

IBM Cloud Private System Administrator's Guide

Ahmed Azraq
Wlodek Dymaczewski
Fernando Ewald
Luca Floris
Rahul Gupta
Vasfi Gucer
Anil Patil
Sanjay Singh
Sundaragopal Venkatraman
Dominique Vernier
Zhi Min Wen



In partnership with
IBM Academy of Technology



IBM Redbooks

IBM Cloud Private System Administrator's Guide

April 2019

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

First Edition (April 2019)

This edition applies to IBM Cloud Private Version 3.1.2.

© Copyright International Business Machines Corporation 2019. All rights reserved.

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

Contents

Notices	ix
Trademarks	x
Preface	xi
Authors	xii
Now you can become a published author, too	xv
Comments welcome	xvi
Stay connected to IBM Redbooks	xvi
Part 1. IBM Cloud Private overview, architecture, and installation	1
Chapter 1. Introduction to IBM Cloud Private.	3
1.1 IBM Cloud Private overview	4
1.2 IBM Cloud Private node types	6
1.2.1 Boot node	6
1.2.2 Master node	7
1.2.3 Worker node	7
1.2.4 Management node	8
1.2.5 Proxy node	8
1.2.6 VA (Vulnerability Advisor) node	9
1.2.7 An etcd node	10
1.3 IBM Cloud Private architecture	10
1.4 IBM Cloud Private features and benefits	12
1.4.1 A unified installer	12
1.4.2 Robust logging with ELK stack	12
1.4.3 Monitoring and alerts	13
1.4.4 Metering	13
1.4.5 Identify and access	13
1.4.6 Security	13
1.4.7 IBM Vulnerability Advisor	13
1.4.8 IBM Cloud Automation Manager	14
1.4.9 IBM Cloud Transformation Advisor	15
1.4.10 IBM Microclimate	16
1.4.11 IBM Cloud Private management console	16
1.4.12 Kubernetes	17
1.4.13 Private Docker image registry	17
1.4.14 Helm with enhanced security controls	17
1.4.15 Catalog	18
1.4.16 Kubernetes Service Catalog for managing service brokers	18
1.5 Helm	19
1.5.1 Helm components and terminology	19
1.5.2 Why you should use Helm	20
1.6 IBM Multicloud Manager	21
1.7 IBM Cloud Paks	22
1.8 IBM Cloud Private Editions	23
1.9 Persistent volumes	24
1.9.1 Volume and claim lifecycle	25
1.9.2 IBM Cloud Private Storage providers	26
1.10 IBM Cloud Private terms	26

Chapter 2. High availability installation	31
2.1 High availability considerations	32
2.1.1 Fault tolerance	32
2.1.2 Considerations for sizing the IBM Cloud Private cluster	33
2.1.3 Sample sizing for your IBM Cloud Private cluster	34
2.2 High Availability models for IBM Cloud Private cluster	35
2.2.1 Intra cluster	36
2.2.2 Intra cluster with multiple availability zones	36
2.2.3 Inter Cluster with federation on different availability zones	37
2.3 Performance considerations for IBM Cloud Private setup	38
2.3.1 Nodes considerations	38
2.3.2 Tuning the IBM Cloud Private setup	39
2.4 Step-by-step installation guide using Terraform	40
2.4.1 Environment preparation	42
2.4.2 Upload IBM Cloud Private binaries	43
2.4.3 Configure the Terraform template	46
2.4.4 Apply the Terraform template	51
2.5 Post installation verification	53
2.5.1 IBM Cloud Private command line interface	53
2.5.2 IBM Cloud Private Console user interface	57
2.6 Installing IBM Cloud Private on other Cloud platforms	63
2.6.1 Typical scenario of running IBM Cloud Private on other Cloud platforms	64
2.6.2 Installing IBM Cloud Private on AWS using Terraform	64
2.6.3 Installing IBM Cloud Private on Microsoft Azure using Terraform	64
2.6.4 Installing IBM Cloud Private on Google Cloud using Terraform	64
2.6.5 Installing IBM Cloud Private on RedHat OpenShift	64
2.6.6 Installing IBM Cloud Private on OpenStack Cloud provider	65
2.6.7 Installing IBM Cloud Private on VMware vSphere Cloud provider	65
2.6.8 Install IBM Cloud Private on existing Virtual Machines	65
2.7 Setting up IBM Cloud Private catalog in an airgap environment	65
2.7.1 Prerequisites	66
2.7.2 Steps to follow	66
2.8 Changing certificates post installation	67
Part 2. IBM Cloud Private system administration tasks	69
Chapter 3. Backup and restore of an IBM Cloud Private cluster	71
3.1 The purpose of backing up a cluster	72
3.2 Backup versus high availability, disaster recovery, and continuous availability	72
3.3 Backup options	73
3.3.1 Infrastructure backups	73
3.3.2 Platform backups	74
3.4 Backup and restore strategy	76
3.4.1 Infrastructure backup process	77
3.4.2 Infrastructure restore process	79
3.4.3 Platform backup process	84
3.4.4 Platform restore process	100
Chapter 4. Managing persistence in IBM Cloud Private	115
4.1 Designing the cluster for data persistence	116
4.1.1 Workload specific requirements	116
4.1.2 Maintainability requirements	117
4.1.3 Windows worker node support	117
4.2 Persistent storage for platform services	118

4.3	Configuring persistent storage for application containers	118
4.3.1	Configuring vSphere storage provider for IBM Cloud Private	119
4.3.2	Configuring NFS Storage for IBM Cloud Private	120
4.3.3	Configuring GlusterFS for IBM Cloud Private	125
4.3.4	Configuring Ceph and Rook for IBM Cloud Private	131
4.3.5	Configuring Portworx in IBM Cloud Private	140
4.3.6	Configuring Minio in IBM Cloud Private	147
4.4	Managing the storage hosted on IBM Cloud Private	147
4.4.1	Monitoring storage status and performance	147
4.4.2	Extending the available storage	150
4.5	Performance considerations	151
4.5.1	Performance test using dbench	151
4.5.2	PostgreSQL database performance	152
Chapter 5.	Logging and monitoring	153
5.1	Introduction	154
5.1.1	Elasticsearch, Logstash and Kibana	154
5.2	IBM Cloud Private Logging	155
5.2.1	ELK architecture	155
5.2.2	How Elasticsearch works	156
5.2.3	Default logging configuration	161
5.2.4	ELK security	163
5.2.5	Capacity planning	164
5.2.6	Role based access control	181
5.2.7	Using Kibana	182
5.2.8	Management	188
5.2.9	Forwarding logs to external logging systems	200
5.2.10	Forwarding logs from application log files	211
5.3	IBM Cloud Private Monitoring	222
5.3.1	How Prometheus works	222
5.3.2	How AlertManager works	225
5.3.3	How Grafana works	225
5.3.4	Accessing Prometheus, Alertmanager and Grafana dashboards	227
5.3.5	Configuring Prometheus Alertmanager and Grafana in IBM Cloud Private	227
5.3.6	Creating Prometheus alert rules	229
5.3.7	Configuring Alertmanager to integrate external alert service receivers	230
5.3.8	Using Grafana	233
Chapter 6.	Security	237
6.1	How IBM Cloud Private handles authentication	238
6.1.1	OIDC-based authentication	238
6.1.2	SAML-based authentication	239
6.2	How authorization is handled in IBM Cloud Private	239
6.2.1	Cloud resource names (CRN) specification	239
6.2.2	Role-based access control (RBAC) for pods	241
6.3	Isolation on IBM Cloud Private	241
6.3.1	Scenarios	242
6.4	The significance of the admission controller in IBM Cloud Private	246
6.4.1	Pod security policy	246
6.4.2	ResourceQuota	247
6.4.3	LimitRange	248
6.4.4	AlwaysPullImages	248
6.5	Image security	248

6.5.1 Pushing and pulling images	249
6.5.2 Enforcing container image security	250
Chapter 7. Networking	257
7.1 Introduction to container networking	258
7.2 Pod network	259
7.2.1 Calico	259
7.2.2 NSX-T	261
7.3 High availability	262
7.3.1 External load balancer	262
7.3.2 Virtual IP addresses	263
7.3.3 Ingress controller	265
7.4 Service discovery (kube-dns)	270
7.4.1 Headless services	272
7.4.2 External services	272
Chapter 8. Troubleshooting	273
8.1 Common errors during the IBM Cloud Private installation	274
8.1.1 Customizing the config.yaml file	274
8.1.2 Customizing the /cluster/hosts file	274
8.1.3 SSH key error	275
8.1.4 Missing the IBM Cloud Private binary files in the installation folder	275
8.1.5 Missing the minimum system requirements	276
8.1.6 Perform the system cleanup when the installation fails	276
8.2 Network configuration errors	277
8.2.1 Calico troubleshooting	277
8.2.2 IPsec troubleshooting	279
8.3 Common errors when installing a Helm chart	281
8.3.1 When accessing an application getting the 504 error	281
8.3.2 No CPU available	283
8.3.3 The required port is in use	285
8.3.4 Deployment fails due to a missing permission	286
8.4 Common errors when running applications	286
8.4.1 Getting the 504 or 500 errors when trying to access the application	286
8.5 Opening a support case	287
Chapter 9. Service mesh implementation using Istio	289
9.1 Overview	290
9.2 Role of the service mesh	290
9.2.1 Service registry	291
9.2.2 Service discovery	291
9.2.3 Load balancing	291
9.2.4 Traffic encryption	291
9.2.5 Observability and traceability	291
9.2.6 Access control	292
9.2.7 Circuit breaker pattern support	292
9.3 Istio architecture	292
9.3.1 Components	292
9.3.2 Istio functions	295
9.4 Installation of Istio and enabling the application for Istio	295
9.4.1 Install Istio with the helm command	296
9.4.2 Enable application for Istio	298
9.4.3 Uninstallation	301
9.5 Service resiliency	301

9.5.1	Retry	302
9.5.2	Timeout	305
9.5.3	Load balancer	307
9.5.4	Simple circuit breaker	307
9.5.5	Pool ejection	311
9.6	Achieving E2E security for microservices using Istio	311
9.6.1	Inbound traffic	311
9.6.2	Outbound traffic	313
9.6.3	Mutual TLS authentication	315
9.6.4	White or black listing	317
9.6.5	Istio authorization	319

