initializing blockchain console for other users

In this section, we show how to initialize blockchain console with users.

Log in to blockchain console

Perform the following steps to log in to blockchain console.

1. Accept the Certificates Proxy prompt, which is shown in Figure 4-18. Visit the URL for the proxy and accept certificates if they appear. Unless you opted to use your own TLS certificates, your use of the default self-signed certificates require that you accept the security risk on your browser.

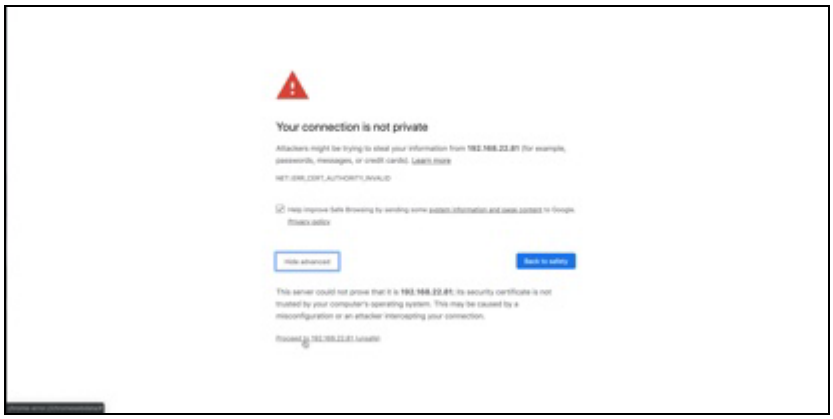


Figure 4-18 initializing blockchain console with users -1

After you do this for the proxy, the **up** message is displayed, as you see in Figure 4-19.

Figure 4-19 initializing blockchain console with users -2

2. **Accept Certificates console.** Go to the IBM Blockchain Platform console main page and accept any certificate message that you see there, too. (Depending on your web browser, you might not see a message.)
3. **Log in to the console.** You must enter your username and password in the console. The username is the email address that you entered. The password is the password that you entered in the Kubernetes secret before you installed the console. You can find the secret name first with this command:

```
kubectl get secrets --field-selector=type=Opaque
```

Then, extract the password from the secret by using this command:

```
kubectl get secret <secret_name> -o jsonpath='{.data.password}' | base64
--decode && echo
```

We do this on our cluster in Example 4-21.

Example 4-21 Secret name and password

```
kubectl get secrets --field-selector=type=Opaque
NAME            TYPE     DATA   AGE
ibp-ui-secret   Opaque   1      10h

kubectl get secret ibp-ui-secret -o jsonpath='{.data.password}' | base64
--decode && echo
I-Live-4-SECurity-OhYEAH
```

Note: The password that your **`get secret`** command displays will be the default password for all users you initialize in the console, unless you change it later. All users are forced to change their password when they first log in. For this example deployment, the default password is visible in Example 4-21 and is as follows: `I-Live-4-SECurity-OhYEAH`

Now, you use these credentials for the login, shown in Figure 4-20.

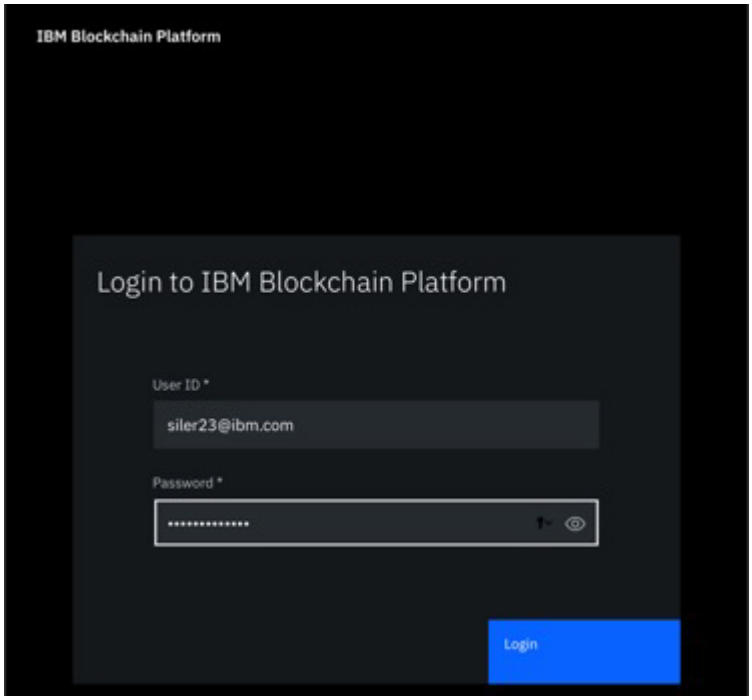


Figure 4-20 initializing blockchain console with users -3

4. Reset your password. After logging in the first time, the IBM Blockchain Platform for Multicloud console immediately requests that you reset your password and log in again

before you can access the platform itself. After you reset your password, you log in and see the formal greeting from the welcome screen that is shown in Figure 4-21.

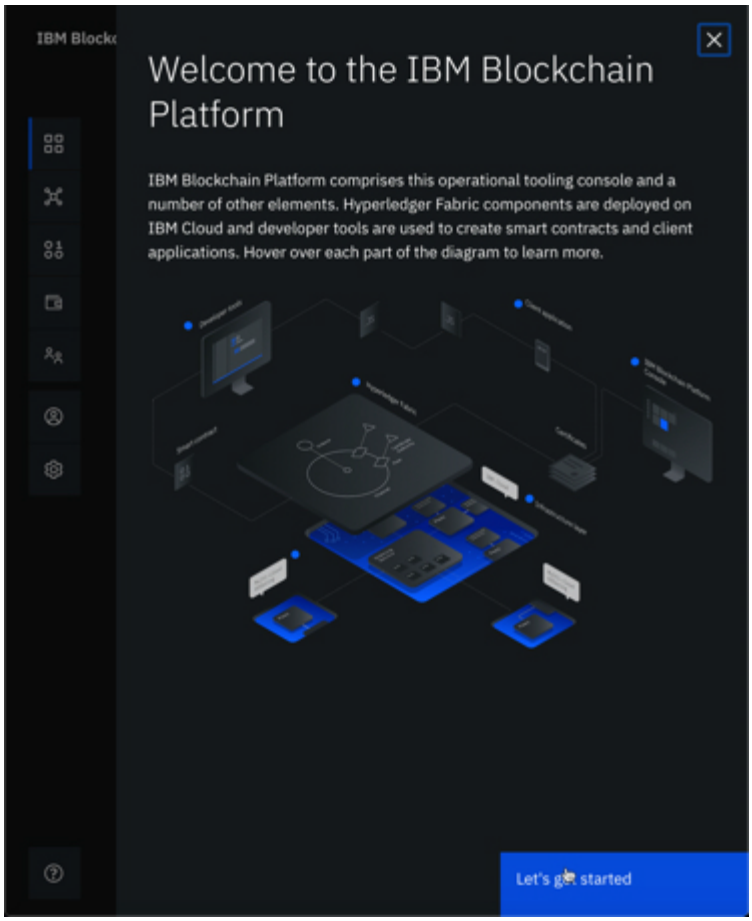


Figure 4-21 initializing blockchain console with users -4

5. Click **Let's get started** to enter the IBM Blockchain Platform for Multicloud console for the first time.

Managing console - Adding users and more

1. Go to the Users section by clicking the round icon of a person on the lower left and highlighted in blue in Figure 4-22 on page 170. Here, administrators manage existing users, add new users, and update the configuration of couchdb. (There, you can change

the default password for the network, which is the password that everyone starts with before they reset their password.).

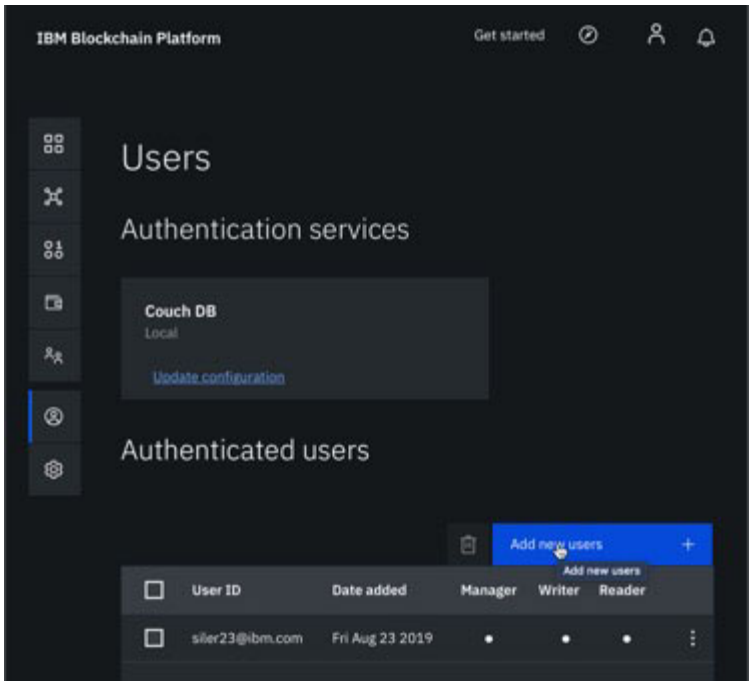


Figure 4-22 Managing console -1

2. Add new users with their email addresses by first clicking **Add new users** as shown in Figure 4-22 on page 170. Then, add the new users' email addresses with one of the available roles. In Figure 4-23, the administrator enters the email addresses of several users, with same role. The UI also displays the specific privileges that each role gives to a user.

Note: The default password for all these users is whatever the default password was originally set to. This should be the same initial password that you had before you were forced to change it at login, unless you changed the **couchdb** configuration Figure 4-22 on page 170. This will also be the password for reset passwords that are sent to users who forget their password. Users must change their passwords immediately upon entering the console for the first time.

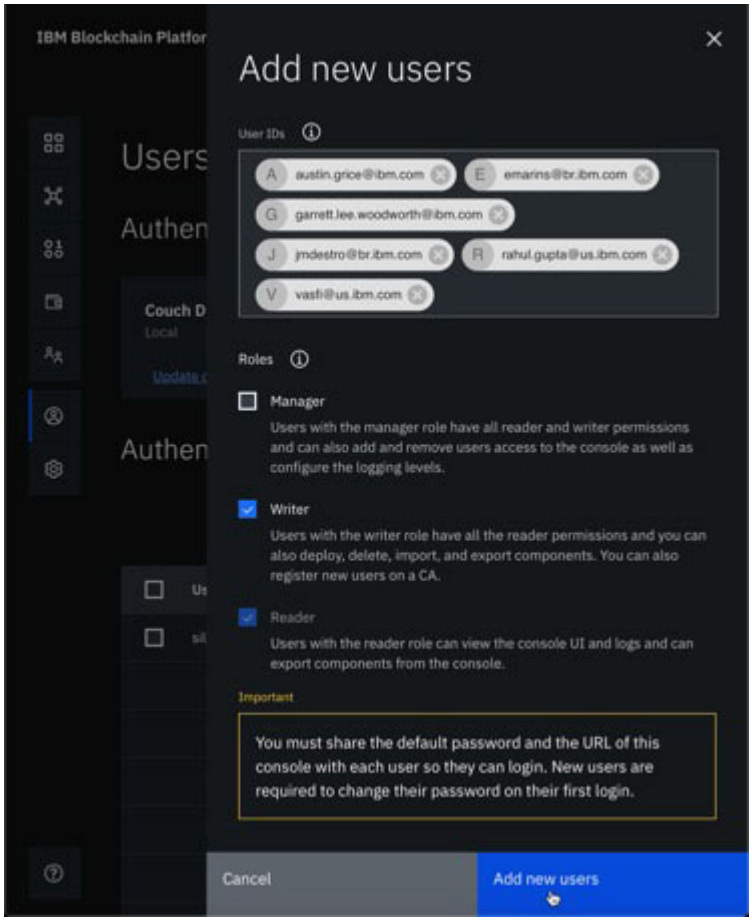


Figure 4-23 Managing console -2

Figure 4-24 on page 172 shows the users that have been added.

Note for long-term reference: To operate components that are deployed by the console, you need both permission and the cryptographic material for that component. Currently, the console stores the cryptographic material for new components in the user's web browser. Thus, if they want to give someone else access to that component, they must export the credentials (from their wallet) and securely transport it to the other individual (and that individual can add them to their wallet). No action is required at this time, other than keeping this convention in mind as you manage the console.

For more details on the credential transferring process visit this website:
https://cloud.ibm.com/docs/services/blockchain?topic=blockchain-ibp-console-identities#ibp-console-identities-wallet

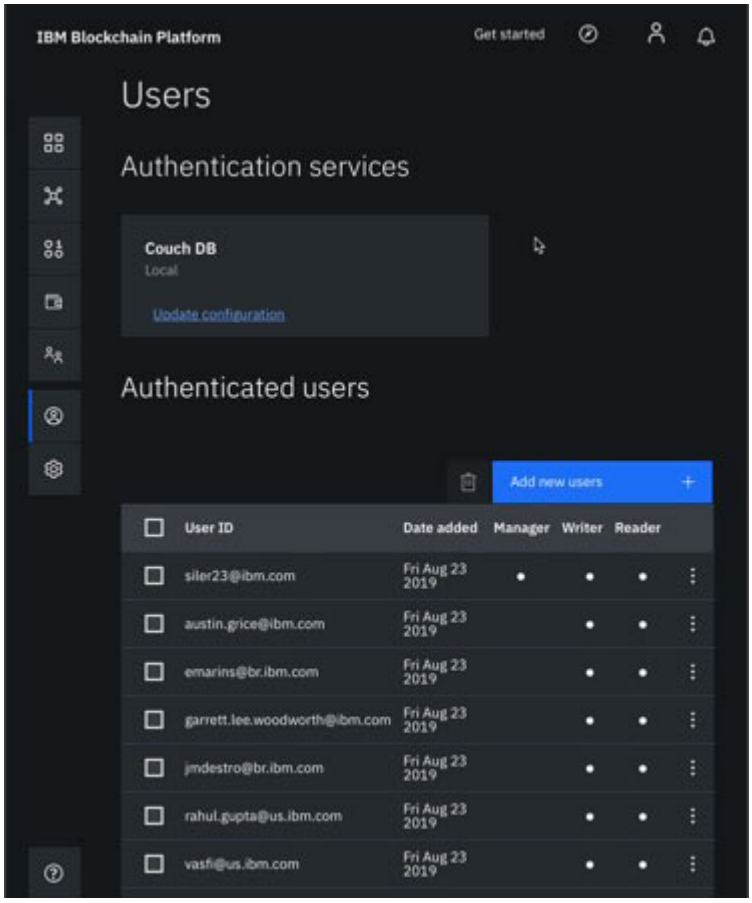


Figure 4-24 Managing console -3

3. **Console is initialized.** Start using the console with your group of users. Go to 4.3, “IBM Blockchain Platform installation” on page 173 to set up your blockchain network with the IBM Blockchain Platform for Multicloud.
4. **Monitor notifications.** The bell icon in the upper right of the console tracks events that might require your attention. Zero (0) notifications indicates that the console is running smoothly, as in Figure 4-25.

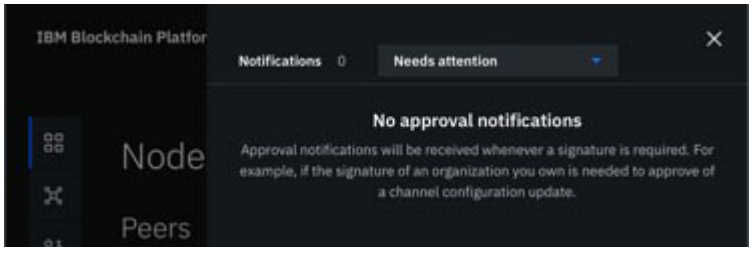


Figure 4-25 Managing console -4

For more information about managing the console, see this section of the IBM Cloud documentation:
https://cloud.ibm.com/docs/services/blockchain?topic=blockchain-console-icp-manage

5. **Further training.** To access the “Getting Started” tutorials, click **Get started** in the upper right of the UI (See Figure 4-26) to access links to training with the IBM Blockchain Platform.

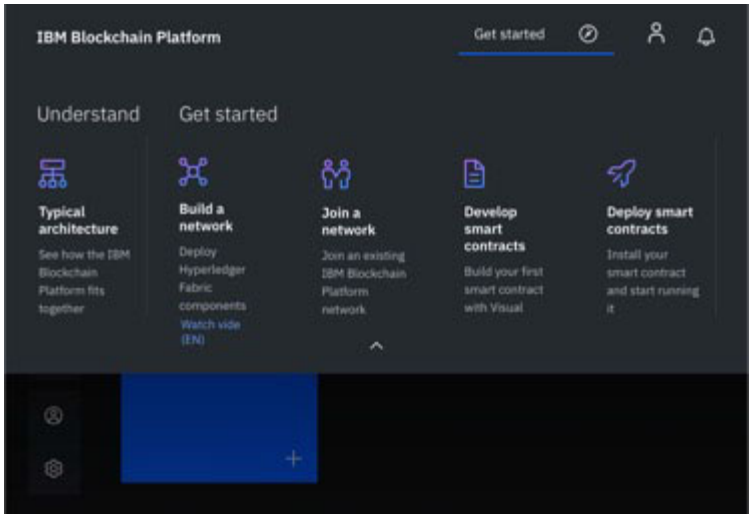


Figure 4-26 Managing console -5

4.3 IBM Blockchain Platform installation

IBM Blockchain Platform is a *blockchain-as-a-service* offering that enables you to develop, deploy, and operate blockchain applications and networks

The IBM Blockchain Platform is highly customizable. For the resources in your Kubernetes cluster, you can use the console to deploy components in an endless array of configurations. Figure 4-27presents an example of a business network that is composed of two

organizations, with one peer each, an ordering service, the peer organization and ordering service respective CAs and one channel.

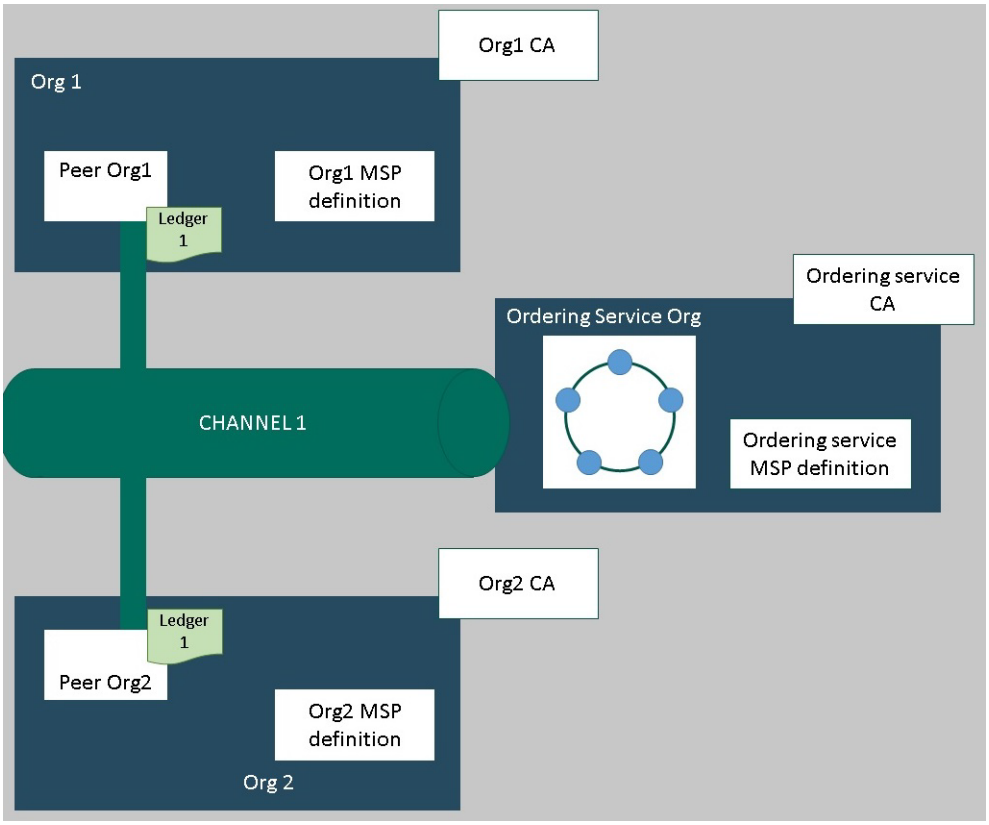


Figure 4-27 Example of business network and components

This configuration is sufficient for two purposes:

- Testing applications and smart contracts.
- As a guide for building components and joining production networks that suit your needs.

The network contains the following components:

- **Peer organizations**: Organizations that need to transact with each other. Represented in Figure 4-27 as Org 1 and Org 2.
- **Ordering service organization**: In a distributed ledger, the peers and ordering service should be part of separate organizations. Therefore, a separate organization is created for the ordering service. Among other things, an ordering service orders the blocks of transactions that are sent to the peers to be written to their ledgers and become the blockchain.
- **Certificate authority (CA)**: The node that issues certificates to both the users and the nodes that are associated with an organization. It's a best practice to deploy one CA per organization. These CAs also create the definition of each organization, which is encapsulated by a Membership Service Provider (MSP). A TLS CA is automatically deployed together with each organization CA. The TSL CA provides the TLS certificates that are used for communication between nodes.
- **Ordering service**: Either a one-node ordering service or a crash fault tolerant five-node ordering service. The ordering service uses an implementation of the Raft protocol, for both one- and five-node configurations. One-node ordering service is suitable for development and testing, while five-node ordering service provides crash fault-tolerance

and is suitable for production. Currently, only one ordering service organization per ordering service is supported, regardless of the number of ordering nodes associated with that organization. This ordering service adds peer organizations to its "consortium", which is the list of peer organizations that can create and join channels. If you want to create a channel that has organizations that are deployed in different clusters — which is how most production networks are structured — the ordering service admin must import a peer organization that has been deployed in another console into their console. This action allows the peer organization to join the channel that is hosted on that ordering service.

- **Peers**: Distributed peers that maintain the ledger. In Figure 4-27, Ledger x is the ledger. These peers are deployed by using Couch DB as the state database in a separate container that is associated with the peer. This database holds the current value of all "states" (as represented by key-value pairs), also known as world-state. The blockchain, the list of transactions, is stored locally on the peer.
- **Channel**: An area that allows sets of organizations to transact without exposing their data to organizations that are not members of the channel. Each channel has its own ledger, which is collectively managed by the peers that are joined to that channel. The smart contract is instantiated on the channel that the organizations can use to transact.

Figure 4-28 depicts the steps that are required to build a new blockchain network from scratch.

Figure 4-28 Steps to build a fresh blockchain network in IBM Blockchain Platform

4.3.1 Creating peer organizations

For each organization that you want to create with the console, you deploy at least one CA. A CA is the node that issues certificates to all network participants (peers, ordering services, clients, admins, and so on). These certificates, which include a signing certificate and private key, allowing network participants to communicate, authenticate, and ultimately transact. These CAs create all the identities and certificates that belong to your organization. They also define the organization itself. Then, you use those identities to deploy nodes, create admin identities, and submit transactions.

1. Navigate to the **Nodes** tab on the left and click **Add Certificate Authority**. The side panels allow you to configure the CA that you want to create and the organization that this CA will issue keys for.
2. Select the option **Create a Certificate Authority**, then click **Next**.
3. In the second side panel, give your CA a **display name**.
4. On the next panel, give your CA admin credentials by specifying a **CA administrator enroll ID** and a secret.
5. Set the resource allocation for the node.
6. Review the Summary page and then, click **Add certificate authority**.

After the CA is successfully created and it is running, a green box appears in the tile, as shown in Figure 4-29.

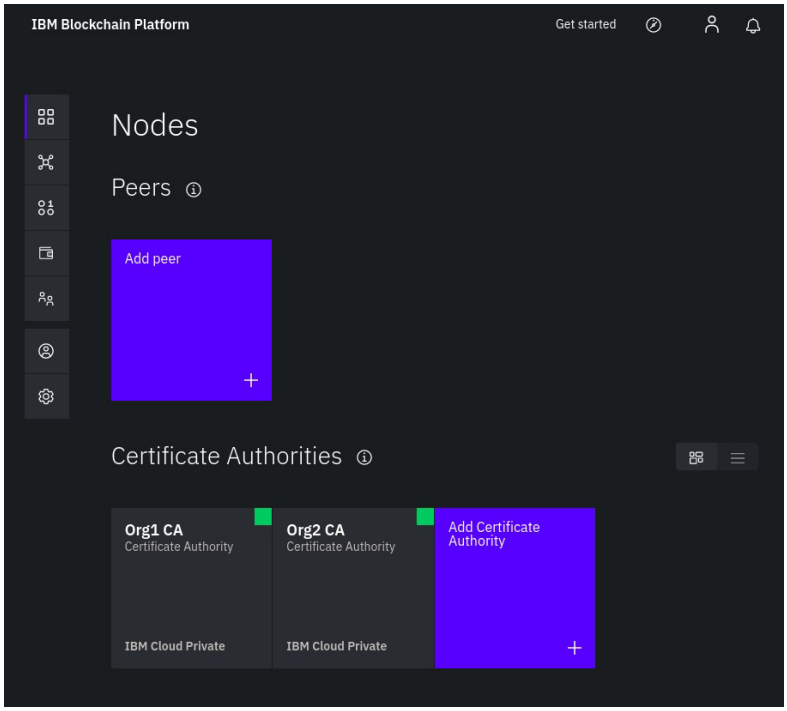


Figure 4-29 CAs created successfully for Org 1 and Org 2

Registering identities in a CA

Each node or application that you want to create needs a certificate and private key to participate in the blockchain network. You also need to create admin identities for these nodes and applications so that you can manage them from the console. You need at least two identities:

- **An organization admin**: This identity allows you to operate nodes by using the platform console.
- **A peer identity**: This is the identity of the peer itself. Whenever a peer performs an action (for example, endorsing a transaction), it signs by using its certificate.

After the CA is running, as indicated by the green box in the tile, generate these certificates by completing the following steps. (These organization names are only examples. Use names that match the needs of your organization.)

1. Click **Org1 CA** and ensure that the **admin** identity that you created for the CA is visible in the table. Then, click **Register User**.
2. First, register the organization admin:
   a. Give an **Enroll ID** of `admin` and a secret.
   b. Set the `Type` for this identity as `client`
   (Always register `admin` identities as `client` . In contrast, always register node identities as `peer`).
   The **Maximum enrollments** field is optional.
   c. Click **Next**.
3. Optionally, you can specify any attribute-based access control attributes for the user. For example, you can use this section to create another CA admin with the authority to register and enroll new identities.
   You can see a full list of available Fabric CA attributes in the *Registering a new identity* section of the *Fabric CA User's Guide* on this web page:

(https://hyperledger-fabric-ca.readthedocs.io/en/release-1.4/users-guide.html) When you are ready, click **Register User**.

4. Repeat this process for the identity of the peer (also using the Org1 CA). For the peer identity, select `peer` as the Type.
5. Repeat the steps for Org2 CA.

Note: These identities must be enrolled before they can be used. Enrollment for an organization admin happens during the creation of the MSP and the peer identity is enrolled during the creation of the peer.

Figure 4-30 shows the list of registered users of Org1 CA after the users are registered.

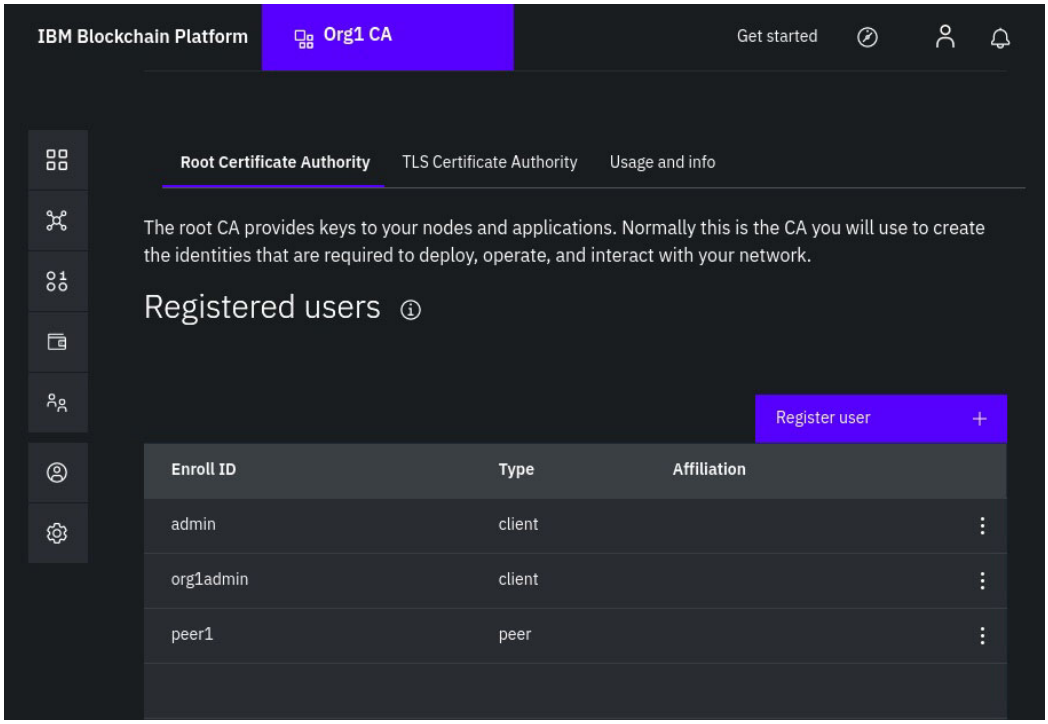


Figure 4-30 Registered users in Org1 CA

Creating peer organization MSP definition

After creating the peer's CA and using it to register identities for the admin and for the peer, we need to create a formal definition of the peer's organization, which is known as an MSP.