Part 3. Cloud Foundry related topics 323

Chapter 10. IBM Cloud Private Cloud Foundry and common systems administration tasks		325
10.1	Introduction	326
10.1.1	IaaS flavors	326
10.1.2	Technology BOSH versus Kubernetes	326
10.2	Installation and extensions	327
10.2.1	Installation of the installer container in a Cloud Foundry Full Stack environment	327
10.2.2	Installation of the installer container in a CFEE environment	328
10.2.3	Config-manager role	329
10.2.4	Extensions	330
10.3	High availability installation	333
10.3.1	Zoning	333
10.3.2	External database	334
10.3.3	External objects store	335
10.4	Backup and restore strategy	335
10.4.1	Installation data	335
10.4.2	Director	335
10.4.3	Cloud Foundry database	336
10.5	Storage and persistent volumes	336
10.5.1	Cloud Foundry Full Stack	336
10.5.2	Cloud Foundry Enterprise Environment (CFEE) technology preview	337
10.6	Sizing and licensing	337
10.7	Networking	338
10.8	Security	338
10.8.1	TLS encryption	338
10.8.2	Inbound routing	339
10.8.3	Credentials and certificates	339
10.9	Monitoring and logging	340
10.9.1	Monitoring	340
10.9.2	Logging	341
10.10	Integrating external services	342
10.10.1	IBM Cloud Private services	342
10.10.2	IBM Cloud services	343
10.10.3	Legacy services	343
10.11	Applications and buildpacks	343
10.11.1	Installing extra buildpacks	343
10.11.2	Application for an airgap environment	344
10.12	iFix and releases	344

10.12.1 Zero downtime	345
Appendix A. Command line tools	347
Helm command line interface (helmcli)	348
Installing the Helm CLI	348
Verifying the installation	348
Using helmcli.	349
IBM Cloud Private CLI (cloudctl)	350
Installing the IBM Cloud Private CLI	350
General cloudctl commands	351
cloudctl catalog commands	352
cloud cm commands	353
Kubectl	353
kubectl get	355
kubectl get namespace	356
kubectl get nodes	356
kubectl get pods	356
kubectl logs	357
kubectl describe	358
Cheat sheet for production environment	361
Use kubectl drain to remove a node from service	361
Enabling autocomplete for kubectl	361
Removing a pod from a service	361
Editing kubernetes resources	361
Taints and tolerations	362
Viewing and finding resources	362
Updating resources	363
Scaling resources	363
Interacting with running pods	363
Additional kubectl commands	364
Appendix B. Additional material	365
Locating the GitHub material	365
Cloning the GitHub material	365
Related publications	367
IBM Redbooks	367
Online resources	367
Help from IBM	368

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.

Cognos®	IBM®	Redbooks (logo)  ®
DataPower®	Lotus®	SPSS®
DataStage®	Passport Advantage®	Tivoli®
Domino®	PowerPC®	WebSphere®
Global Business Services®	Redbooks®	
IBM Watson™	Redpapers™	

The following terms are trademarks of other companies:

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, 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.

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

Preface

IBM® Cloud Private is an application platform for developing and managing containerized applications across hybrid cloud environments, on-premises and public clouds. It is an integrated environment for managing containers that includes the container orchestrator Kubernetes, a private image registry, a management console, and monitoring frameworks.

This IBM Redbooks® covers tasks performed by IBM Cloud Private system administrators, such as installation for high availability, configuration, backup and restore, using persistent volumes, networking, security, logging and monitoring, Istio integration, troubleshooting and so on.

The authors team has many years of experience in implementing IBM Cloud Private and other cloud solutions in production environments, so throughout this document we took the approach of providing you the recommended practices in those areas.

As part of this project, we also developed several code examples. You can download those from the IBM Redbooks GitHub location (<https://github.com/IBMRedbooks>).

If you are an IBM Cloud Private system administrator, this book is for you. If you are developing applications on IBM Cloud Private, you can see the IBM Redbooks publication *IBM Cloud Private Application Developer's Guide*, SG24-8441.

Authors

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



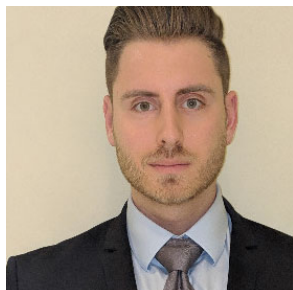
Ahmed Azraq is a Cloud Solutions Leader in IBM. He works as part of IBM Cloud Solutioning Center, and his primary responsibility is to help clients across Middle East and Africa (MEA) and Asia Pacific to realize their journey towards adopting IBM Cloud and IBM Watson. Since joining IBM in 2012, Ahmed worked as a technical team leader and architect. He has successfully led and won several IBM Cloud Private deals as a pre-sales architect in the region, and eagerly participated in Academy of Technology studies related to IBM Cloud Private. Ahmed has acquired several professional certifications, including Open Group IT Specialist, and contributed to developing and authoring six professional IBM Cloud Certification Exams. Ahmed has also delivered training on IBM Cloud, DevOps, hybrid cloud Integration, Node.js, and Watson APIs to IBM clients, IBM Business Partners, university students, and professors around the world. He is the recipient of several awards, including Eminence and Excellence Award in the IBM Watson worldwide competition Cognitive Build, the IBM Service Excellence Award for showing excellent client value behaviors, and knowledge-sharing award. Ahmed is a platinum IBM Redbooks author and has authored several other IBM Redbooks publications.



Wlodek Dymaczewski is a Cloud Solution Architect in IBM Poland. He has over 25 years of experience in IT in the field of systems and network management. He was part of IBM Cloud unit since the beginning, focusing on hybrid cloud management and devops solutions. Since 2017, he works as IBM Cloud Private Technical Lead for Central and Eastern Europe. Wlodek holds MSc degree from Poznań University of Technology and MBA degree from Warwick Business School.



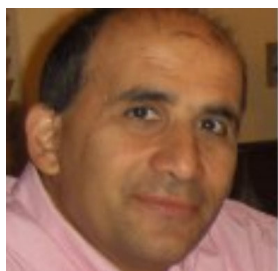
Fernando Ewald holds a Bachelor Degree in Computer Science and a Masters Degree in Computer Networking. Currently he works as the IBM Cloud Private Technical Team Lead - L2 Support in IBM Austin. He has over 20 years of experience in IT solutions. Fernando joined IBM Brazil in 2009 and there he was a member of the Innovation and Technical Leadership Team. In 2016, Fernando relocated to Austin, TX to work at the Hybrid Cloud division with DevOps tools and in 2018 he became the Technical Team Lead for IBM Cloud Private Support L2 team. Before joining IBM, Fernando worked with a Sugar and Alcohol Company, creating high availability solutions for the industry and headquarters. He also has worked as a Teacher at Universidade de Franca - Brazil, where he taught at the Computer Science and System of Information Bachelor courses.



Luca Floris is a Cloud Technical Consultant in IBM EMEA. Luca joined IBM 4 years ago as a graduate through the IBM Graduate Scheme and he has 8 years of experience in IT, graduating from Plymouth University with a degree in Computer Science. His primary focus is containerization and application modernization through IBM Cloud Private and related technologies. He is well recognized as a technical focal point for IBM Cloud Private from IBM Cloud Innovation Labs in EMEA, and has written extensively about IBM Cloud Private on Kubernetes.



Rahul Gupta is a Cloud Native Solutions Architect in IBM Cloud Solutioning Centre in USA. Rahul is a 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 a IBM Master Inventor and also works on MQTT protocol in 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 23 years of experience in the areas of cloud computing, 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 three years. Vasfi is also an IBM Certified Senior IT Specialist, Project Management Professional (PMP), IT Infrastructure Library (ITIL) V2 Manager, and ITIL V3 Expert.



Anil Patil is a senior Solution Architect at IBM US. He is a Certified Cloud Solution Architect and Solution Advisor - DevOps with more than 18 years of IT experience in Cognitive Solutions, IBM Cloud, Microservices, IBM Watson API, and Cloud-Native Applications. His core experience is in Microservices, AWS, Cloud Integration, API Development, and Solution Architecture. He is currently a Lead Solution Architect and Cloud Architect for various clients in North America. Anil has been a Redbooks author and technical contributor for various IBM material and blogs, such as Cognitive Solution, IBM Cloud, API Connect, and Docker Container.



Sanjay Singh is a senior Software Engineer. He has worked on various cloud products over 11 years of his career with IBM. As a technical consultant and Lab Advocate he has been engaged with customers in making their Cloud Adoption and implementation successful. He has lead adoption of IBM Cloud Private in one of the largest Telecom providers in India over eight months of engagement. His core experience is in Kubernetes, Openstack, AWS, Node.JS, JavaScript, Python, and Scripting. He holds a Btech degree in Computer Science from NIT Trichy (India).



Sundaragopal Venkatraman (Sundar) is a cloud evangelist and a Thought Leader on application infrastructure, application modernization, performance, scalability, and high availability of enterprise solutions. With experience spanning over two decades, he has been recognized as a trusted advisor to various MNC Banks & other IBM clients in India/Asia-Pacific. He has been recognized as a technical focal point for IBM Cloud Private from IBM Cloud Innovation Labs in India. He has delivered deep dive sessions on IBM Cloud Private on International forums and conducts boot camps worldwide. He pioneered the IBM Proactive Monitoring toolkit, a lightweight Monitoring solution which was highlighted on IBM showcase events. He has authored various IBM Redbooks publications and Redpapers, and is a recognized author. He thanks his wife and his family for supporting him in all his endeavors.



Dominique Vernier is a Senior Software Developer in IBM US. He holds a degree in Computer Science from the Université libre de Bruxelles. Over his 30 years of IT experience he covered different technologies including system integration, service-oriented architecture, and Cloud public and private as a developer, IT architect, analyst, and technical sales with worldwide positions. He has a broad knowledge of IT environments, working in banking, supply chain management, telecommunications, and other IT Sectors. He has written several articles on developer Works, IBM Social media platform, and his own blog. During his carrier, he applied for a number of patents. His current expertise is on the private cloud, where he automates the installation of Cloud Foundry on IBM Cloud Private.



Zhi Min Wen is a senior managing consultant at IBM Singapore. He has 20+ years of experience in IT industry specialised in IT service management and cloud. He is passionate about open source solutions, and he enjoys exploring the new edge of technology. He was an early explorer of Docker and Kubernetes, and he has written extensively on IBM Cloud Private and Kubernetes. He attained IBM Outstanding Technical Achievement Awards 2018.