Note: Many peers can belong to an organization. It is not necessary to create a new organization every time that you create a peer.

During the process of creating the MSP, you enroll the admin identity and add it to the Wallet.

1. Navigate to the **Organizations** tab in the left navigation, and click **Create MSP definition**.
2. Give your MSP a display name and an MSP ID. You must follow the specifications about the limitations to this name that are mentioned in the tooltip.
3. Under **Root Certificate Authority** details, specify the CA that you used to register the identities in the previous step.

4. The Enroll ID and Enroll secret fields below are automatically populated with the enroll ID and secret for the first user that you created with your CA. However, using this identity would give your organization the same admin identity as your CA. For security reasons, this practice is not recommended. Instead, select the enroll ID that you created for your organization admin from the drop-down list and enter its associated secret. Then, give this identity a display name.
5. Click **Generate** to enroll this identity as the admin for your organization and export the identity to the Wallet. There, it can be used when you create the peer and create channels.
6. Click **Export** to export the admin certificates to your file system.

> **Note:** This identity is not stored in your console or managed by IBM. It is only stored in local browser storage. If you change browsers, you must import this identity into your Wallet to be able to administer the peer.

7. Click **Create MSP definition**.

   Figure 4-31 depicts the MSP after the organization definitions are in place.

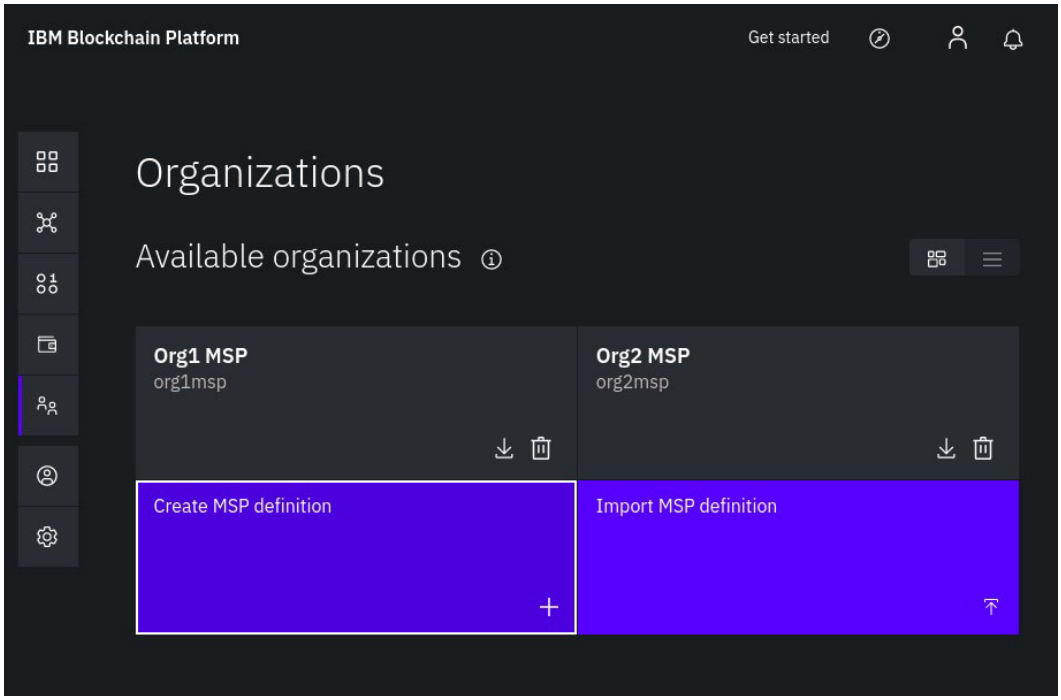


Figure 4-31 MSP definition for Org1 and Org2

After you create the MSP, you see the peer organization admin in your Wallet. You can access the Wallet by clicking the Wallet icon in the left navigation. See Figure 4-32.

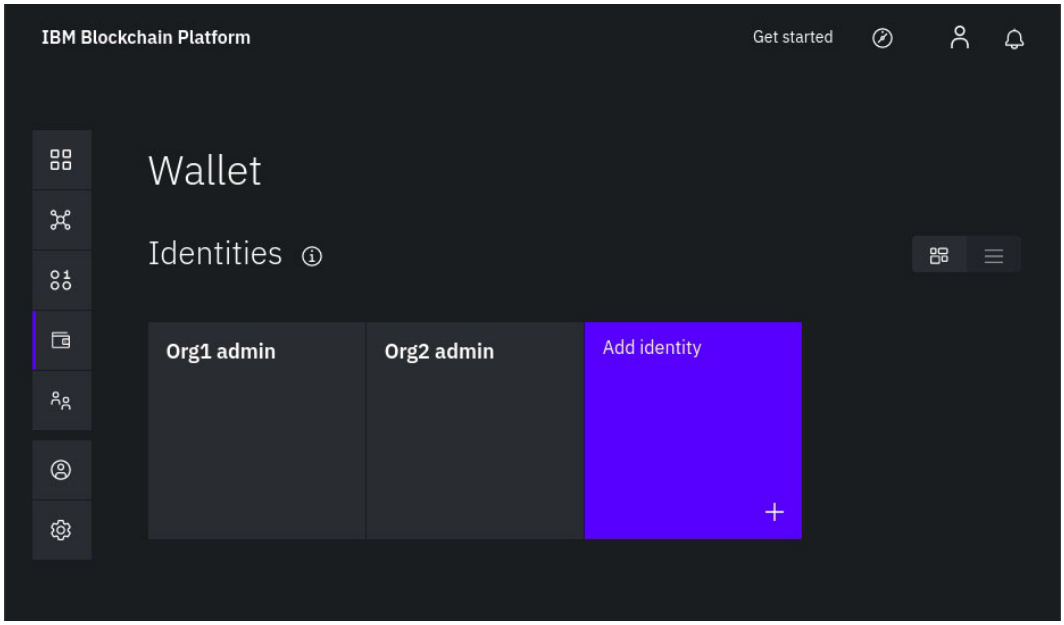


Figure 4-32 Wallet with two identities

4.3.2 Creating a peer

At this stage, you must create a peer for each organization.

Remember that organizations themselves do not maintain ledgers. Peers do. Organizations also use peers to sign transaction proposals and approve channel configuration updates. Having at least two peers per organization on a channel makes them highly available. For this reason, having *three* peers per organization joined to a channel is considered a best practice for production-level implementations. That way, you ensure high availability even while one peer is down for maintenance.

This section shows how to create one peer. You can create more peers to match your business requirements.

From a resource allocation perspective, you can join the same peers to multiple channels. The design of the peer ensures that the data from one channel cannot pass to another through the peer. However, because the peer stores a separate ledger for each channel, you must ensure that the peer has enough processing power and storage to handle the transaction and data load.

Use your console to perform the following steps:

1. On the **Nodes** page, click **Add peer**.
2. Ensure that the **Create a peer** option is selected. Then, click **Next**.
3. Give your peer a **Display name**. Choose to use certificates that are provided by an external CA or use a CA that is hosted by IBM. Click **Next**.

Note: If you choose to use certificates from an external CA, you must define an organization for this peer by building an MSP definition file. This file must include certificates from the external CA. Then, import that file into the Organizations tab before you proceed.

4. If you are using a CA that is hosted by IBM, take the following actions:
   On the next screen, select a CA you used to register the peer identity. Select the **Enroll ID** for the peer identity that you created for your peer from the drop-down list, and enter its associated **secret**. Then, select the appropriate MSP from the drop-down list, and click **Next**.
5. The next side panel asks for TLS CA information. When you created the CA, a TLSCA was created alongside it. This CA is used to create certificates for the secure communication layer for nodes. Therefore, select the **Enroll ID** for the peer identity that you created for your peer from the drop-down list. Then, enter the associated **secret**. The **TLS Certificate Signing Request (CSR)** hostname is an option for advanced users who want to specify a custom domain name. This name is used to address the peer endpoint.
6. On the next panel, you can configure resource allocation for the node.
7. In the last side panel, you can **Associate an identity** to make it the admin of your peer.
8. Confirm the summary of your changes, and click **Add peer**.

Peer creation might take 15 minutes or more. After the peer is successfully created and running, a green box is displayed in the tile, as shown in Figure 4-33.

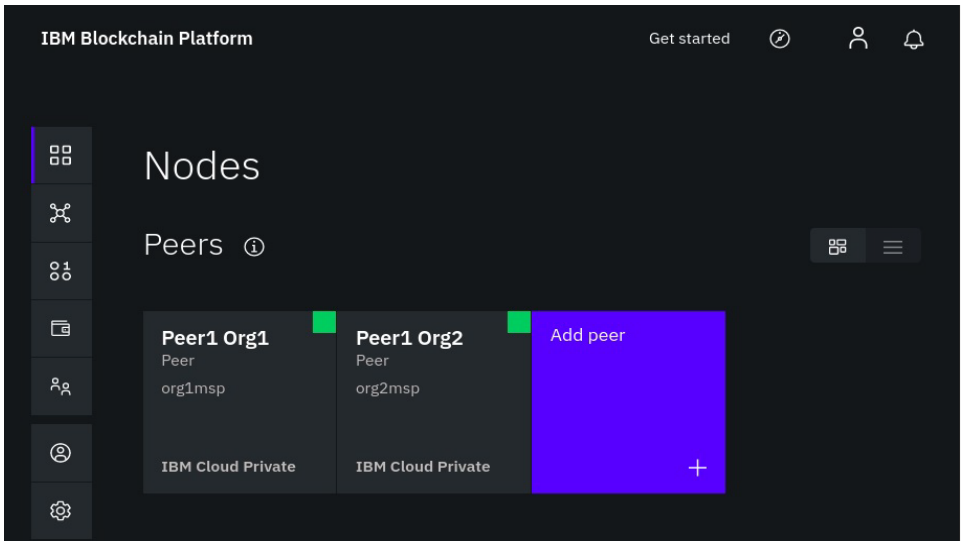


Figure 4-33 Peers created and running

4.3.3 Creating the ordering service

Distributed blockchains such as Ethereum and Bitcoin have no central authority to order transactions and send them out to peers. In contrast, the blockchain that the IBM Blockchain Platform is based on, *Hyperledger Fabric*, works differently. It features a node, or a cluster of nodes, that is called an *ordering service*.

The ordering service is a key component in a network because it fulfills some essential functions:

- Orders the blocks of transactions that are sent to the peers to be written to their ledgers.
- Maintains the ordering system channel, where the following components reside:
  - The consortium, which is a multi-tenancy vehicle. By design, a single ordering service can host multiple consortia.
  - The list of peer organizations that are permitted to create channels.
- Enforces the policies of the consortium or the channel administrators. These policies define everything from who gets to read or write to a channel, to who can create or modify a channel.

  Consider a policy scenario. When a network participant asks to modify a channel or consortium policy, the ordering service must run through this process:

  a. Confirm that the participant has the proper administrative privileges for that configuration update.
  b. Validate it against the existing configuration.
  c. Generate a new configuration.
  d. Relay it to the peers.

Just as with the peer, the following action is required before we can create an ordering service:
We must create a CA to supply the identities and the MSP of our ordering service organization.

In this release, distributed ordering services — in which multiple organizations contribute nodes to an ordering service — are not supported. Every ordering node in the ordering service is administered by a single organization.

The production-level ordering service that is available is a crash fault tolerant (CFT) ordering service based on an implementation of Raft protocol. Raft follows a “leader and follower” model, where a leader node is elected (per channel) and its decisions are replicated by the followers.

Currently, for ordering nodes, *five nodes* is the only available configuration that is crash fault tolerant. While it is possible to create a crash fault tolerant ordering service with as little as three nodes, this configuration incurs risk. For example, if one node goes down during a maintenance cycle, only two nodes would be left. If another node is lost during this cycle for any reason, only one node would be left. In that state — literally now a one-node ordering service, when you started with three — you no longer have a majority of nodes available. This majority is also known as a "quorum". When a quorum does not exist, no transactions can be pushed. *The channel ceases to function*.

In contrast, with five nodes you can lose two nodes and still maintain a quorum. This approach means that you can go through a maintenance cycle while you maintain high availability. Production networks should choose the five-nodes option, because, by definition, a one-node ordering service is not crash fault tolerant. Therefore, such a one-node service is suitable only for development and test networks.

Creating the ordering service organization

The process for creating a CA for an ordering service is identical to creating it for a peer.

1. Navigate to the **Nodes** tab and click **Add Certificate Authority**.
2. Select the option to **Create a Certificate Authority** then click **Next**.
3. Give this CA a unique **Display name**.

4. Enter the CA administrator **Enroll ID** and a **secret**.
5. Configure **Resource allocation** for the CA.
6. Review the **Summary** page, then click **Add certificate authority**.

As with the peer, advanced users might already have their own CA and not want to create a new CA by using the console. If your existing CA can issue certificates in `X.509` format, you can use your own external CA instead of creating a new one here.

Registering ordering service admin and ordering service node

After the CA is running, as indicated by the green box in the tile, generate these certificates by completing the following steps:

1. Click **Ordering Service CA** in the Nodes tab and ensure the admin identity that you created for the CA is visible in the table. Then, click **Register User**.
2. Register the organization admin by giving an **Enroll ID** and a **secret**. Then, set the **Type** for this identity as `client` (admin identities should always be registered as `client`, while node identities should always be registered by using the `peer` type). The **Maximum enrollments** field is optional. Click **Next**.
3. Optionally, you can specify any attribute-based access control attributes for the user. For example, you can use this section to create another CA admin with the authority to register and enroll new identities. You can see a full list of available Fabric CA attributes in the *Registering a new identity* section of the *Fabric CA User's Guide*. When you are ready, click **Register User.**
4. After the organization admin has been registered, repeat this same process for the identity of the ordering service. For the ordering service node identities, give an **Enroll ID**, a **secret**, and select `peer` as the **Type**. The **Maximum enrollments** field and the attributes for the user are both optional.

Figure 4-34 shows the registered organization admin and ordering service users. (The usernames that are shown are only examples. Names can be set to match the needs of your organization).

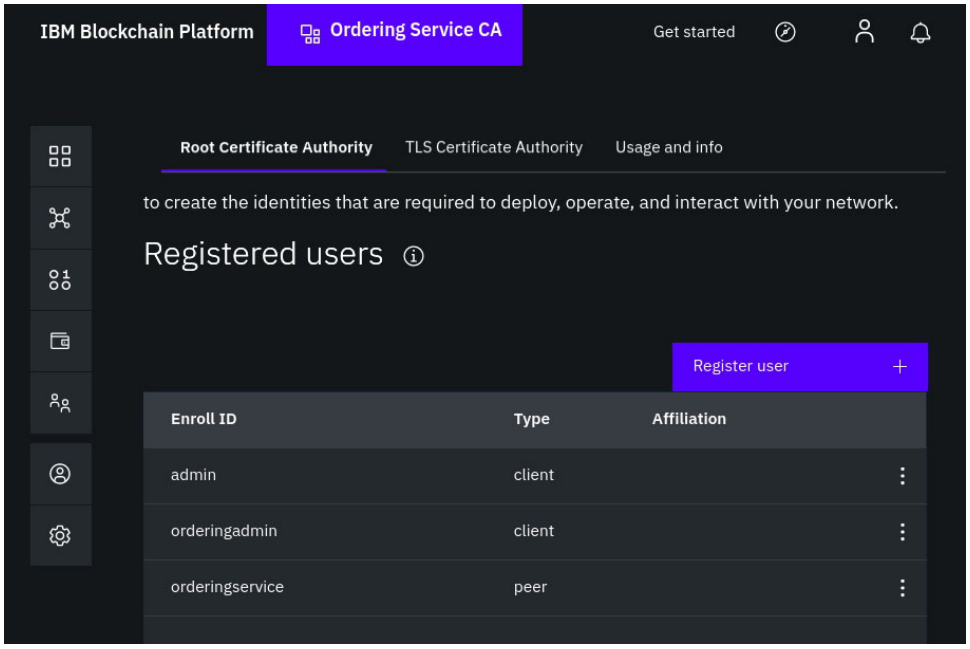


Figure 4-34 Ordering Service users created

Creating ordering service organization MSP definition

Create your ordering service organization MSP definition, and specify the admin identity for the organization. After you register the ordering service admin and ordering service users, you must create the MSP ID and enroll the admin user that was registered as the admin of the organization.

1. Navigate to the **Organizations** tab in the left navigation and click **Create MSP definition**.
2. Give the MSP definition a **Display name** and an **MSP ID**.
3. Under **Root Certificate Authority** details, select the Ordering Service CA created before.
4. The **Enroll ID** and **Enroll secret** fields below automatically populate with the enroll ID and secret for the first user that you created with your CA. However, using this identity would give your organization the same identity as your CA identity. For security reasons, this approach is not recommended. Instead, in the drop-down list select the enroll ID that you created for your organization admin, and enter its associated secret. Then, give this identity a **Display name**.
5. Click **Generate** to enroll this identity as the admin of your organization, and export the identity to the **Wallet**.
6. Click **Export** to export the admin certificates to your file system. As we said above, this identity is not stored in your console or managed by IBM. It is only stored in your browser. If you change browsers, you must import this identity to the other browser so that you can administer the ordering service there.
7. Click **Create MSP definition**.

Figure 4-35 shows the Ordering Service MSP that was created.

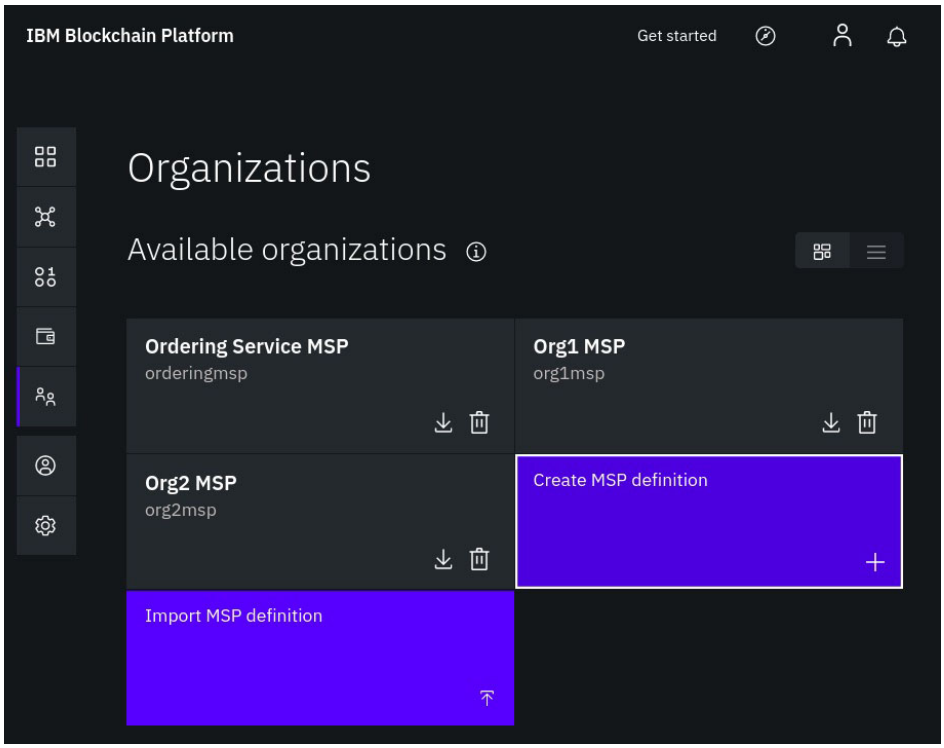


Figure 4-35 Ordering Service MSP created

Deploying ordering nodes

Now, deploy the ordering service in your organization.

1. On the **Nodes** page, click **Add ordering service**.
2. Confirm that the option to **Create an ordering service** is selected. Then, click **Next**.
3. Give your ordering service a **Display name**, and choose whether you want your ordering service to have one node (sufficient for development and testing) or five nodes (good for production). If you want to use an external CA, select it here.

   Note: To use an external CA, you must first build an MSP definition file that includes certificates from the external CA. Then, import that file into the Organizations tab.

   Click **Next**.
4. On the next panel, select Ordering Service CA as your CA. Then, select the **enroll ID** for the node identity that you created for your ordering service from the drop-down list, and enter the associated **secret**. Then, select your MSP from the drop-down list.
5. The next side panel asks for TLS CA information. When you created the CA, a TLS CA was created alongside it. This CA is used to create certificates for the secure communication layer for nodes. Therefore, in the drop-down list select the **enroll ID** for the ordering service identity that you created, and enter its associated **secret**. The TLS Certificate Signing Request (CSR) hostname is an option for those who want to specify a custom domain name that can be used to address the ordering service endpoint. Click **Next**.
6. Configure resource allocation for the node. The selections that you make here are applied to all five ordering nodes. If you want to learn more about how to allocate resources in IBM Cloud for your node, see this topic in the IBM Knowledge Center: https://cloud.ibm.com/docs/services/blockchain?topic=blockchain-ibp-console-govern#ibp-console-govern-allocate-resources
7. In the **Associate identity** step, you can choose an admin for your ordering service. Select an admin and click **Next**.
8. Review the **Summary page** and click **Add ordering service**.

Figure 4-36 confirms that the ordering service was successfully deployed.

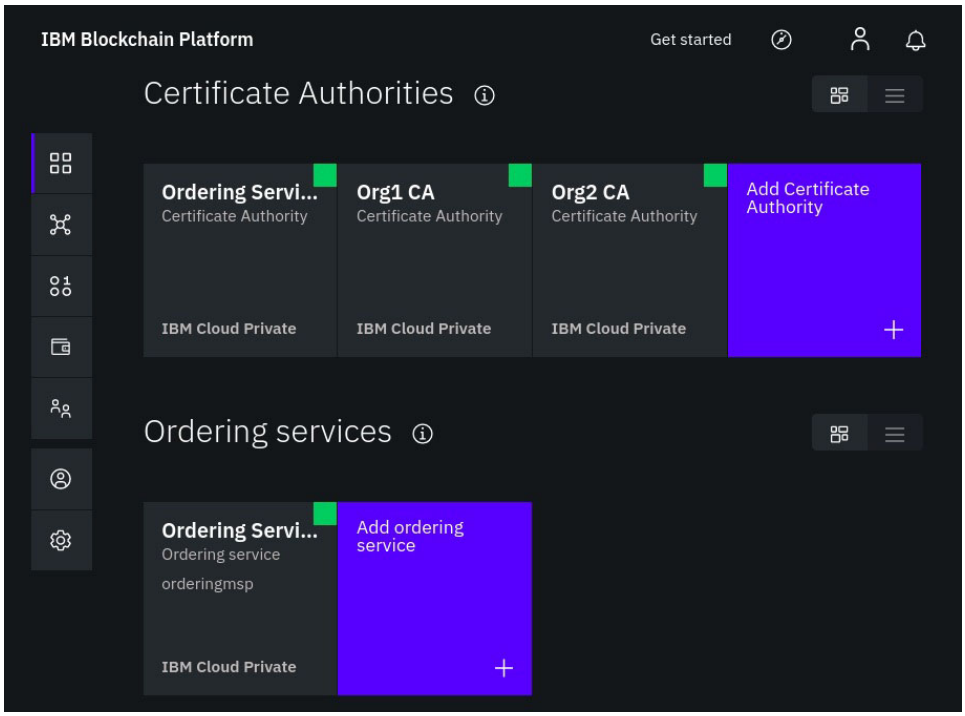


Figure 4-36 Ordering service deployed

4.3.4 Join the consortium

A peer organization must be known to the ordering service before it can create or join a channel. (This activity is also called joining the *consortium*, the list of organizations that are known to the ordering service.)

At the technical level, channels are messaging paths between peers through the ordering service. A peer can be joined to multiple channels without information passing from one channel to another. Likewise, an ordering service can have multiple channels run through it without exposing data to organizations on other channels.

Because only ordering service admins can add peer organizations to the consortium, you either must be the ordering service admin or send MSP information to the ordering service admin.

In the console, you join organizations to a consortium as follows:

1. Navigate to the **Nodes** tab.
2. Scroll down to the ordering service that you created and click to open it.
3. Under **Consortium Members**, click **Add organization**.
4. From the drop-down list, select the MSP of the organization that you want to add.
5. Click **Add organization**.
6. Repeat the steps for any organization that you want to add to the consortium.

Figure 4-37 depicts the organizations that were added to the consortium.

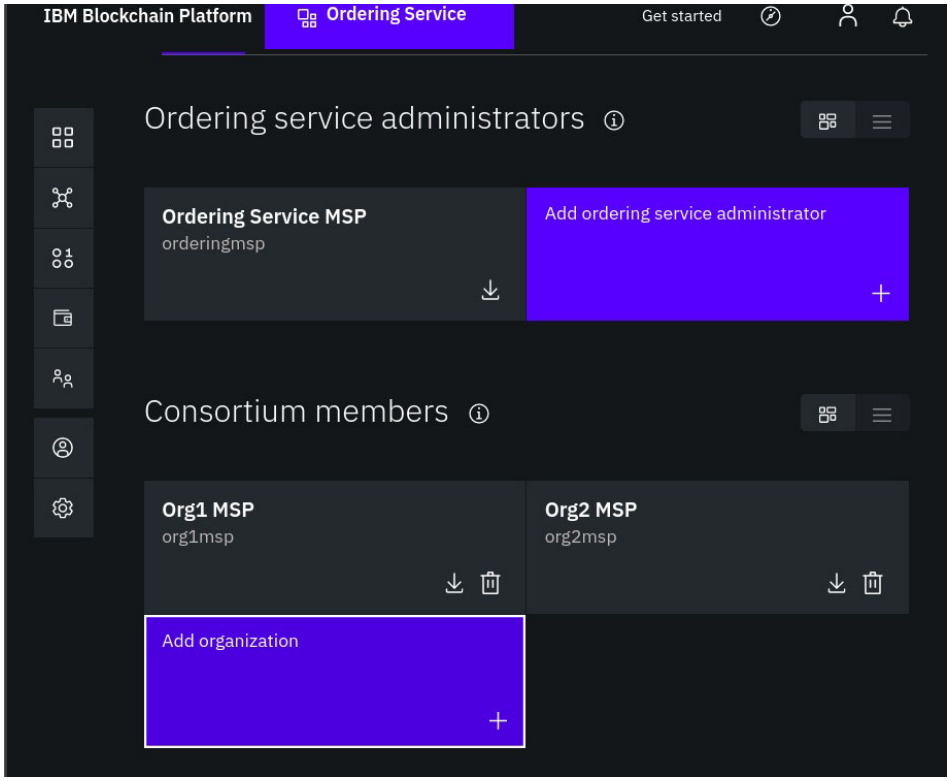


Figure 4-37 Consortium members added

When this process is complete, it is possible for the organizations that participate in the consortium to create or join a channel that is hosted on your ordering service.

In a typical production scenario, the MSP definitions of other organization would be created by different network operators in their own cluster and by using their own IBM blockchain console. In those cases, when the organization wants to join your consortium, the organization MSP definition of the organization must be sent to your console in an out-of-band operation. Also, you must export your ordering service and send it to them. That way, they can import it into their console and join a peer to a channel (or create a new channel).

4.3.5 Creating a channel

The members of a network are usually related business entities that want to transact with each other. However, there might be instances when subsets of the members want to transact without the knowledge of the others. This is possible if you create a channel on which these transactions can take place. Channels replicate the structure of a blockchain network in that they contain members, peers, an ordering service, a ledger, policies, and smart contracts. But channels restrict the membership, and even the knowledge of the channel, to particular subsets of the network membership. In this way, channels ensure that network members can leverage the overall structure of the network while maintaining privacy, where needed.

The console uses peers to gather information about the channels that the peers belong to. Consequently, the console cannot interact with the channel, unless an organization has joined a peer to a channel.