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

Ann Lund, Erica Wazewski
IBM Redbooks, Poughkeepsie Center

Robin Hernandez, Atif Siddiqui, Jeff Brent, Budi Darmawan. Eduardo Patrocinio, Eswara Kosaraju, David A Weilert, Aman Kochhar, Surya V Duggirala, Kevin Xiao, Brian Hernandez, Ling Lan, Eric Schultz, Kevin G Carr, Nicholas Schambureck, Ivory Knipfer, Kyle C Miller, Sam Ellis, Rick Osowski, Justin Kulikauskas, Christopher Doan, Russell Kliegel, Kip Harris
IBM USA

Juliana Hsu, Radu Mateescu, Stacy Pedersen, Jeffrey Kwong
IBM Canada

Brad DesAulniers, Mihai Criveti, Hans Kristian Moen
IBM Ireland

Raffaele Stifani
IBM Italy

Ahmed Sayed Hassan
IBM Singapore

Santosh Ananda, Rachappa Goni, Shajeer Mohammed, Sukumar Subburaj, Dinesh Tripathi
IBM India

Qing Hao, Xiao Yong, AZ Zhang
IBM China

The team would also like to express thanks to the following IBMers for contributing content to the Cloud Foundry section of the book while continuing to develop the next IBM Cloud Private Cloud Foundry release:

Chris Ahl, Subbarao Meduri
IBM US

Kevin Cormier, Roke Jung, Colton Nicotera, Lindsay Martin, Joshua Packer
IBM Canada

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:

ibm.com/redbooks

- Send your comments in an email:

redbooks@us.ibm.com

- Mail your comments:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Part 1

IBM Cloud Private overview, architecture, and installation

This first part of the book provides an overview of IBM Cloud Private including the architecture and major features. It also covers installing IBM Cloud Private in a high availability configuration.



Introduction to IBM Cloud Private

This chapter provides an introduction to IBM Cloud Private and related technologies and has the following sections:

- ▶ 1.1, “IBM Cloud Private overview” on page 4
- ▶ 1.2, “IBM Cloud Private node types” on page 6
- ▶ 1.3, “IBM Cloud Private architecture” on page 10
- ▶ 1.4, “IBM Cloud Private features and benefits” on page 12
- ▶ 1.5, “Helm” on page 19
- ▶ 1.6, “IBM Multicloud Manager” on page 21
- ▶ 1.7, “IBM Cloud Paks” on page 22
- ▶ 1.8, “IBM Cloud Private Editions” on page 23
- ▶ 1.9, “Persistent volumes”
- ▶ 1.10, “IBM Cloud Private terms” on page 26

GitHub materials: If you’d like to follow the code examples in this IBM Redbooks publication, you can download the GitHub repository of this book. See Appendix B, “Additional material” on page 365 for instructions.

1.1 IBM Cloud Private overview¹

IBM Cloud Private is an application platform for developing and managing containerized applications across hybrid cloud environments, on-premises and public clouds. It is an integrated environment that includes the container orchestrator Kubernetes, a private image registry, a management console, and monitoring frameworks.

IBM Cloud Private is a next-generation, pre-packaged, enterprise-class solution and platform for developing and managing containerized applications. This integrated environment can be deployed behind firewalls, and managed or controlled by whomever the enterprise determines. It is built on Kubernetes.

With a lightweight footprint and powerful platform capabilities, IBM Cloud Private enables enterprises to unleash their developmental creativity, using industry-common technologies and process guidance, in a minimal time frame.

Platform technologies enabling cloud native development include Docker containers and Kubernetes with integrated operations management for security, logging, monitoring and event management. IBM Cloud Private also provides access to necessary application runtimes and data services.

Figure 1-1 shows the IBM Cloud Private capabilities.

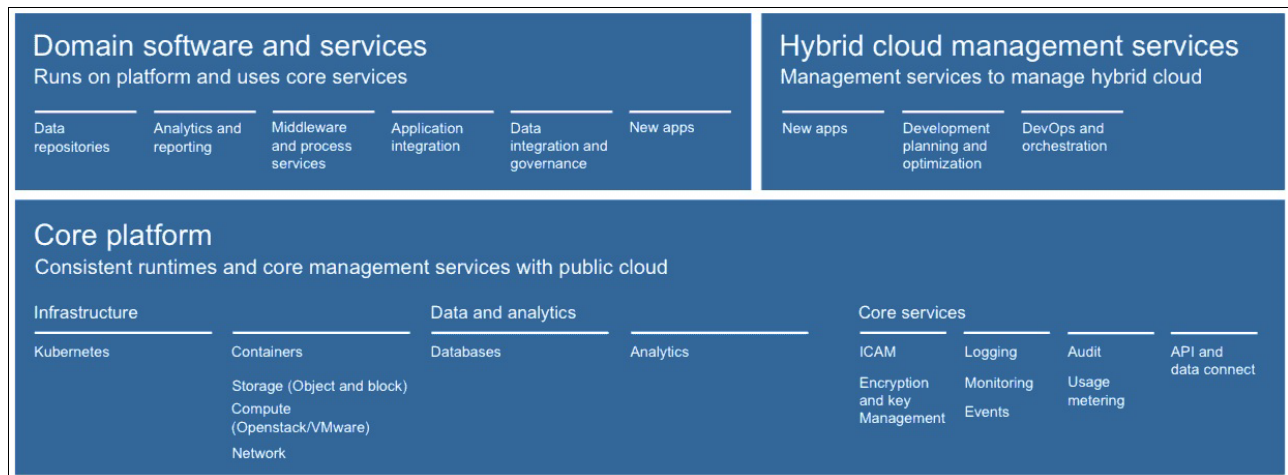


Figure 1-1 IBM Cloud Private capabilities

Use cases for IBM Cloud Private include the following situations:

- ▶ Create new cloud-native apps.
- ▶ Modernize your existing apps on cloud.
- ▶ Open your data center to work with cloud services.

See [What is IBM Cloud Private](#) for more information.

IBM Cloud Private is differentiated by providing production application services, application runtimes, data and analytics services, messaging services, caching services, plus many more that are necessary for developers to quickly and iteratively innovate based on their enterprise business needs.

¹ Parts of this section are based on the whitepaper “IBM Cloud Private: The cloud-native application platform for the enterprises” written by Raffaele Stifani (Executive Architect - Software Group) from IBM Italy.

This private cloud platform is part of the broader IBM Cloud portfolio and provides choice with consistency across IBM public, private, and dedicated cloud models. IBM Cloud Private delivers a choice of open-source application runtimes, consistent with IBM public cloud offerings: Kubernetes and containers or Cloud Foundry technology.

Your enterprise can choose the prescriptive development approach of Cloud Foundry, or the more customizable and portable approach of Kubernetes and Docker Containers.

Along with the application runtime frameworks, IBM delivers a core set of management services for these frameworks and the applications being developed on top. Some examples of the management services include logging, monitoring, access control, and event management.

Enterprises can use these management tools integrated with the platform and ready to use. These are tools frequently used by enterprise clients today and leverage existing skills. If needed, these tools can be integrated with enterprise instantiations, so that the management needs are operationalized from one location.

One of the most beneficial aspects of the IBM Cloud Private platform is the application services that move innovation from idea to reality. As shown in Figure 1-2, IBM Cloud Private includes services for data, messaging, Java, integration, Blockchain, DevOps, analytics, and others. These services are crucial for enterprise application creation, and with IBM Cloud Private they can be deployed rapidly and ready to accelerate new ideas.

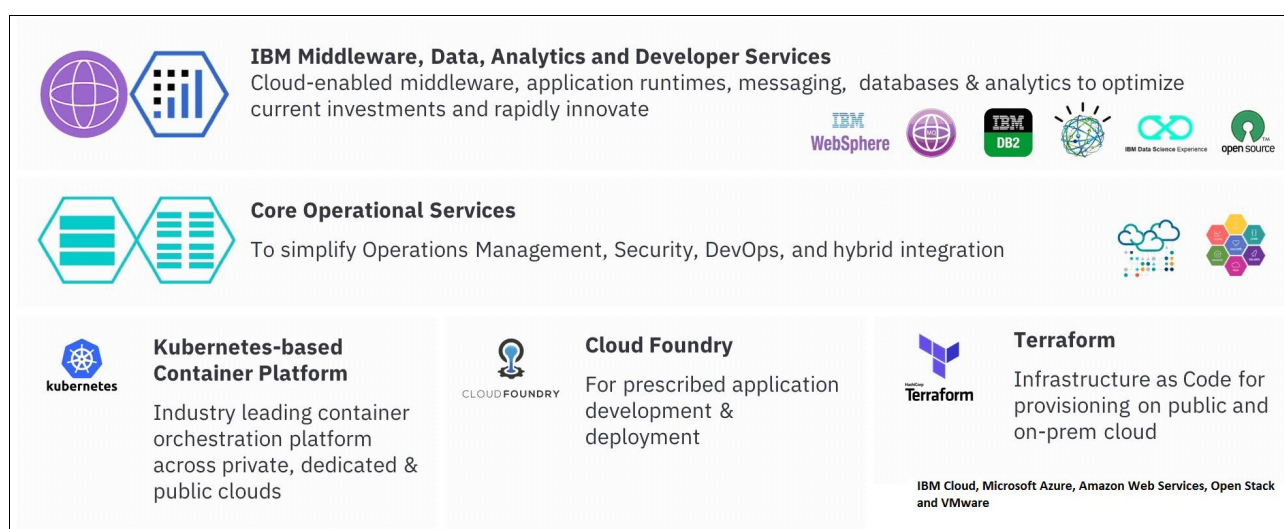


Figure 1-2 IBM Cloud Private components

IBM Cloud Private supports choice in application development with Kubernetes, Cloud Foundry, and function-based programming models. It provides these benefits:

- Containers and orchestration that are based on Kubernetes for creating microservices-based applications.
- A common catalog of enterprise and open services to accelerate developer productivity.
- A choice of compute models for rapid innovation, including Kubernetes and Cloud Foundry.
- Common base services to support the scalable management of microservices, including Istio, monitoring with Prometheus, logging with Elasticsearch, Logstash, and Kibana (ELK).
- Automatic horizontal and non-disruptive vertical scaling of applications.

- ▶ Network and storage policy-based controls for application isolation and security.
- ▶ Automated application health checking and recovery from failures.
- ▶ Support over IaaS infrastructure, including OpenStack and VMware.
- ▶ Through the Cloud Automation Manager component, support for Terraform and Chef to allow the orchestration of infrastructure.

1.2 IBM Cloud Private node types

An IBM Cloud Private cluster has the following node types:

- ▶ Boot node
- ▶ Master node
- ▶ Worker node
- ▶ Management node
- ▶ Proxy node
- ▶ VA node
- ▶ etcd node

1.2.1 Boot node

A boot or bootstrap node is used for running installation, configuration, node scaling and cluster updates. Only one boot node is required for any cluster. A single node can be used for both master and boot. See Figure 1-3.