When you have created your CAs, identities, MSPs, ordering service, a peer, and have added your peer organization to the consortium, navigate to the Channels tab in the left navigation. Channel creation and management is handled here.

When you first navigate to this tab, it is empty except for the **Create channel** and **Join channel** buttons. This state reflects the fact that you have not yet created a channel and joined a peer to it.

Consider a case where the organization is not a member of the consortium at channel creation time. You can create the channel and add the organization later by clicking **Settings** on the page of the relevant channel and going through the Update Channel flow.

Follow these steps to create a channel:

1. Navigate to the **Channels** tab.
2. Click **Create channel**. A side panel opens.
3. Give the channel a **name**. Make a note of this value. Later, you must share it with anyone who wants to join this channel.
4. Select the desired ordering service from the drop-down list.
5. Choose the **Organizations** who will be a part of this channel. Make at least one organization an **Operator**.

Note: Do not use the Ordering Service MSP here.

6. Choose a **Channel update policy** for the channel. This is the policy that dictates how many organizations will have to approve updates to the channel configuration. As you add organizations to the channel, you should change this policy to reflect the needs of your use case. A sensible standard is to use a majority of organizations. For example, 3 out of 5.
7. Specify any **Access control** limitations that you want to make.

Note: This is an advanced option. If you set the access to a resource to a particular organization, it will restrict access to that resource for every other organization in the channel. Consider a case where the default access to a particular resource is the Readers of all organizations, and that access is changed to the Admin of Org1. In this case, only the admin of Org1 will have access to that resource. Because access to certain resources is fundamental to the smooth operation of a channel, you must plan access control decisions carefully. If you decide to limit access to a resource, ensure that the access to that resource is added, as needed, for each organization.

8. Select the **Channel creator organization**. Because the console allows multiple organizations to be owned by a single user, you must specify which organization is creating the channel.
9. When you are ready, click **Create channel**. The console returns you to the **Channels** tab and you can see a pending tile of the channel that you just created (Figure 4-38).

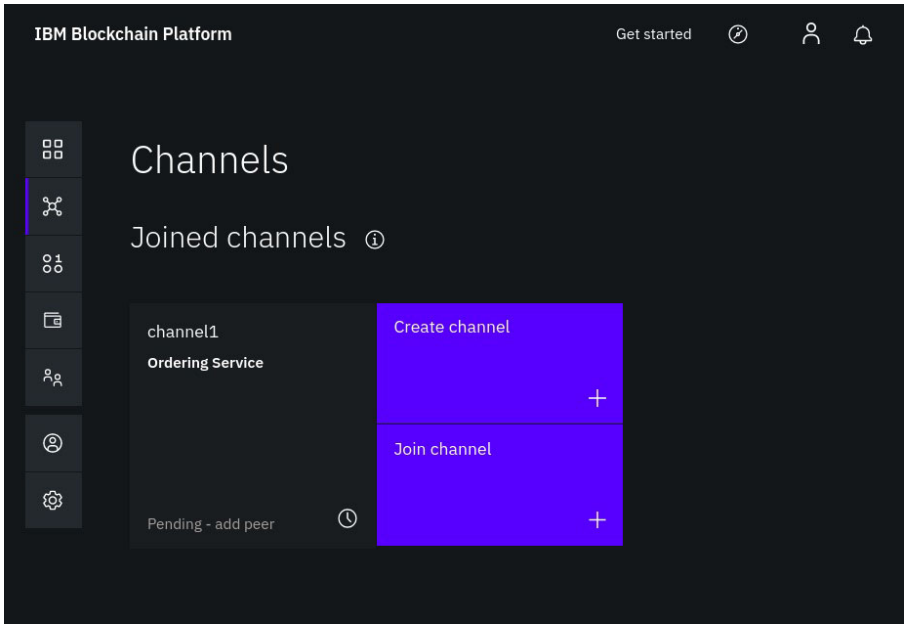


Figure 4-38 Created channel will remain in status pending until one or more peers are added

4.3.6 Joining peers to channel

Perform the following steps from your console:

1. Navigate to the **Channels** tab
2. Click the pending tile channel to launch the side panel.
3. Select which peers you want to join to the channel.
4. Click **Join channel**.

After you join peers to a channel, the genesis block is created and deployed to peers. The tile for the channel shows the block height as one block as seen in Figure 4-39.

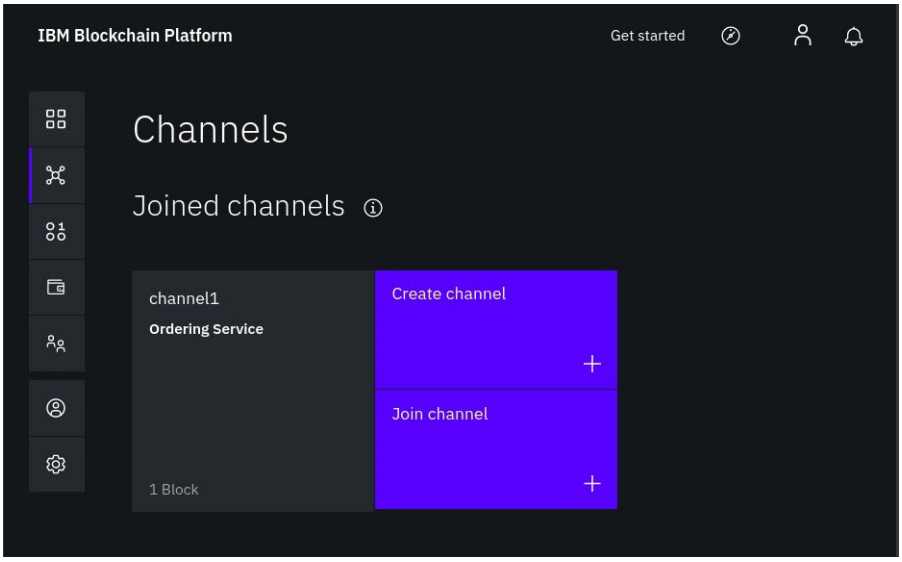


Figure 4-39 Peers joined the channel and it now shows block height

About anchor peers: For service discovery and private data to work, you must enable cross-organizational communication that is based on the gossip protocol. An anchor peer must exist for each organization. This anchor peer is not a special type of peer. It is merely the peer that the organization makes known to other organizations and it works to bootstrap cross-organizational gossip. Therefore, you must define at least one anchor peer for each organization in the collection definition. For more information on the gossip protocol (see https://hyperledger-fabric.readthedocs.io/en/stable/gossip.html).

To configure a peer to be an **anchor peer**, click the **Channels** tab and open the channel where the smart contract was instantiated. Then, follow the steps below:

1. Click the **Channel details** tab.
2. Scroll down to the **Anchor peers** table and click **Add anchor peer**.
3. Select at least one peer from each organization in collection definition that you want to serve as the anchor peer for the organization. For redundancy reasons, you can select more than one anchor peer from each organization in the collection.

4.3.7 Deploying smart contracts

A smart contract is the code, sometimes referred to as chaincode, that allows you to read and update data on the blockchain ledger. A smart contract can turn business logic into an executable program that is agreed to and verified by all members of a blockchain network.

Smart contracts are installed on peers and then instantiated on channels. *All members that want to submit transactions or read data by using a smart contract need to install the contract on their peer*. A smart contract is defined by its name and version. Therefore, both the name and version of the installed contract must be consistent across all peers on the channel that plan to run the smart contract.

Note: Until a peer installs a smart contract, it cannot access or update data on the ledger.

After a smart contract is installed on the peers, a single network member instantiates the contract on the channel. The network member needs to have joined the channel in order to perform this action. Instantiation updates the ledger with the initial data that is used by the smart contract. Then, it starts smart contract containers on peers that are joined to the channel that have the contract installed. The peers can then use the running containers to transact.

- Only one network member needs to instantiate a smart contract.
- If a peer with a smart contract installed joins a channel where the same smart contract version has already been instantiated, the smart contract container starts automatically.

Figure 4-40 depicts the workflow of smart contracts, including every step that is required to manage them.

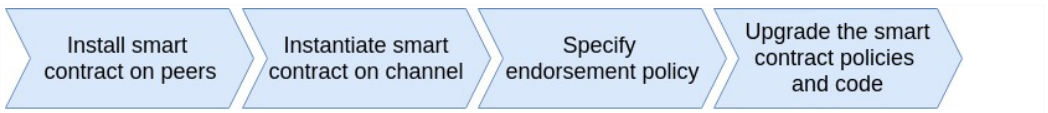


Figure 4-40 Workflow for smart contracts.

Install smart contract on peers

The IBM blockchain console manages the *deployment* of smart contracts rather than the development. For smart contract development, you can use the IBM Blockchain Platform Visual Studio (VS) Code extension.

Use your console to perform these steps:

1. Click the **Smart contracts** tab to install one or more smart contracts.
2. Click **Install smart contract** to upload the smart contract package file in `.cds` format.

   Note: The smart contract package file must be less than 4 MB in size.

   You can use the IBM Blockchain Visual Studio code extension to create a smart contract package in `.cds` format. When you install the package from the **Smart contracts** tab, you can select one or more peer nodes to install the smart contracts on.
3. If only one peer exists in the console, the smart contract is installed on it. Otherwise, you are prompted to select a peer on which to install the smart contract. After you select the peers that you want, click **Install smart contract**.

   Installed smart contracts are listed under “Installed smart contracts” sections of Smart Contracts tab, as seen in Figure 4-41.

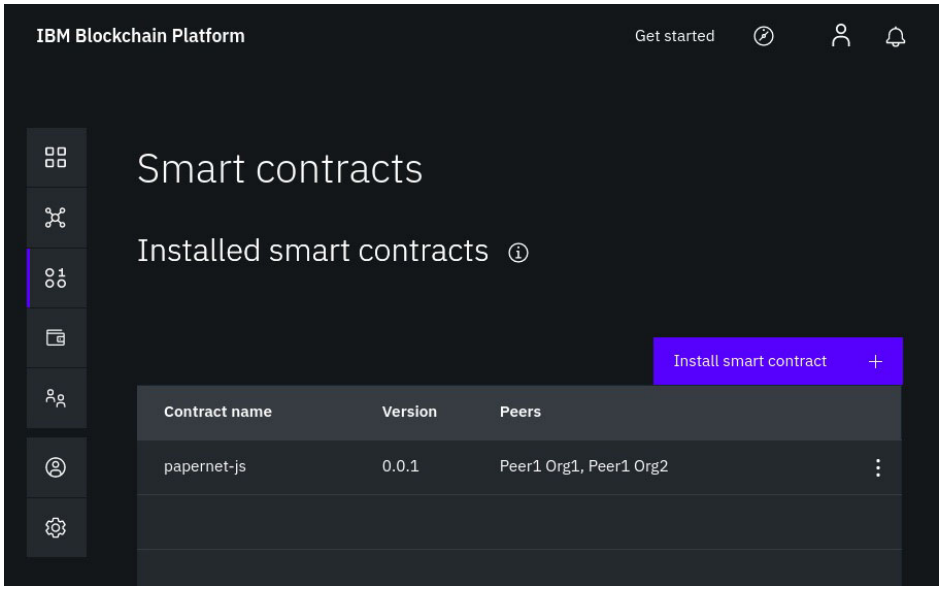


Figure 4-41 Contract name, version and peers where the smart contract is installed

4. You can navigate to the **Nodes** tab and click a peer that is managed by your console. There, you see the list of smart contracts that are installed on an individual peer.

Instantiate a smart contract and specify endorsement policy

Smart contracts are instantiated on a channel. Any console member with peers that are joined to a channel can instantiate a smart contract and specify the associated endorsement policy.

The combination of installation and instantiation is a powerful feature. That way, a peer to use a single smart contract across many channels. Peers might want to join multiple channels that use the same smart contract, but with different sets of network members able to access the data. A peer can install the smart contract once, and then use the same smart contract

container on any channel where it has been instantiated. This lightweight approach saves compute and storage space, and helps you scale your network.

Perform these steps in the console to instantiate a smart contract:

1. On the **smart contracts** tab, find the smart contract from the list that is installed on your peers. Then, click **Instantiate** from the overflow menu on the right side of the row.
2. On the side panel that is displayed, select a **channel** on which to instantiate the smart contract. Then, click **Next**.
3. You can use **Simple** or **Advanced** specification for the endorsement policy for the smart contract:
   - **Simple:** From the list of peers that have installed the smart contract on the channel, you select the peers that need to endorse the transaction. You can use this method to specify an endorsement policy of all channel members, a majority of them, a single member, or a simple +1 preventing members from self-signing. The default endorsement policy is set to 1 of x, meaning only a single member is required to endorse a smart contract transaction.
   - **Advanced:** Use this option when you want to specify a policy in JSON format. You can use this method to specify more complicated endorsement policies. For example, a policy can require that a certain member of the channel must validate a transaction, along with a majority of other members.

   Note: Endorsement policies are not updated automatically when new organizations join the channel and install a chaincode. Consider a scenario where the endorsement policy requires two of five organizations to endorse a transaction. When a new organization joins the channel, that policy is not updated to require two out of *six* organizations. Instead, the new organization will not be listed on the policy, and they will not be able to endorse transactions. To add another organization to such an endorsement policy, you must upgrade the relevant chaincode and update the policy.

4. On the **Select peer** panel, select a peer from the drop-down list that is from an organization that is a member of the channel. Click **Next.**
5. If your smart contract includes Fabric private data collections, you need to upload the associated collection configuration JSON file. Otherwise, you can skip this step.
   For more information on using private data, see this topic:
   https://cloud.ibm.com/docs/services/blockchain/howto?topic=blockchain-ibp-console-smart-contracts#ibp-console-smart-contracts-private-data
6. On the last panel, you are prompted to specify the name of the smart contract initialization function, and also the associated arguments to pass to that function.

Tip: View all of the smart contracts that have been instantiated on a channel as follows: 1) Click the channel icon in the left navigation. 2) Select a channel from the table. 3) Click the **Channel details** tab.

Upgrading a smart contract

You can upgrade a smart contract to change its code, endorsement policy, or private data collection while it maintains its relationship to the assets on the ledger. There are a various reasons why you might want to upgrade an instantiated smart contract.

- You can upgrade the smart contract to add or remove functionality from its code and iterate on the logic of your business network.

- Whenever a new member is added to a channel, the endorsement policy of the instantiated smart contracts must be updated to include the new channel member. To work with the new channel member, the smart contract must be repackaged with a new version number and instantiated on the channel, even if the smart contract itself is unchanged. Otherwise, transaction endorsement cannot succeed.
- When a private data collection has changed, for example an organization is added or removed you need to upgrade your smart contract. Or, use this action whenever a new private data collection is added to the collection configuration JSON file.
- The smart contract initialization arguments have changed.

To upgrade a smart contract, install the updated code by specifying the *same smart contract name but by using and a new version number*. If you have installed a newer version of a smart contract on any peer in the channel, notice that the original version now has the **Upgrade Available** button next to it in the **Instantiated smart contracts** table (in **Smart contracts** tab).

Follow these steps to upgrade your smart contract:

1. Navigate to the **Smart contracts** tab on the left.
2. Scroll down to the **Instantiated smart contracts** table.
3. In the **Instantiated smart contracts** table, locate the smart contract version and channel that you want to upgrade. It must have the **Upgrade Available** label next to it.
4. Click the overflow menu on the right side of the smart contract row and click **Upgrade**.
5. Select the **smart contract version** that you want to upgrade on the channel from the drop-down list.
6. Update the endorsement policy by adding or removing channel members. You can also click **Advanced** to paste in a new JSON formatted string, which modifies the existing policy.
7. On the **Select peer** panel, you need to select a peer that can approve the proposal to upgrade the smart contract. Therefore, you must select a peer from the drop-down list that is from an organization that was a member of the channel before the smart contract was last instantiated on the channel.
8. If you want to associate a private data collection configuration file with the smart contract, you can upload the JSON file. Or if you want to update an existing collection configuration, you can upload the JSON file.
9. If the smart contract was previously instantiated with a collection configuration file, you must again upload the previous version or a new version of the collection configuration file during this step.
10. *(Optional)* Modify the smart contract initialization argument values if the parameters have changed. If you are unsure about it, check with your smart contract developer. If they have not changed, you can leave this field blank.

    After you upgrade the smart contract, you will change the version of the contract that is instantiated on the channel. And you will change the smart contract container for all the peers that have installed the new version. If you are using private data collections, be sure that you have configured anchor peers on the channel.

4.3.8 Verifying blockchain components installation

To verify installation of the blockchain components, you should verify the deployments, replicaSets, Pods, and Services that were created for blockchain. Use the commands in Example 4-23, Example 4-24, and Example 4-25 to verify each of these Kubernetes objects.

Example 4-22 Blockchain component deployments showing ordering service, CA and Peer deployments

```
$ kubectl get deploy --namespace=redbook-team00
NAME                          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
orderingservice1              0         0         0            0           5d
orderingserviceca             1         1         1            1           5d
org1ca                        1         1         1            1           6d
org2ca                        1         1         1            1           6d
peer1org1                     0         0         0            0           5d
peer1org2                     0         0         0            0           5d
redbook-team00-ibp-console    1         1         1            1           11d
```

Example 4-23 Blockchain component replicaset showing ordering service, CA and Peer replicaset

```
$ kubectl get rs --namespace=redbook-team00
NAME                                    DESIRED   CURRENT   READY     AGE
orderingservice1-75c5955d79             0         0         0         5d
orderingservice1-9fc97c8b4              0         0         0         5d
orderingserviceca-c4d577476             1         1         1         5d
org1ca-66d5848fc8                       1         1         1         6d
org2ca-5d6bd5d8b4                       1         1         1         6d
peer1org1-5bbf776ff5                    0         0         0         5d
peer1org1-74c654f4c7                    0         0         0         5d
peer1org1-78d4458c57                    0         0         0         5d
peer1org2-6b949769cc                    0         0         0         5d
peer1org2-7f9c44dc75                    0         0         0         5d
redbook-team00-ibp-console-78b77b676d   1         1         1         11d
```

Example 4-24 Blockchain component pods showing ordering service, CA and Peer pods

```
$ kubectl get pods --namespace=redbook-team00
NAME                                      READY     STATUS    RESTARTS   AGE
orderingserviceca-c4d577476-z88px         1/1       Running   0          4d
org1ca-66d5848fc8-5nvb9                   1/1       Running   0          4d
org2ca-5d6bd5d8b4-8wx8k                   1/1       Running   0          4d
redbook-team00-ibp-console-78b77b676d-db2dc   5/5       Running   2          11d
```

Example 4-25 Blockchain component services

```
$ kubectl get svc --namespace=redbook-team00
NAME                                TYPE        CLUSTER-IP     EXTERNAL-IP    AGE     PORT(S)
ibp-operator                        ClusterIP   10.0.233.8     <none>         11d     8383/TCP
orderingservice1-service            NodePort    10.0.62.129    <none>          5d     7050:30858/TCP,
  8443:31497/TCP,8080:30644/TCP,7443:31575/TCP
orderingserviceca-service           NodePort    10.0.81.10     <none>          5d     7054:32127/TCP,
  9443:31979/TCP
org1ca-service                      NodePort    10.0.76.40     <none>          6d     7054:32152/TCP,
  9443:30170/TCP
org2ca-service                      NodePort    10.0.245.113   <none>          6d     7054:31141/TCP,
  9443:31797/TCP
peer1org1-service                   NodePort    10.0.146.99    <none>          5d     7051:31599/TCP,
  7052:32455/TCP,9443:32558/TCP,8080:30374/TCP,7443:31755/TCP
peer1org2-service                   NodePort    10.0.196.158   <none>          5d     7051:30635/TCP,
  7052:30707/TCP,9443:30790/TCP,8080:30002/TCP,7443:32503/TCP
redbook-team00-ibp-console-optools  NodePort    10.0.92.28     <none>         11d     3000:30000/TCP,
  3001:30001/TCP
```

4.4 OpenShift support: Statement of direction

In July 2019, IBM acquired Red Hat, an enterprise software company with an open source development model. Red Hat is now a part of IBM's Hybrid Cloud division.

OpenShift (https://www.openshift.com/) is a RedHat Platform-as-a-Service (PaaS) solution based on Kubernetes. At the time this book is published, OpenShift is not supported in IBM LinuxONE. Availability of support is expected by 1Q 2020. After OpenShift is available in LinuxONE, we are planning to update this book to include directions on how to enable IBP on OpenShift.

4.5 Troubleshooting the installation

This section explains some typical errors that you might see when installing the IBM Blockchain Platform (IBP) and the IBP console.

4.5.1 Troubleshooting console installation

The command in Example 4-26 shows how to find errors that are related to IBP console.

Example 4-26 Command to grep for errors for console

```
$ kubectl logs redbook-team00-ibp-console-5c7c48669d-zhfzd --all-containers | grep -i ERROR
```

Common errors

Here are typical errors that arise during installation of the console.

You might get an **out of cpu** or **out of storage** error. Example 4-27 shows an **out of cpu** error. This means that you need more resources on your machine to schedule the console, according to the resource requests that are specified for the console installation.

Example 4-27 .Out of cpu error.

```
$kubectl  describe pods peer1org1-5bbf776ff5-tw6r  -n redbook-team00
Events:
  Type     Reason            Age                 From               Message
```

```
  ----     ------            ----                 ----               -------
  Warning  FailedScheduling  42s (x85 over 56m)  default-scheduler  0/1 nodes are
available: 1 Insufficient cpu
```

You also might get an role-based access control (RBAC) error based on lack of permissions to a resource within Kubernetes. If you get this, make sure that you satisfy all the prerequisites for installing the console. Focus on checking that you have created all necessary clusterroles, roles, clusterrolebindings, rolebindings, and pod security policies.

Example 4-28 Incorrect permissions (permissions errors for rbac)

```
kubectl get pods redbook-team00-ibp-console-78b77b676d-68cnn
NAME                                          READY   STATUS             RESTARTS   AGE
redbook-team00-ibp-console-78b77b676d-68cnn   4/5     CrashLoopBackOff   5          3m58s

kubectl logs redbook-team00-ibp-console-78b77b676d-68cnn -c operator
{"level":"error","ts":1567087886.392721,"logger":"k8sutil","msg":"Failed to get
Pod","Pod.Namespace":"redbook-team00","Pod.Name":"redbook-team00-ibp-console-78b77b676d-68cnn","error":"pod
s \"redbook-team00-ibp-console-78b77b676d-68cnn\" is forbidden: User
\"system:serviceaccount:redbook-team00:ibp\" cannot get resource \"pods\" in API group \"\" in the
namespace \"redbook-team00\""
```

4.5.2 Troubleshooting blockchain component installation

The commands in Example 4-29 show how to grep errors for blockchain components.

Example 4-29 Command to grep for errors in blockchain components

```
$ kubectl logs orderingserviceca-c4d577476-z88px  --all-containers -n redbook-team00 | grep -i ERROR
$ kubectl logs org1ca-66d5848fc8-5nvb9  --all-containers -n redbook-team00 | grep -i ERROR
$ kubectl logs org2ca-5d6bd5d8b4-8wx8k  --all-containers -n redbook-team00 | grep -i ERROR
```

Common errors

Here are some common errors that you might see for IBP installation:

- *Need more permissive PodSecurity policy*. Example 4-30 shows this error, which deals with Certificate Authority component permissions.

Example 4-30 PodSecurity policy error

```
$kubectl get events
LAST SEEN   TYPE      REASON                  KIND                    MESSAGE
9s          Warning   FailedCreate            ReplicaSet              Error creating: pods
"peer1ca-846fcbf996-" is forbidden: unable to validate against any pod security policy:
[spec.initContainers[0].securityContext.runAsNonRoot: Invalid value: false: must be true
spec.initContainers[0].securityContext.capabilities.add: Invalid value: "CHOWN": capability may not be
added spec.initContainers[0].securityContext.capabilities.add: Invalid value: "FOWNER": capability may not
be added spec.containers[0].securityContext.capabilities.add: Invalid value: "NET_BIND_SERVICE": capability
may not be added]
54s         Normal    ExternalProvisioning    PersistentVolumeClaim   waiting for a volume to be created,
either by external provisioner "nfs-provisioner" or manually created by system administrator
```

- *Insufficient memory allocation in Peer pod* causes DinD container to fail as shown in Example 4-31.

Example 4-31 DinD container memory allocation error

```
Error processing tar file(exit status 1): open
/usr/local/lib/node_modules/npm/node_modules/path-exists/license: cannot allocate memory
Loaded image: noderuntime:latest
Done loading images
REPOSITORY TAG IMAGE ID CREATED SIZE carruntime latest e1c971eb0642 2 weeks ago 14.4MB golangruntime latest
e1c971eb0642 2 weeks ago 14.4MB noderuntime latest 36f10d8170f0 2 weeks ago 839MB
time="2019-06-27T19:37:11.889461068Z" level=error msg="Not continuing with pull after error:
errors:\ndenied: requested access to the resource is denied\nunauthorized: authentication required\n"
time="2019-06-27T19:37:11.889486118Z" level=info msg="Ignoring extra error returned from registry:
unauthorized: authentication required"
time="2019-06-27T19:37:12.024115280Z" level=warning msg="could not write error response: write tcp
127.0.0.1:2375->127.0.0.1:44280: write: broken pipe
```

Insufficient memory allocation to the Peer pod might cause underlying errors for DinD chaincode containers. Confirm that you meet the prerequisites for the installation.

- *Unable to configure RAFT node to System channel* error is shown in Example 4-32.

Example 4-32 RAFT node unable to connect to system channel

```
Unable to get system channel. If you associated an identity without administrative
privilege on the ordering service node, you will not be able to view or manage
ordering service details
```

Check the pod logs of ordering service. Those logs might reveal failures in deployment of the ordering service pods.

5

Specific scenarios

In this chapter we discuss specific deployment scenarios. This chapter has the following sections.

5.1 Behind firewalls (isolated blockchain environment)

Highly regulated industries often require an air-gapped, disconnected cloud; in other words, a private cloud solution that is not connected to the internet. Even these industries can reap the benefits of DevOps, Microservices, Cloud, and Containers by setting up local catalogs and offline Docker image repositories, private Docker registry, and private Helm chart repository.

IBM Blockchain Platform is delivered as a Helm chart with all the images that are required to install the solution. There is no need for an internet-connected server. You only need an internet-connected laptop or host somewhere that you can use to download the Helm charts from IBM Passport Advantage (PPA).

Note: If you use JavaScript to write your chaincode, you must set up local repositories for `npm modules`.

5.2 Using proxies

This section describes two options for installing Docker in your IBM Cloud Private, behind an HTTP proxy. First, you must manually install Docker on your boot node. Second, you can either manually install Docker on the rest of your cluster nodes, or the installer can automatically install Docker.

5.2.1 Manual installation of Docker

1. Create the **docker.service.d/** folder. You must run the following command on all nodes (boot, master, management, proxy, work, and VA nodes):

Example 5-1 Creating necessary docker service folder

```
sudo mkdir -p /etc/systemd/system/docker.service.d
```

2. Create the **docker.service.d/http-proxy.conf** file and add the following variables: HTTP_PROXY, HTTPS_PROXY and NO_PROXY

*Example 5-2 Sample of /*etc/sytemd/system/docker.service.d/http-proxy.conf file

```
[Service]
Environment="HTTP_PROXY=http://1.2.3.4:3128" "HTTPS_PROXY=http://1.2.3.4:3128"
"NO_PROXY=localhost,127.0.0.1,<cluster_CA_domain>.icp,<ICP ip address/range>
```

Note: The NO_PROXY entry dictates that no proxy should be used for the IBM Cloud Private's Docker private registry. <*cluster_CA_domain*> is the certificate authority (CA) domain that was set in the config.yaml file during installation. Change <*ICPipaddress*/*range*> to the IP address range of your ICP nodes, for example, 192.168.1.0/24. This setting ensures that Docker doesn't use the proxy for inter-Docker communications.

3. Run the following commands to restart Docker:

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

4. Customize the IBM Cloud Private config.yaml file by setting the **tiller_http_proxy** and **tiller_https_proxy** parameters as shown in Example 5-3. This configures proxy settings of the Helm tiller daemon to populate the IBM Cloud Private App Catalog.

Example 5-3 Sample of /<installation_directory>/cluster/config.yaml file

```
/<installation_directory>/cluster/config.yaml

# Licensed Materials - Property of IBM
# @ Copyright IBM Corp. 2017 All Rights Reserved
# US Government Users Restricted Rights - Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
---
## Network Settings
network_type: calico
## Network in IPv4 CIDR format
network_cidr: 10.1.0.0/16
## Kubernetes Settings
service_cluster_ip_range: 10.0.0.1/24
...
tiller_http_proxy: http://1.2.3.4:3128
tiller_https_proxy: http://1.2.3.4:3128
...
```

5. Continue the standard IBM Cloud Private installation process. From the IBM Cloud Private management console, check **Catalog**.