Note: In the following images, the clusters represent minimal IBM Cloud Private configurations. Actual production configurations can vary.

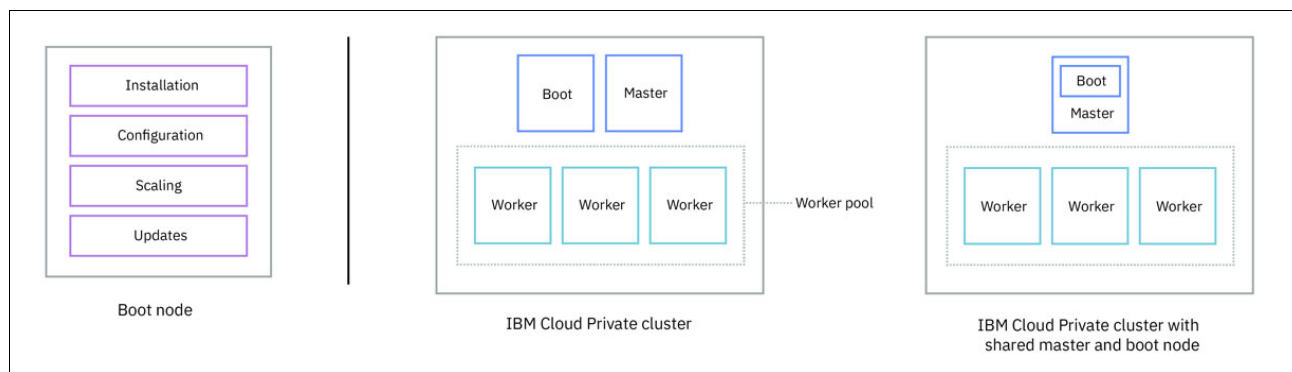


Figure 1-3 Boot node

Note: You can use a single boot node for multiple clusters. In such a case, the boot and master cannot be on a single node. Each cluster must have its master node. On the boot node, you must have a separate installation directory for each cluster. If you are providing your own certificate authority (CA) for authentication, you must have a separate CA domain for each cluster.

1.2.2 Master node

A master node provides management services and controls the worker nodes in a cluster. Master nodes host processes that are responsible for resource allocation, state maintenance, scheduling, and monitoring. Multiple master nodes are used in a high availability (HA) environment to allow for failover if the leading master host fails.

Hosts that can act as the master are called master candidates. See Figure 1-4.

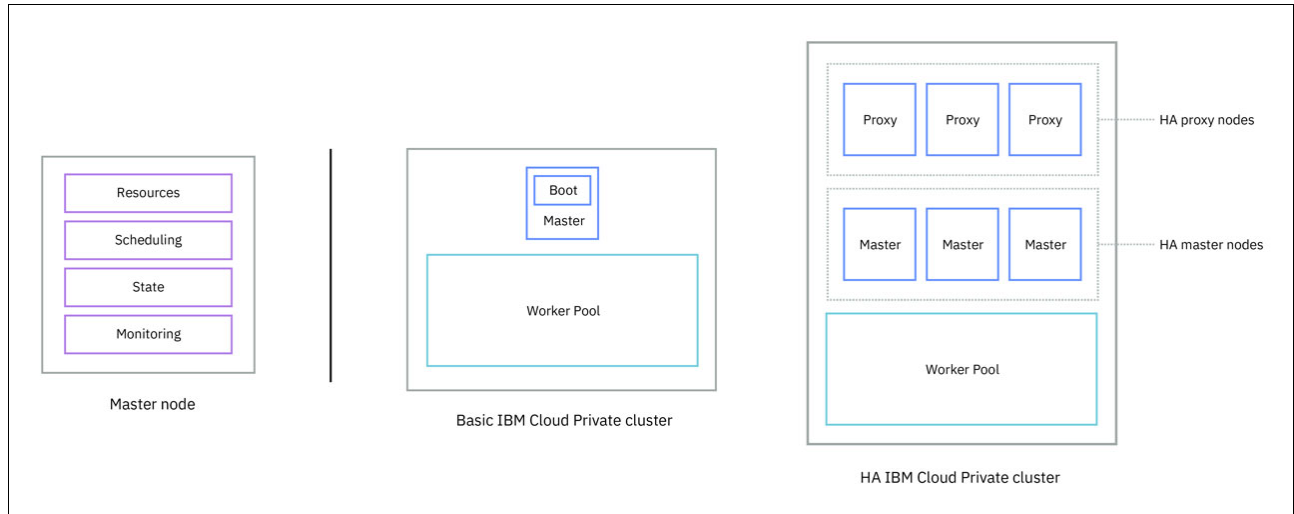


Figure 1-4 Master node

Note: if you do not specify a separate proxy, management, or etcd node, those components are also handled by the master node.

1.2.3 Worker node

A worker node is a node that provides a containerized environment for running tasks. As demands increase, more worker nodes can easily be added to the cluster to improve performance and efficiency. A cluster can contain any number of worker nodes, but a minimum of one worker node is required. See Figure 1-5.

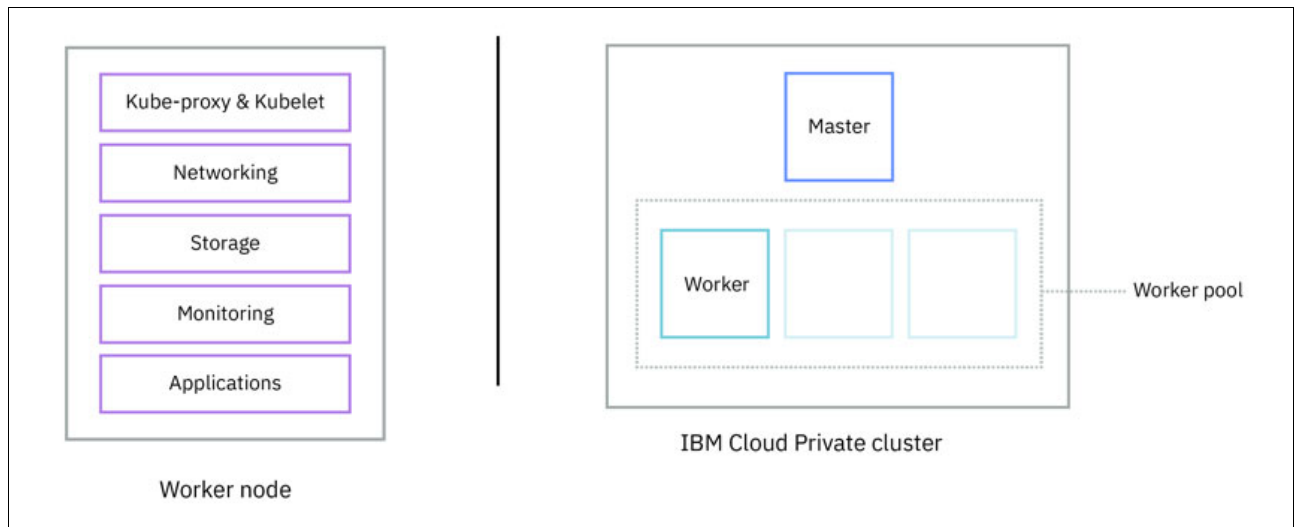


Figure 1-5 Worker node

1.2.4 Management node

A management node is an optional node that only hosts management services, such as monitoring, metering, and logging. By configuring dedicated management nodes, you can prevent the master node from becoming overloaded. See Figure 1-6.

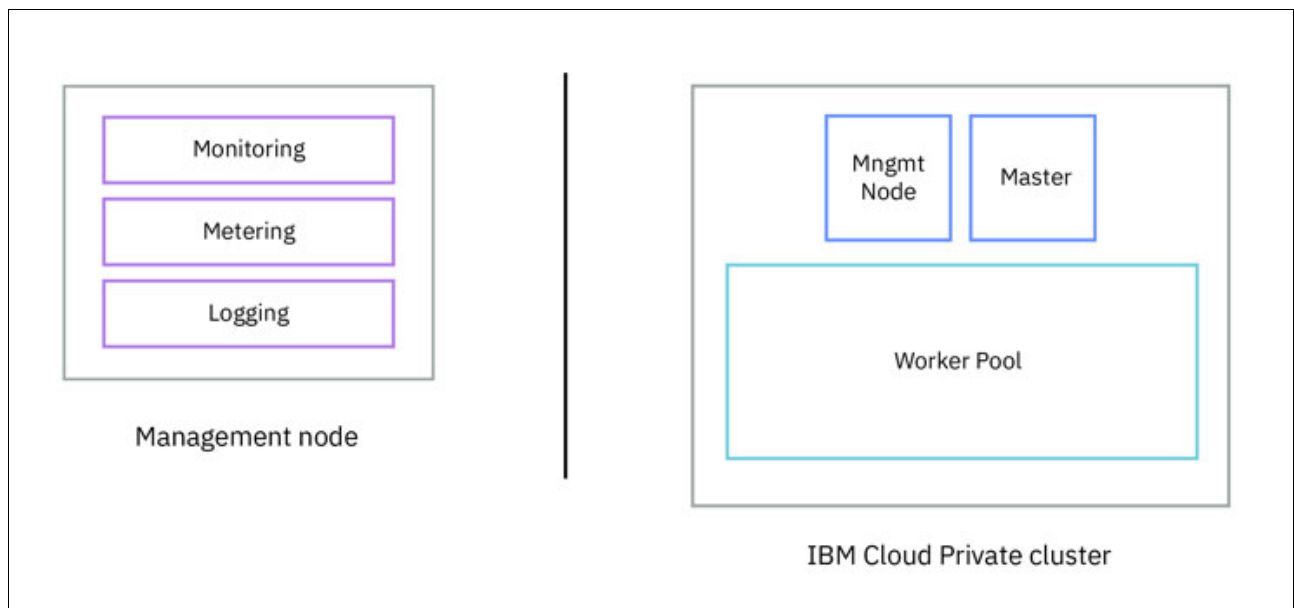


Figure 1-6 Management node

1.2.5 Proxy node

A proxy node is a node that transmits external requests to the services created inside the cluster. Multiple proxy nodes are deployed in a high availability (HA) environment to allow for failover if the leading proxy host fails. While a single node is used as both master and proxy, it is best to use dedicated proxy nodes to reduce the load on the master node. A cluster must contain at least one proxy node if load balancing is required inside the cluster. See Figure 1-7 on page 9.