5.2.2 Automatic installation of Docker by using IBM Cloud Private

If you installed Docker on your boot node manually and you haven't installed it on your other cluster nodes, you use the IBM Cloud Private installer to automatically deploy Docker. Follow the steps below to install IBM Cloud Private behind an HTTP proxy.

1. Uncomment the following Docker environment variables in your config.yaml file, which are bolded here for clarity:

Example 5-4 Sample of config.yaml file

```
## Docker environment setup
docker_env:
- HTTP_PROXY=http://1.2.3.4:3128
- HTTPS_PROXY=http://1.2.3.4:3128
- NO_PROXY=localhost,127.0.0.1,{{ cluster_CA_domain }}
## Install/upgrade docker   version
```

2. Customize the IBM Cloud Private config.yaml file, set the **tiller_http_proxy** and **tiller_https_proxy** parameters, as shown in the following command:

Example 5-5 Sample of /<installation_directory>/cluster/config.yaml file

```
# Licensed Materials - Property of IBM
# IBM Cloud private
# @ Copyright IBM Corp. 2017 All Rights Reserved
# US Government Users Restricted Rights - Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
---
## Network Settings
network_type: calico
## Network in IPv4 CIDR format
network_cidr: 10.1.0.0/16
## Kubernetes Settings
```

```
service_cluster_ip_range: 10.0.0.1/24
...
tiller_http_proxy: http://1.2.3.4:3128
tiller_https_proxy: http://1.2.3.4:3128
...
```

3. Continue the standard IBM Cloud Private installation process. From the IBM Cloud Private management console, check **Catalog**.

5.2.3 Post-installation proxy configuration

After installation is completed, you can edit proxy settings of IBM Cloud Private as described here:

1. From the IBM Cloud Private management console, go to **Workloads** → **Deployments.**
2. In the Deployments window search for `helm-api`.
3. Click **Edit**.
4. Look for the following lines as shown in Example 5-6:

Example 5-6 ICP proxy configuration -1

```
{
"name": "HTTP_PROXY"
},
{
"name": "HTTPS_PROXY"
},
{
"name": "NO_PROXY",
"value": "<ICP cluster
IP>,mycluster.icp,mongodb,platform-identity-provider,localhost,127.0.0.1"
},
```

5. Edit the HTTP_PROXY and HTTPS_PROXY values as appropriate.

Example 5-7 ICP proxy configuration -2

```
{
"name": "HTTP_PROXY",
"value": "http://1.2.3.4:3128"
},
{
"name": "HTTPS_PROXY",
"value": "http://1.2.3.4:3128"
},
{
"name": "NO_PROXY",
"value": "<ICP cluster
IP>,mycluster.icp,mongodb,platform-identity-provider,icp-management-ingress,iam-pap,loca
lhost,127.0.0.1"
},
```

Note: Depending on your environment, the NO_PROXY values might vary. For example, they might include Kubernetes Ingresses and Services resources. It is important that NO_PROXY is fully configured to prevent IBM Cloud Private from communicating over the proxy.

6. Click **Submit**.
7. Go to Catalog to confirm that the Helm charts are shown.

5.3 High availability and disaster recovery

This section presents considerations for high availability and disaster recovery.

5.3.1 High availability

High availability is a core discipline in an IT infrastructure to keep solutions up and running, even after a partial or full site failure. The main purpose of high availability is to eliminate potential points of failures in an IT infrastructure. For example, a solution might be prepared for the failure of one system by adding redundancy and setting up failover mechanisms. Solutions can achieve high availability on different levels in the IT infrastructure and within different layers of Kubernetes cluster. The level of availability that is right for solution depends on several factors, such as business requirements and the Service Level Agreements.

As with many other complex IT systems, we can create a reliable blockchain solution based on inherently unreliable components. We achieve this goal by leveraging redundancy and careful design. A highly available blockchain solution should ensure that blockchain components of the same type and organizations are deployed across different worker nodes. By adding redundancy across the blockchain network a solution, you avoid failures and downtime.

Consider the following issues for a highly available blockchain solution:

Table 5-1 Considerations for a highly available blockchain solution

Consideration	Details
Peers	High Availability for peers means always having redundant peers. In other words, have at least two peers available for each organization on the same channel to process requests from client applications. Multiple peers can be deployed to a single worker node, or spread across worker nodes. Whenever you deploy multiple peers and join them to the same channel, the peers act as highly available pairs. This is possible because the channel and the data are automatically synchronized across all peers in the channel. By design, a blockchain network can have multiple organizations that transact on the same channels. Therefore, the common deployment model for any given channel is as follows: ► Create redundant peers for each organization ► Spread the peers across several organization account clusters. ► Ensure that the clusters are all synchronizing data between each other. ► Each organization can have a peer in their own cluster in any region.
Ordering service	IBM Blockchain Platform is built upon Hyperledger Fabric v1.4.1, which includes the Raft ordering service. Raft is a crash fault tolerant (CFT) ordering service that is based on an implementation of the Raft protocol. By design, Raft ordering nodes automatically synchronize data between each other by using Raft based consensus. In IBM Blockchain Platform, an organization network operator can choose one of these options: ► Stand up a single-node Raft-based orderer, with no HA. OR ► Stand up five orderers in a single region that are automatically configured for HA through Raft.
Hyperledger Fabric CA	The IBP approach is to deploy the Hyperledger Fabric CA server in Kubernetes cluster mode for HA. This deployment option is well documented in the Hyperledger Fabric documentation.

Consideration	Details
Transaction endorsement policy	If a policy requires a particular peer to endorse a transaction and that peer is down, every transaction fails endorsement. When possible, leverage endorsement policy rules that are not peer specific. For example, “2 out-of 3 must endorse” is a policy rule that is not peer specific. For more information, see Section 4.3.7, “Deploying smart contracts” on page 189. **Note:** Apart from the blockchain components, consider designing the integration layer and application layer to be highly available to achieve a highly available solution.

5.3.2 Disaster recovery

The goal of disaster recovery is to restore the business after an unplanned outage. It does this by providing a standby of a primary storage and keeping it current through replication of changes from the primary. Changes can be replicated synchronously or asynchronously. If done asynchronously and an outage happens, changes might be lost (if the primary is down permanently) or stranded (if the primary is down temporarily). The amount of change data tht is lost or stranded depends on the latency of replicated changes. If done synchronously, no committed changes are lost or stranded in an outage.

In all cases, to protect against data corruption, it is recommended that a solution regularly back up the storage that is associated with every deployed component. Because the blockchain ledger is shared across all the peers and ordering nodes, taking regular backups is critical. For example, if incorrect data is accidentally propagated to a peer's ledger or if data is mistakenly deleted, this might spread to the ledgers of some other peers. This would require the restoration of the ledgers of all the peers from an established backup point to ensure synchronicity. You can decide how often to perform the backups, based your recovery needs. A general guideline is to perform daily backups.

For more information on disaster recovery on IBM Cloud Private, see this website:

https://www.ibm.com/cloud/garage/practices/manage/high-availability-ibm-cloud-private/1_0

6

Performance considerations

This chapter discusses the performance considerations of this solution and has the following sections:

6.1 Blockchain performance considerations

When you run IBM Blockchain Platform for Multicloud, you can make many decisions about the configuration of your blockchain application and network. This section outlines the effect of different decisions on the overall performance of your solution.

6.1.1 Application client

Application client sends transactions to the endorsing nodes in the blockchain network. The programming language in which you develop the client application influences the choices of Hyperledger SDK to use. Hyperledger SDKs are available in Go, NodeJS, Java, and Python, as shown in GitHub:

https://github.com/hyperledger?utf8=%E2%9C%93&q=fabric-sdk&type=&language=

The choice of SDK affects performance depending on the language in which it is written. The Go and Java SDKs might yield better performance than the Node.js or Python SDK for one simple reason: *Go and Java are compiled and support multiple threads of execution*.

6.1.2 Smart contract programming language

You can develop smart contracts or chaincode with Go, JavaScript, or Java. As with SDK choices, Go or Java for your chaincode might yield performance than JavaScript, because, to repeat, Go and Java are compiled and support multiple threads of execution. Specifically, Go is a multi-threaded compiled language that takes advantage of single instruction multiple data (simd) computers.

6.1.3 Endorsement policy

Choice of endorsement policy and the number of available endorsing nodes for a channel can affect peer performance. Throughput of the blockchain network is indirectly proportional to number of endorsement peers in the policy. You can scale your throughput by load balancing your endorsement across a pool of endorsers.

6.1.4 Orderer Block Configuration

In ordering service nodes (OSN), the configuration parameters that determine the block size and frequency can affect performance. Modifying these parameters can influence the latency that transactions experience, starting from submission to the OSN and ending with being committed to the ledger. If too few messages are received for a channel to fill the block (per the *BatchSize* configuration), the minimum latency is the *BatchTimeout* setting.

- **Batch Size:** These parameters dictate the number and size of transactions in a block. No block appears that is larger than the value for **absolute_max_bytes** or greater than the value for **max_message_count** transactions inside the block. If it is possible to construct a block with a size that is under **preferred_max_bytes**, a block is cut prematurely, and transactions larger than this size will appear in their own block. Example 6-1 shows the example of configuring batch size on that channel.

Example 6-1 Batch size configuration on channel

```
{
  "absolute_max_bytes": 102760448,
  "max_message_count": 10,
  "preferred_max_bytes": 524288
}
```

- **Batch Timeout**: This value is the amount of time to wait after the first transaction arrives for additional transactions before cutting a block. Decreasing this value improves latency, however decreasing it too much migth decrease throughput by not allowing the block to fill to its maximum capacity. Example 6-2 shows the example of configuring batch timeout on that channel.

Example 6-2 Batch timeout configuration on channel

```
{ "timeout": "2s" }
```

Other properties can also be configured on a channel. See the following URL for a summary of these properties:

https://hyperledger-fabric.readthedocs.io/en/release-1.4/config_update.html

6.1.5 Peer container resource allocation

The physical and virtual infrastructure that all these services run on that can affect performance. For instance, giving the peer nodes more virtual CPUs (vCPUs) can help to improve performance in many cases. However, this approach comes at a cost. So you must balance the cost of running a peer with the benefits of the improved performance.

6.2 Blockchain Input/Output (I/O) accelerated: IBM HiperSockets

When it runs on IBM Z and LinuxONE, IBM Blockchain Platform can take control of IBM HiperSockets in the background. This section explains where blockchain uses I/O, and how employing HiperSockets can improve performance, security, and availability. This section covers the following issues:

- 6.2.1, “Where does blockchain use I/O? Everywhere”
- 6.2.2, “What are HiperSockets” on page 206
- 6.2.3, “HiperSockets benefits” on page 207

6.2.1 Where does blockchain use I/O? Everywhere

Blockchain is a transactional system. Like many other transactional systems, it depends on throughput of transactions that is high enough to satisfy the needs of the use case. This throughput is dependent in part on I/O speed. This is true because this peer-to-peer network employs communication among the peers in the network for data dissemination and cross-verification.

In Hyperledger Fabric, the peers perform their communication function through the gossip protocol (see https://hyperledger-fabric.readthedocs.io/en/stable/gossip.html). This protocol enlists each peer to gossip to other peers about these issues:

- Confirming which peers are currently part of the network.
- Updating all peers regarding new blocks.
- Broadcasting transactions to nearby peers to enable better network scaling.

Each of these communications in the IBM Blockchain Platform, requires a TLS network connection for security. Furthermore, networking between components takes effect at these points:

- Each time that the ordering service sends blocks of data to initial sets of peers.
- When the client sends transactions to peers for endorsement during every transaction submission.

In a Kubernetes network, these components can be spread across multiple virtual machines for reliability and availability and also security. To improve the performance of the network, administrators must optimize these many connections to eliminate delays in network traffic. Otherwise, such delays might lead to both slowdown in the number of transactions that are processed per second and certain peers falling behind the overall network in terms of blockchain records.

One way to optimize performance is HiperSockets. It allows quick transfer of information between blockchain components that sit on different Logical Partitions (LPARS). HiperSockets also offers an EAL5+ certified isolation of components for a level of security that meets the standards of the isolation of separate physical machines. Combining performance and efficiency, HiperSockets unleash our blockchain solution to communicate efficiently, while eliminating the need for excess components. The following two sections describe HiperSockets and all of the potential benefits that they provide.

6.2.2 What are HiperSockets

HiperSockets provides high-speed TCP/IP connectivity for Logical partitions (LPARs) that run in the same LinuxONE machine. Also, it eliminates the need for any physical cabling or external networking connection between servers that run in different LPARs.

The communication is through the system memory of the processor, so servers are connected to form a “internal LAN.”

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

Therefore, the LinuxONE can provide high network connection speeds for blockchain workloads, which require read/write of many components that communicate to each other over a network. Because of blockchain’s requirement for many TCP/IP connections, HiperSockets can play an important role in these communications. It helps to create a solid infrastructure layer to run your blockchain workloads.

As shown in Figure 6-1, HiperSockets enables quick transfer of information between LPARs.