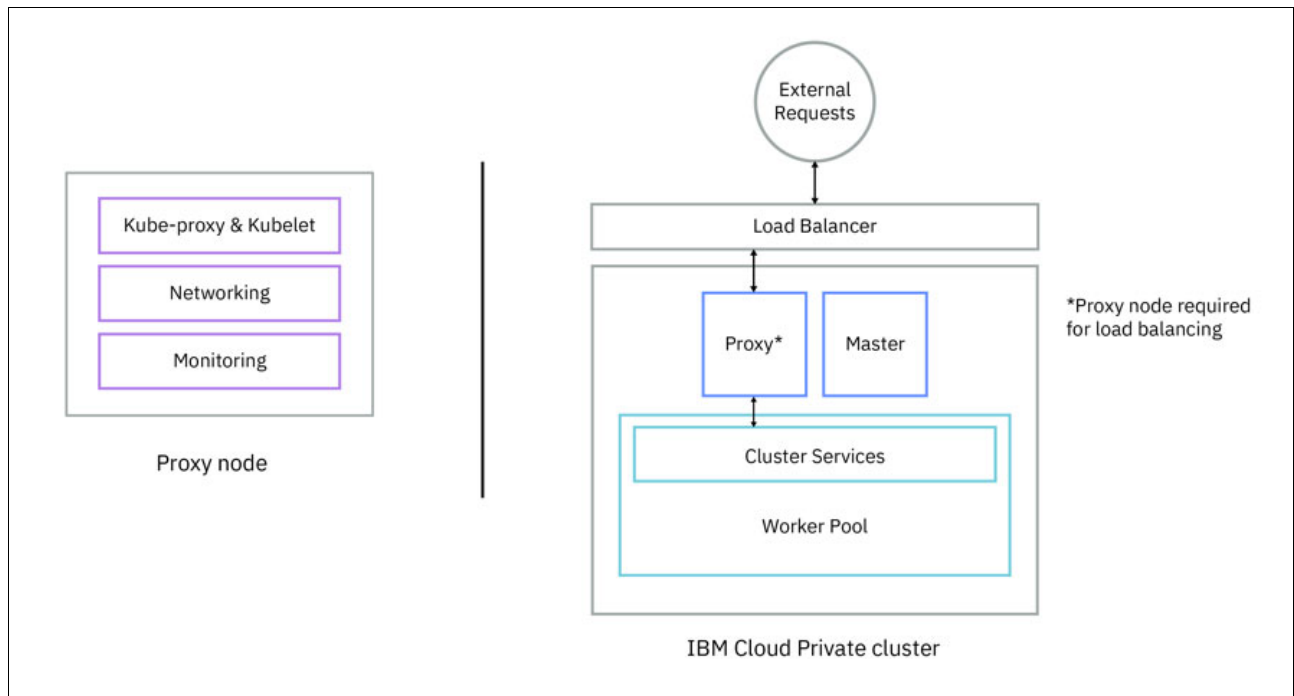


Figure 1-7 Proxy node

1.2.6 VA (Vulnerability Advisor) node

A VA (Vulnerability Advisor) node is an optional node that is used for running the Vulnerability Advisor services. Vulnerability Advisor services are resource intensive. If you use the Vulnerability Advisor service, specify a dedicated VA node. See Figure 1-8.

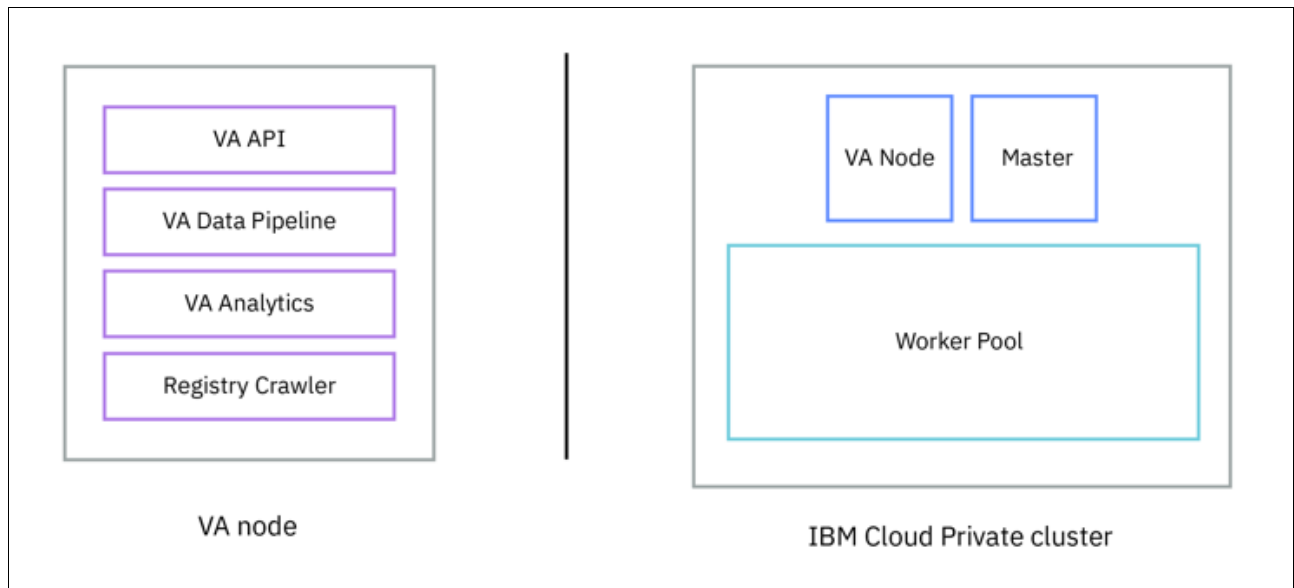


Figure 1-8 VA node

1.2.7 An etcd node

An etcd node is an optional node that is used for running the etcd distributed key value store. Configuring an etcd node in an IBM Cloud Private cluster that has many nodes, such as 100 or more, helps to improve the etcd performance. See Figure 1-9.

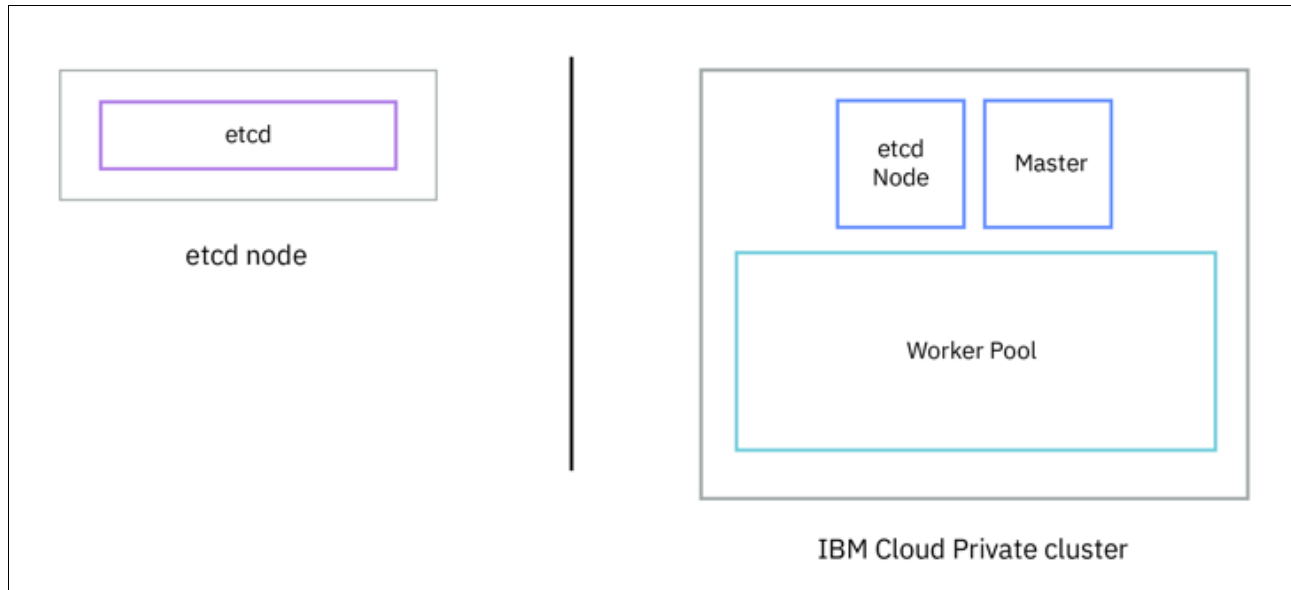


Figure 1-9 etcd node

Supported platforms: For the most recent information on supported platforms for each IBM Cloud Private version 3.1.2 node type, see the [Supported operating systems and platforms](#) IBM Knowledge Center link.

1.3 IBM Cloud Private architecture

The IBM Cloud Private architecture provides container-as-a-service (*CaaS*) and platform-as-a-service (*PaaS*) capabilities.

Figure 1-10 on page 11 shows the IBM Private Cloud architecture.

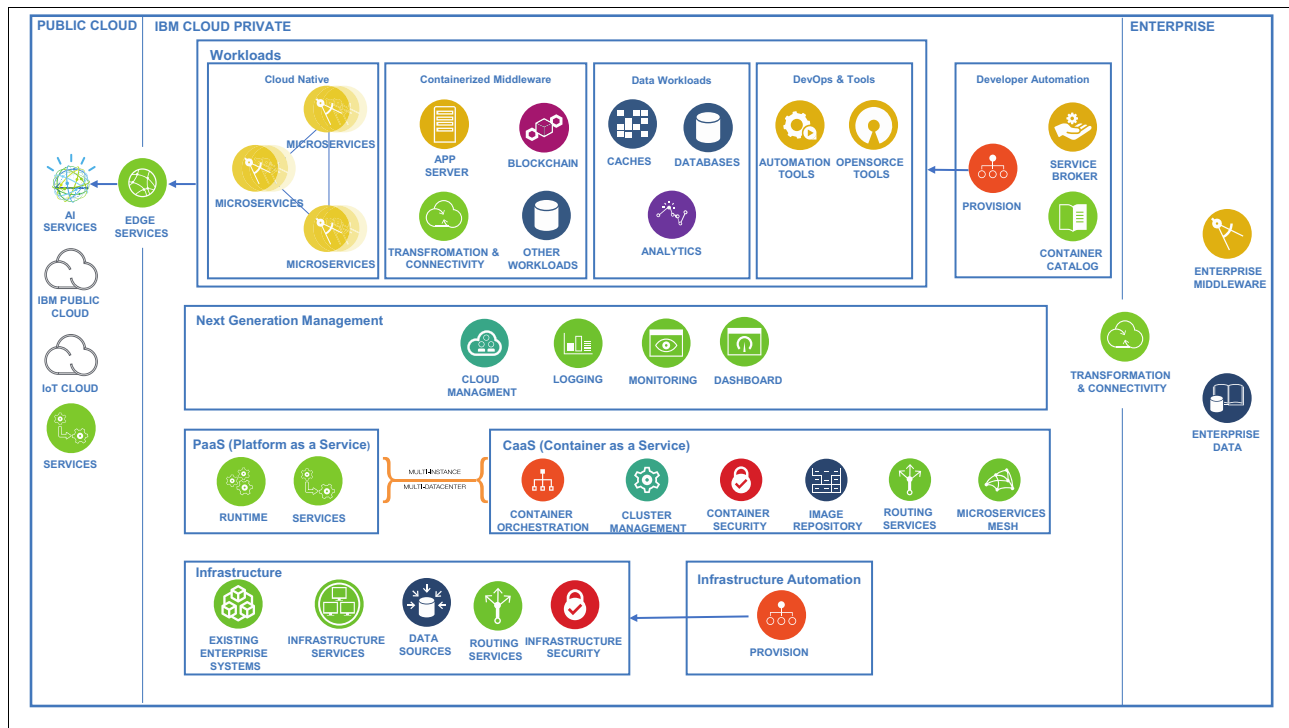


Figure 1-10 IBM Private Cloud architecture²

The architecture provides several benefits, as shown in Figure 1-10:

- ▶ Container orchestration, which is at the core of the architecture; this layer provides cluster management, security capabilities, image repositories, routing services and microservices mesh.
- ▶ A PaaS layer, which can enhance a container environment by providing higher-level runtimes and service bindings that allow for an easier development experience.
- ▶ The CaaS and PaaS layer, which sit over an infrastructure layer that provides compute through virtual machines, network, storage and security.
- ▶ Automation and orchestration for the underlying infrastructure, allowing for an infrastructure-neutral software-defined data center; infrastructure automation can provide predefined infrastructure templates to create repeatable patterns.
- ▶ A private cloud platform provides monitoring for container-based applications to provide logging, dashboards and automation; it supports network and storage policy-based controls for application isolation and security, and automated application health checking and recovery from failures.
- ▶ The ability to run containerized workloads for several patterns, such as cloud-native, data workloads, integration workloads, tooling workloads and some middleware, such as Java Application Server.
- ▶ A developer catalog of workloads that you can provision over containers to automate the developer experience.

² Image taken from IBM Cloud Architecture Center
<https://www.ibm.com/cloud/garage/architectures/private-cloud/reference-architecture>

The private cloud architecture is supported by IBM Cloud Private, as shown in Figure 1-11. By supporting Kubernetes and Cloud Foundry, IBM Cloud Private provides choice in application development. You get a wealth of content that can be containerized, tools for end-to-end automation, and management tools.

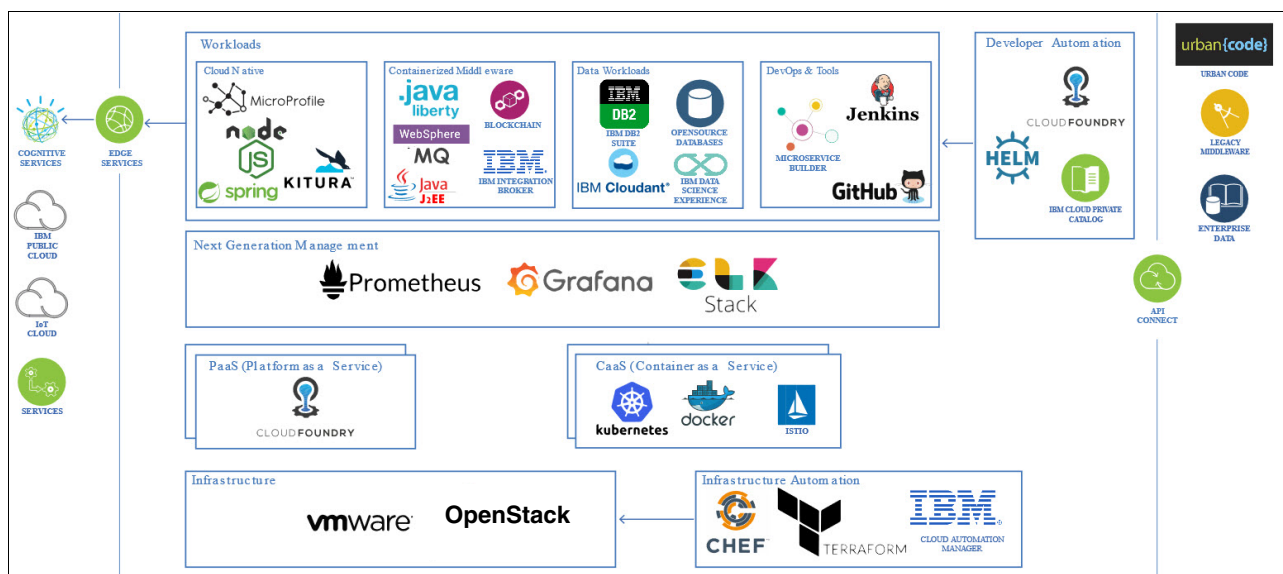


Figure 1-11 IBM Private Cloud architecture with product names³

For more information about the IBM Cloud Private architecture, visit the following IBM Cloud Architecture link:
<https://www.ibm.com/cloud/garage/architectures/private-cloud/reference-architecture>

1.4 IBM Cloud Private features and benefits

The following sections describe the major features in IBM Cloud Private.

1.4.1 A unified installer

Rapidly set up a Kubernetes based cluster that contains master, worker, proxy, and optional management and Vulnerability Advisor nodes by using an Ansible based installer. This Ansible based installer is fast and simple to use. Run a few simple commands from a single boot node, and your cluster is up and running in a few minutes.

See Chapter 2, “High availability installation” on page 31 for details on installing IBM Cloud Private.

1.4.2 Robust logging with ELK stack

Logs are critical for debugging and post-mortem in production failures. Twelve-factor applications break down into many microservices, which increases the number of logs across the containers you need to debug. IBM Cloud Private uses the ELK (Elasticsearch, Logstash, Kibana) stack and Filebeat for monitoring and logging.

³ Image taken from IBM Cloud Architecture Center
<https://www.ibm.com/cloud/garage/architectures/private-cloud/reference-architecture>

This process provides a centralized store for all logs and metrics, better performance, and increased stability when you access and query logs and metrics. You can use the results from these queries to produce insightful graphs and reports: that is the dashboard part.

See Chapter 5, “Logging and monitoring” on page 153 for more information on the ELK stack.

1.4.3 Monitoring and alerts

Every container must have its health monitored. Basic liveness probes in Kubernetes ensure failed pods are restarted.

IBM Cloud Private configures custom Prometheus collectors for custom metrics. Custom metrics help provide insights and building blocks for customer alerts and custom dashboards. IBM Cloud Private uses a Prometheus and Grafana stack for system monitoring.

See Chapter 5, “Logging and monitoring” on page 153 for more information on monitoring and alerts in IBM Cloud Private.

1.4.4 Metering

Every container must be managed for license usage. You can use the metering service to view and download detailed usage metrics for your applications and cluster. Fine-grained measurements are visible through the metering UI and the data is kept for up to three months. Monthly summary reports are also available for you to download and are kept for up to 24 months.

1.4.5 Identify and access

IBM Cloud Private introduces the concept of teams on top of raw Kubernetes roles and cluster roles. Teams bind a collection of resources, both inside and outside of Kubernetes, to a set of users with defined roles. The team model is based on the access control model from IBM UrbanCode Deploy.

See Chapter 6, “Security” on page 237 for more information on this topic.

1.4.6 Security

IBM Cloud Private ensures data in transit and data at rest security for all platform services. All services expose network endpoints via TLS and store data which is encrypted at rest. You need to configure IPsec and dm-crypt in order to accomplish that.

All services must provide audit logs for actions performed, when they were performed, and who performed the action. The security model ensures consistent audit trails for all platform services and compliance across all middleware.

See Chapter 6, “Security” on page 237 for details on managing security in IBM Cloud Private.

1.4.7 IBM Vulnerability Advisor

Vulnerability Advisor checks the security status of container images that are provided by IBM, third parties, or added to your organization’s registry namespace. If the Container Scanner is installed in each cluster, Vulnerability Advisor also checks the status of containers that are running.

When you add an image to a namespace, the image is automatically scanned by Vulnerability Advisor to detect security issues and potential vulnerabilities. If security issues are found, instructions are provided to help fix the reported vulnerability.

Vulnerability Advisor provides security management for IBM Cloud Container Registry, generating a security status report that includes suggested fixes and best practices.

Any issues that are found by Vulnerability Advisor result in a verdict that indicates that it is not advisable to deploy this image. If you choose to deploy the image, any containers that are deployed from the image include known issues that might be used to attack or otherwise compromise the container. The verdict is adjusted based on any exemptions that you specified. This verdict can be used by Container Image Security Enforcement to prevent the deployment of nonsecure images in IBM Cloud Kubernetes Service.

Fixing the security and configuration issues that are reported by Vulnerability Advisor can help you to secure your IBM Cloud infrastructure.

1.4.8 IBM Cloud Automation Manager

IBM Cloud Automation Manager (CAM) is a multi-cloud, self-service management platform running on IBM Cloud Private that empowers developers and administrators to meet business demands. This platform allows to efficiently manage and deliver services through end-to-end automation while enabling developers to build applications aligned with enterprise policies.

IBM Cloud Private with Cloud Automation Manager provides choice and flexibility for multiple IT across the organization to build and deliver applications and application environments into production more quickly, with greater consistency and control.

With IBM Cloud Private, developers can get up and running quickly with a lightweight development environment optimized for delivering Docker containerized applications with an integrated DevOps toolchain.

With Cloud Automation Manager, IT infrastructure managers can help provision and maintain cloud infrastructure and traditional VM application environments with a consistent operational experience across multiple clouds.

With the Cloud Automation Manager Service Composer, IT service managers can graphically compose complex cloud services that can be consumed as a service from a DevOps toolchain or delivered in a cloud service catalog.

With a large and growing catalog of pre-built automation content for popular open source and IBM middleware, built to best practices, developers and IT architects can get productive fast.