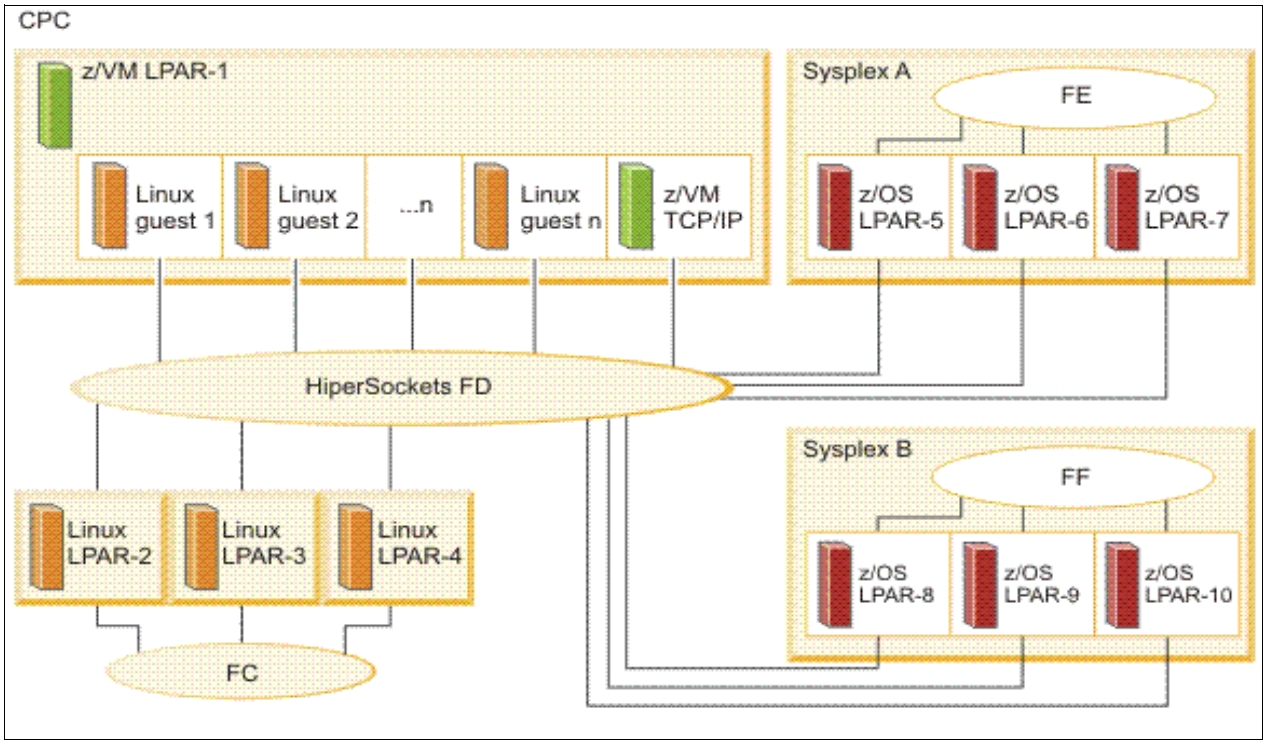


Figure 6-1 HiperSockets enables quick transfer of information between LPARs

6.2.3 HiperSockets benefits

The following table describes the benefits to of IBM HiperSockets:

Table 6-1 Benefits of IBM HiperSockets

Goal	Benefits of IBM HiperSockets
High performance	Consolidated servers that have to access corporate data on the LinuxONE can do so at memory speeds with the lowest possible latency, bypassing network traffic and delays. Also, you can customize HiperSockets to accommodate varying traffic sizes. With HiperSockets, you can define a maximum frame size (MFS) according to the traffic characteristics for each HiperSockets. In contrast, Ethernet LANs have a MFS predefined by their architecture. The bandwidth of your Ethernet interfaces becomes fully available for external network traffic when the internal traffic is running over HiperSockets.
Cost savings	You can use HiperSockets to communicate among consolidated servers in a single processor. Therefore, you can eliminate all of the hardware boxes that run these separate servers. As a result, you reduce floor space and power consumption. With HiperSockets, there are zero external components or cables to pay for, to replace, to maintain, or to wear out.
Consolidation	LinuxONE can consolidate multiple Linux servers into a single machine and can run workloads in SSC (Secure Service Container) partitions. With HiperSockets, this capability reduces the amount of server hardware. It can also reduce the complexity of your physical network, because subnets between the consolidated servers can be turned into virtual HiperSockets networks. The more you consolidate your servers, the greater your savings potential for costs that are related to external servers and their associated networking components.
Flexibility	You make changes to a HiperSockets network more easily, because installation of physical components or re-cabling is not needed.

Goal	Benefits of IBM HiperSockets
Simplicity	HiperSockets is part of LinuxONE technology, including QDIO and advanced adapter interrupt handling. Data transfer is handled much like a cross-address space memory move, using the memory bus. HiperSockets is application-neutral, and it appears as a typical TCP/IP device. Its configuration is simple, so installation is easy. It is supported by familiar management and diagnostic tools.
Security	Cables or external components are vulnerable to physical attacks. Because HiperSockets does not have any physical interfaces, it is protected against any attacks from the outside. This characteristic might enable you to run network traffic without encryption, which results in additional performance improvements. For isolation purposes, you can connect servers to different HiperSockets. All security features, such as firewall filtering, are available for HiperSockets interfaces in the same way that they are for other Internet Protocol network interfaces. In a blockchain environment, HiperSockets increase security because you can define internal network communications between your Linux servers or SSC partitions. As a result, the data traffic stays inside the LinuxONE box, which increases security.
Availability	With HiperSockets, there are no network switches, routers, adapters, or wires that can break or that have to be maintained. The absence of mechanical components greatly improves availability.

6.3 Meet CPACF - Speeding up your blockchain

Blockchain (in the case of IBM Blockchain Platform, specifically Hyperledger Fabric) requires many cryptographic operations, which can become a bottleneck for blockchain performance. Thus, optimal performance depends on the machine's throughput of cryptographic operations. CP Assist for Cryptographic Functions (CPACF) is a set of instructions for the IBM LinuxONE and IBM Z family of systems. This instruction set uses a dedicated portion of the processor chip (for example, a synchronous co-processor) to run cryptographic functions in hardware. As a result CPACF significantly speeds up cryptography operations. This section covers the following topics:

- 6.3.1, "Cryptography's importance in blockchain"
- 6.3.2, "CPACF's role in acceleration and protection" on page 210

6.3.1 Cryptography's importance in blockchain

In this section, we discuss performance considerations for cryptography and blockchain.

Hashing: the key to blockchain's immutability

The blockchain is a data structure with these components:

- A group of blocks, which is a group of transactions that are hashed together.
- Connections among the blocks that form a chain. The chain is made of hashes that point from one block to its previous block.

A *hash* is a one-way function that maps a set of data into a specified number of bits. For example, Secure Hash Algorithm 2 using 256 bits (SHA 2- 256) uses the SHA 2 algorithm to map data into a 256-bit pattern. One-way means that you can use the function to hash the data, but it is not technically feasible to "unhash" the data (find out what the original data set was). Moreover, if the data changes by even one bit, the hash drastically changes. Thus, you can verify that data has not changed by confirming that is has the same hash.

Blockchain systems use hashing to make the blockchain immutable. In other words, based on hashing we can prove that the blockchain has not been changed since it was created. How is this possible? These qualities emerge from hashing:

- If any transactions change, the backing hashes change. Because the block hash is made from a combination of the hashes of the pieces of data in the blockchain, this means that the block hash itself changes.
- Each block in the chain has a pointer to the hash of the previous block. This means that if any of the transactions in that block change, the hash of the block does not match the old hash that is contained in the previous block.
- Unmatched hashes “break” the blockchain, because they signify tampering. The original, untampered set is still be available in the other copies of the ledger (on the other peers). These copies represent the “true” data.
- Verification quickly identifies an invalid blockchain among valid ones:
  - **Valid:** Any blockchain with an unchanged hash — any hash that is unchanged between the time that it was made and the current time — is valid, because it meets the immutability requirement.
  - **Invalid:** A blockchain with a changed hash has been tampered with.

Permissioned blockchain: signing transactions with identities

Hyperledger Fabric, the blockchain protocol that powers the IBM Blockchain Platform in the background, is a permissioned blockchain. This means that the different organizations that participate in the network are known to the other organizations. The network manages identities on this basis:

- A Public Key Infrastructure (PKI). (See this website: https://en.wikipedia.org/wiki/Public_key_infrastructure)
- Certificate Authorities for each organization that issue keys and certificates for identification purposes. (Each Certificate Authority’s root certificate included as part of the configuration of individual networks at the channel level.)

Each time that a user submits a transaction, it includes as part of the submission its *signature*. (A signature is encryption of the hash of the transaction with the private key of the organization.) Each peer verifies this signature with the public key of the submitter. These public and private keys use Elliptic Curve Cryptography (ECC), specifically the Elliptic Curve Digital Signature Algorithm (ECDSA).

Protecting your connections: TLS/SSL in blockchain

The different components in the blockchain (Orderers, Peers, and Certificate Authorities) and also blockchain clients use TLS/SSL connections to communicate. For more information see this website:
(https://access.redhat.com/documentation/en-US/Red_Hat_Certificate_System/8.0/html/Deployment_Guide/SSL-TLS_ecc-and-rsa.html)

TLS/SSL connections involve an initial handshake that is based on elliptic curve cryptography. The handshake exchanges key information to initiate network connections with traffic encrypted that is based on a symmetric encryption algorithm such as the Advanced Encryption Standard (AES). This encryption enables security for connections between clients and blockchain components. It also enables security for connections between blockchain components, which continuously communicate all of the blocks in the chain to distribute them through the gossip protocol. This protocol is further explained on this web page:
https://hyperledger-fabric.readthedocs.io/en/release-1.4/gossip.html

6.3.2 CPACF's role in acceleration and protection

This section raises performance considerations for acceleration and protection.

Acceleration: hardware encryption

The CPACF provides on-chip acceleration for these operations by using instructions for SHA2 and SHA3 (hashing), AES, and new with IBM z15™ ECC (Elliptic Curve Cryptography) including ECDSA. By performing these operations on a dedicated co-processor, on-chip, these cryptographic algorithms run at the full clock speed of the processing chip and minimize the latency that is caused by data transfer. This capability increases the performance of every hashing and signing operation that the IBM Blockchain Platform employs. No intervention from the user is required, besides enablement of the (no-charge) CPACF feature for AES.

Also, CPACF increases the performance of every connection that you make with TLS by using the Advanced Encryption Standard (AES) instructions. CPACF also boosts performance of the ECDA functions on the chip, in the case of IBM z15™. For examples of how performance can increase with CPACF, look at the performance paper for CPACF on z14, which are available in the following document:
[https://www.ibm.com/downloads/cas/K4AR1NLQ

This performance paper includes detailed data for the z14 cryptographic operations that use CPACF. The dat shows that speeds increase as amounts of data that is encrypted increase. This is true because the loading of data is more efficient. The largest bottleneck becomes the cryptographic operations themselves, which are being accelerated by the CPACF coprocessor.

Note: The performance paper applies to IBM z14. Thus, it doesn't describe ECDSA operations for CPACF. In z15, these operations are available as part of CPACF in addition to the symmetric encryption algorithms such as AES and hashing algorithms such as SHA2.

Security: protecting keys from generation (true random) to use

In cryptography, your keys provide access to the data. Whoever has those keys has access. This means that protecting keys is a top priority. LinuxONE and Linux on Z use true random number generation (TRNG) to create keys. This approach improves security and throughput of key generation. This capability is built into the Linux kernel and is transparently implemented in crypto libraries to enhance the security of your system.

Beyond just generating a random key, it also implements measures to keep the key protected. One of these measures is protected key. Protected key employs a wrapping key — two per-LPAR keys (one for AES and one for DES/TDES) — that CPACF code creates and stores in the protected area of the Hardware System Area (HSA). The wrapping key "wraps" (encrypts) encryption keys to protect them. In this way, protected key keeps the encryption key wrapped by its wrapping key in CPACF. When the key is outside of the addressable program memory, it is in the clear only to Licensed Internal Code (LIC). This approach brings these benefits:

- Protects the key from running in the clear where it might be found by an attacker.
- Increases security for keys that are used for symmetric encryption for something like a TLS connection.

For more information on what you can do with a protected key in Linux, see the following explanation of the protected key driver for the Linux on Z operating system:
https://www.ibm.com/support/knowledgecenter/en/linuxonibm/com.ibm.linux.z.ljdd/ljdd_r_pkey.html

A

Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the GitHub material

The web material that is associated with this book is available in softcopy on the internet from the IBM Redbooks GitHub location:
https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain-Platform-for-Multicloud.

Cloning the GitHub material

Complete the following steps to clone the GitHub repository for this book:

1. Download and install `Git` client if not installed from this web page.
2. Run the following command to clone the GitHub repository:

```
git clone
https://github.com/IBMRedbooks/SG248458-Implementation-Guide-for-IBM-Blockchain
-Platform-for-Multicloud.git.
```

Related publications

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

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Be aware that some publications that are referenced in this list might be available in softcopy only.

- *Developing a Blockchain Business Network with Hyperledger Composer using the IBM Blockchain Platform Starter Plan,* REDP-5492

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

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- Zero to Blockchain IBM Redbooks workshop

  https://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/crse0401.html?Open
- Blockchain use cases:

  http://www.ibm.com/blockchain/use-cases
- IBM Blockchain Platform for Multicloud V3.2 Knowledge Center:

  https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.2.0/featured_applications/ibm_blockchain_platform.html
- IBM LinuxONE website:

  https://www.ibm.com/it-infrastructure/linuxone

Help from IBM

IBM Support and downloads

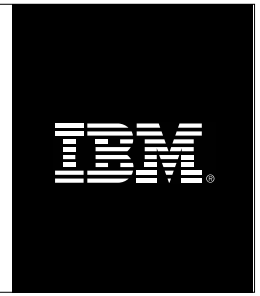
ibm.com/support

IBM Global Services

ibm.com/services

Implementation Guide for IBM Blockchain

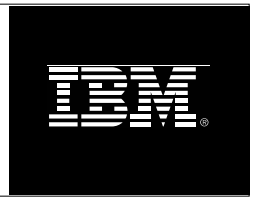
SG24-8458-00
ISBN 0738458031



(1.5” spine)
1.5”<-> 1.998”
789 <->1051 pages

Implementation Guide for IBM Blockchain

SG24-8458-00
ISBN 0738458031



(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

Implementation Guide for IBM Blockchain Platform for Multicloud

SG24-8458-00
ISBN 0738458031

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

Implementation Guide for IBM Blockchain Platform for Multicloud

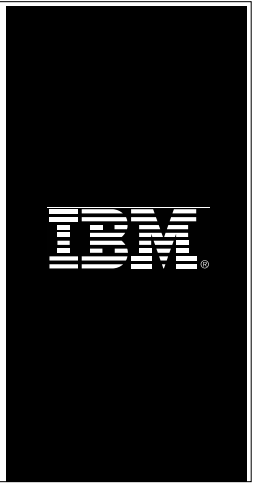
(0.2”spine)
0.17”<->0.473”
90<->249 pages

(0.1”spine)
0.1”<->0.169”
53<->89 pages

Implementation Guide for IBM Blockchain

SG24-8458-00

ISBN 0738458031

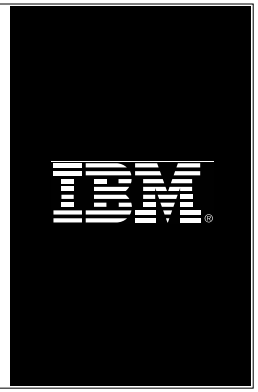


(2.5” spine)
2.5”<->nnn.n”
1315<-> nnnn pages

Implementation Guide for IBM Blockchain Platform for Multicloud

SG24-8458-00

ISBN 0738458031



(2.0” spine)
2.0” <-> 2.498”
1052 <-> 1314 pages

SG24-8458-00

ISBN 0738458031

Printed in U.S.A.