The main capabilities are:

- ▶ **Accelerate innovation:** it helps to provision into multi-cloud environments through a self-service catalog. it easily integrates with existing processes and tools through automated workflow capabilities.
- ▶ **Multi-cloud operations:** it consistently manage and govern across all of your cloud environments.
- ▶ **Open source technology:** it simplifies your multi-cloud provisioning by utilizing open source and industry standards such as Terraform. you can reuse your existing skills and Chef scripts and avoid vendor lock-in.

- ▶ **Integration with your existing data center:** It allows to use your existing tools (IBM Cloud Brokerage, IBM DevOps, IT management, etc.) to extend IBM Cloud Automation Manager capabilities.
- ▶ **Infrastructure as code:** Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

See the following link for more information on IBM Cloud Automation Manager:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/featured_applications/cam.html

1.4.9 IBM Cloud Transformation Advisor

The Transformation Advisor is an interactive analyzer tool for modernizing monolith Java/J2EE workloads of an enterprise. IBM Transformation Advisor scans the application artefact for its compatibility on IBM Cloud Private and produces a report. It highlights code correction if needed, to make the code compatible to run on IBM Cloud Private. Apart from traditional WebSphere Application Server applications, it also supports other competitor server runtime application for the compatibility to be ported on IBM Cloud Private.

The scan report highlights any gaps and efforts needed to make the application cloud ready for deployment. The result of the Application Binary scan also provides the deployment YAML, docker file for containerizing the application and a liberty server.xml file. If an application is fully compliant and does not require any changes, then it can be directly deployed to an IBM Cloud Private through Transformation Advisor console itself for testing.

Benefits:

- ▶ Included and deployed on IBM Cloud Private
- ▶ Introspects applications running on most popular runtime environments
- ▶ Spits out effort estimates for application modernization for the workload
- ▶ Deploy application to a target IBM Cloud Private environment if the application is fully compliant.
- ▶ Provides recommendations for application modernization

Figure 1-12 on page 16 shows the IBM Cloud Transformation Advisor. See the following link for more information on IBM Cloud Transformation Advisor:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/featured_applications/transformation_advisor.html.

Data collector

Recommendations

Profile

Dmgr01

Preferred migration

Liberty on Private Cloud

Source Environment


IBM WebSphere Application Server Network Deployment

Source Version

9.0.0.4

Q

Search items

Application		Tech match	Dependencies	Issues	Est. dev cost in days	Total effort in days	
> StatelessEJB3.ear Liberty on Private Cloud	Simple	100%	1	2	0	5	Migration plan
> jpetstore_war.ear Liberty on Private Cloud	Simple	100%	2	4	0	5	Migration plan
> HealthServicesClient.ear Liberty on Private Cloud	Moderate	80%	1	2 1	2	7	Migration plan
> HealthServices.ear Liberty on Private Cloud	Moderate	50%	0	1 1	2	7	Migration plan
>  DefaultApplication.ear Liberty on Private Cloud	Complex	88%	4	1 1 3	13	18	Migration plan
> DayTrader-EE6.ear Liberty on Private Cloud	Moderate	100%	3	3 1 1	1	6	Migration plan

Items per page: 10 | 1-6 of 6 items

1 of 1 pages

< 1 >

Figure 1-12 IBM Cloud Transformation Advisor

1.4.10 IBM Microclimate

Microclimate provides an end-to-end, cloud-native solution for creating, building, testing and deploying applications. The solution offers services and tools to help you create and modernize applications in one seamless experience. It covers each step of the process from writing and testing code to building and deployment. The solution enables containerized development, rapid iteration with real-time performance insights, intelligent feedback, diagnostic services, an integrated DevOps pipeline and deployment to the cloud.

You can see the *IBM Cloud Private Application Developer's Guide*, SG24-8441 IBM Redbooks for detailed information on IBM Microclimate installation and configuration and how to use it in a sample scenario.

1.4.11 IBM Cloud Private management console

Manage, monitor, and troubleshoot your applications and cluster from a single, centralized, and secure management console.

Figure 1-13 on page 17 shows the IBM Cloud Private management console.

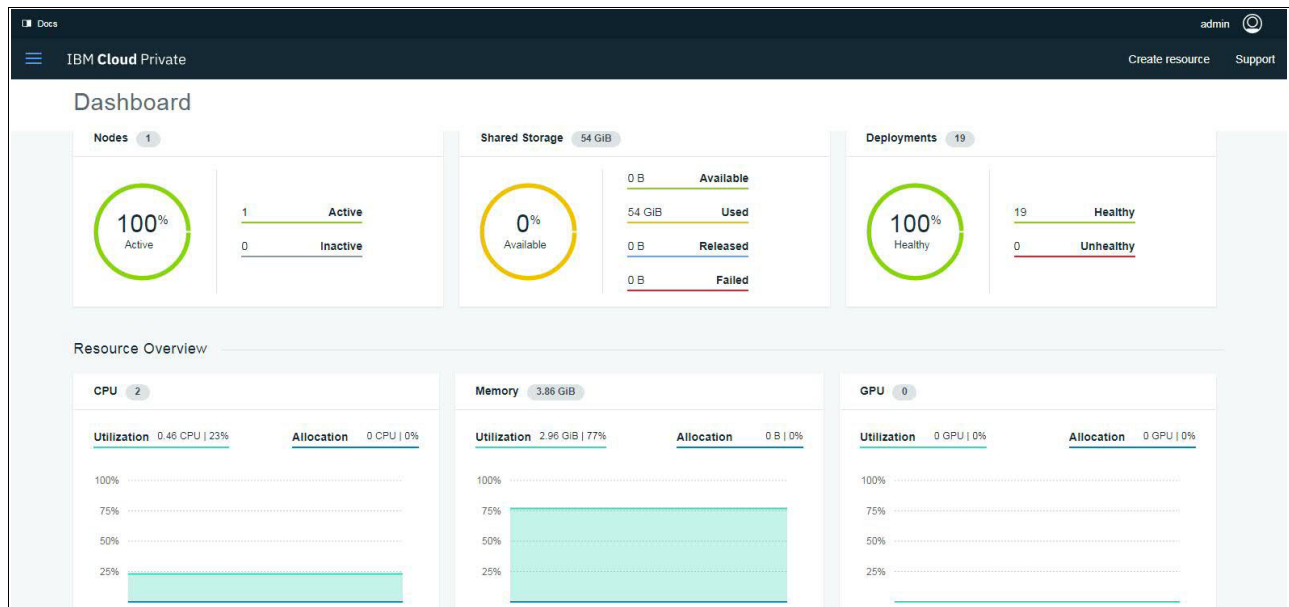


Figure 1-13 IBM Cloud Private management console

1.4.12 Kubernetes

To run a container in production, Kubernetes brings orchestration primitives to support different styles of workloads:

- Stateless ReplicaSets
- Stateful StatefulSets
- Batch Jobs
- System DaemonSets

1.4.13 Private Docker image registry

The Private Docker registry integrates with the Docker registry V2 API to provide a local registry service that functions in the same way as the cloud-based registry service, Docker Hub. This local registry has all the same features as Docker Hub, but you can also restrict which users can view or pull images from this registry.

1.4.14 Helm with enhanced security controls

Helm, the Kubernetes native package management system, is used for application management inside an IBM Cloud Private cluster. The Helm GitHub community curates and continuously expands a set of tested and preconfigured Kubernetes applications. You can add items from this catalog of stable applications to your cluster from the management console. Installing this Helm community catalog provides an extra 80+ Kubernetes applications that are ready for deployment in your cluster.

Helm charts describe even the most complex applications; provide repeatable application installation, and serve as a single point of authority. Helm charts are easy to update with in-place upgrades and custom hooks. Charts are also easy to version, share, and host on public or private servers. You can use helm rollback to roll back to an older version of a release with ease. See 1.5, “Helm” on page 19 for more information on Helm components.

Also you can see the *IBM Cloud Private Application Developer's Guide*, SG24-8441 IBM Redbooks on how Helm is used for application management.

1.4.15 Catalog

IBM Cloud Private provides an easy to use, extend, and compose catalog of IBM and third-party content. The following are some key concepts:

- ▶ **Charts:** A bundle of Kubernetes resources
- ▶ **Repository:** A collection of charts
- ▶ **Releases:** A chart instance loaded into Kubernetes. The same chart can be deployed several times and each becomes its own release

The catalog provides a centralized location from which you can browse for and install packages in your cluster

Packages for additional IBM products are available from curated repositories that are included in the default IBM Cloud Private repository list. Your environment must be connected to the internet for you to access the charts for these packages. To view a list of all the IBM Cloud.

Figure 1-14 shows the IBM Cloud Private catalog.

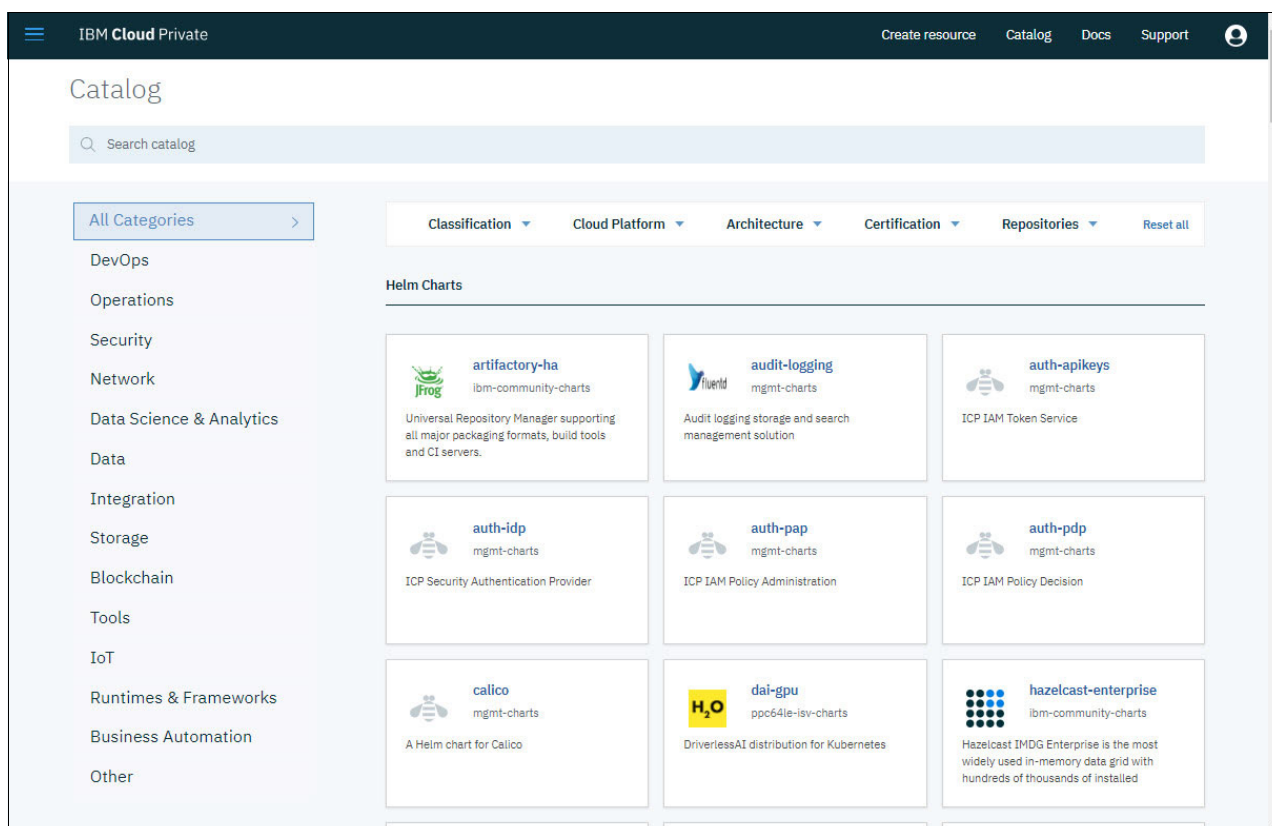


Figure 1-14 IBM Cloud Private catalog

1.4.16 Kubernetes Service Catalog for managing service brokers

IBM Cloud Private supports the Kubernetes Service Catalog. You can configure the service broker applications to manage the Service Catalog resources and details.

The Service Catalog component adds the following Kubernetes resources:

- ▶ ClusterServiceBrokers
- ▶ ClusterServiceClasses
- ▶ ClusterServicePlans
- ▶ ServiceInstances
- ▶ ServiceBindings

The service broker is a component that implements the service broker API to view the available services and plans, create an instance from the available services and plans, and create bindings to connect to the service instance.

1.5 Helm

Helm is a package manager. Package managers automate the process of installing, configuring, upgrading, and removing applications on a Kubernetes cluster.

An application in Kubernetes typically consists of at least two resource types: a deployment resource, which describes a set of pods to be deployed together, and a services resource, which defines endpoints for accessing the APIs in those pods. The application can also include ConfigMaps, Secrets, and Ingress.

For any application deployment, several Kubernetes commands (**kubectl**) are needed to create and configure resources. Instead of manually creating each application dependency resource separately, Helm creates many resources with one command. A Helm chart defines several Kubernetes resources as a set in a *YAML* file. A default chart contains a minimum of a deployment template and a service template.

1.5.1 Helm components and terminology

Helm has two elements. A client (*Helm*) and a server (*Tiller*). The server element runs inside a Kubernetes cluster and manages the installation of the charts. Figure 1-15 on page 20 shows how Helm components are related to each other.

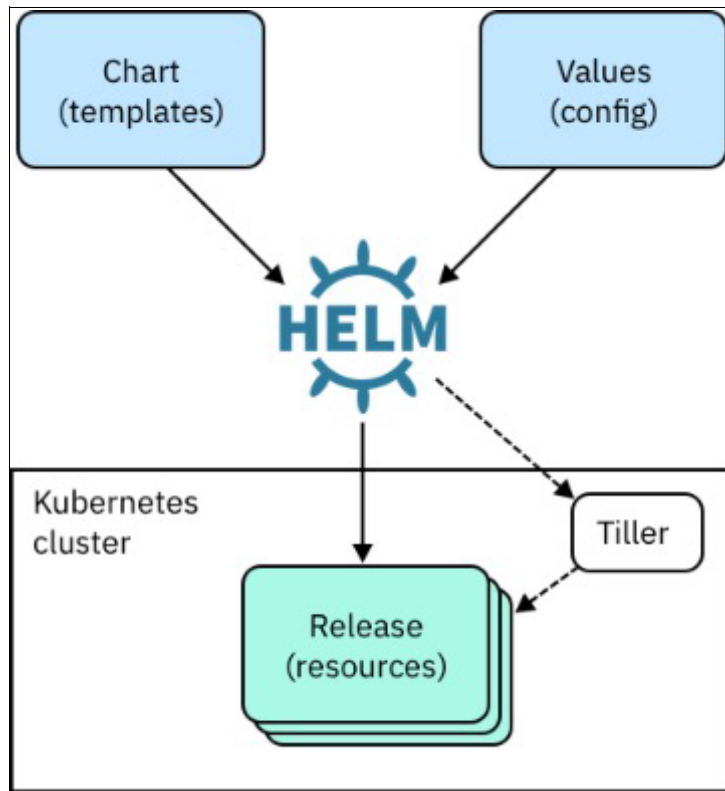


Figure 1-15 Helm components

- ▶ **Helm:** A command-line interface (CLI) that installs charts into Kubernetes, creating a release for each installation. To find new charts, search Helm chart repositories.
- ▶ **Chart:** An application package that contains templates for a set of resources that are necessary to run the application. A template uses variables that are substituted with values when the manifest is created. The chart includes a values file that describes how to configure the resources.
- ▶ **Repository:** Storage for Helm charts. The namespace of the hub for official charts is stable.
- ▶ **Release:** An instance of a chart that is running in a Kubernetes cluster. You can install the same chart multiple times to create many releases.
- ▶ **Tiller:** The Helm server-side templating engine, which runs in a pod in a Kubernetes cluster. Tiller processes a chart to generate Kubernetes resource manifests, which are YAML-formatted files that describe a resource. YAML is a human-readable structured data format. Tiller then installs the release into the cluster. Tiller stores each release as a Kubernetes ConfigMap.

1.5.2 Why you should use Helm

Helm charts describe even the most complex applications, provide repeatable application installation and are easy to share, version and publish.

With Helm, configuration settings are kept separate from the manifest formats. You can edit the configuration values without changing the rest of the manifest. Configuration settings are in a `values.yaml` file. You update the runtime parameters in that file to deploy each application instance differently. A single command is used to install, upgrade and deleting releases as represented below in Figure 1-16 for lifecycle of a release.

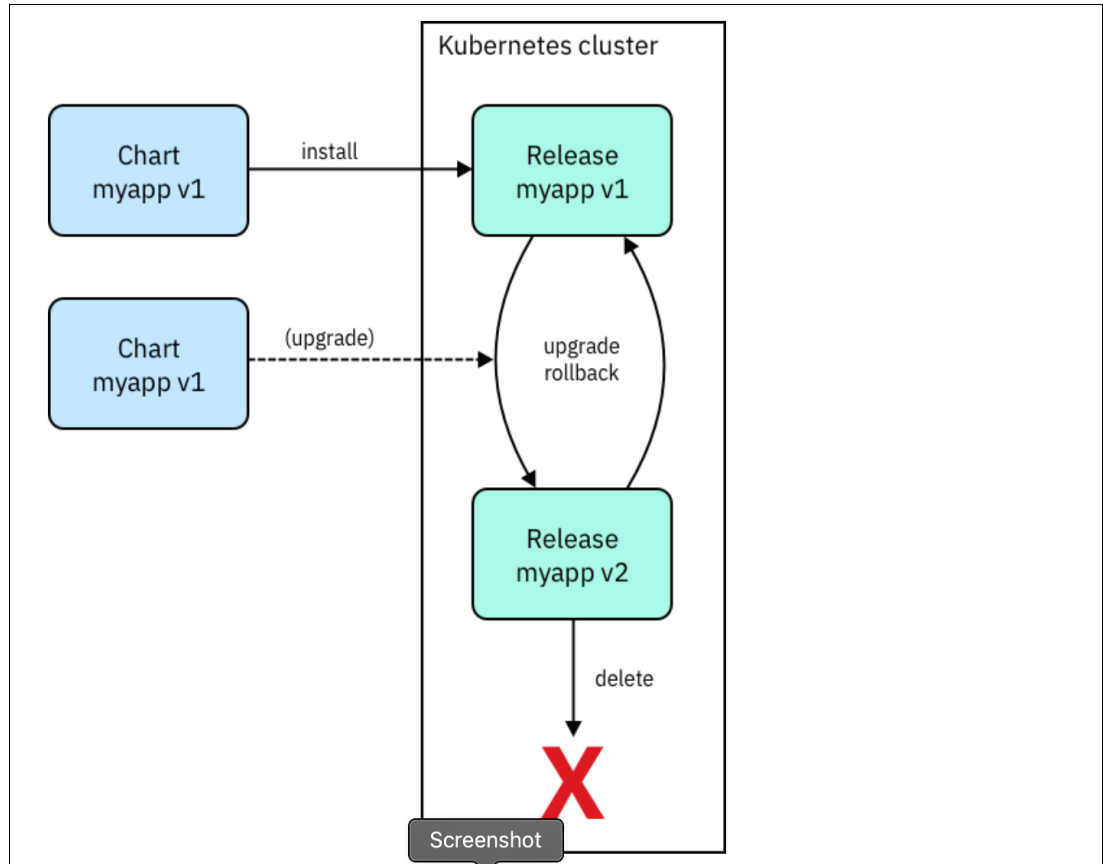


Figure 1-16 Release lifecycle using Helm

1.6 IBM Multicloud Manager

IBM Multicloud Manager is the single dashboard that lets your enterprise oversee multiple Kubernetes clusters wherever they are—public cloud or private.

Working with more than one or two clouds means you can pick the best providers and services across clouds, geographies and functions for specific needs. This helps potentially lower costs, increase performance and solidify governance.

Multicloud enterprises rely on private clouds for better data center performance and availability. They depend on public clouds to be more competitive, get to market faster and build new capabilities like analytics and artificial intelligence.

But successfully developing and deploying these new capabilities means that the average multicloud enterprise uses six or more clouds and hundreds of Kubernetes clusters. This creates a complex, error-prone environment that's expensive and time-consuming to manage. IBM Multicloud Manager supports enterprises to manage resources across clouds and helping decreasing compliance risks and reduce cost.

1.7 IBM Cloud Paks

One key technology contributing to the ability to run software components more efficiently, with improved lifecycle management, scalability and resilience, is known as containerization. IBM delivers Kubernetes as part of its IBM Cloud Private offering as well as in its IBM Cloud Kubernetes Service. IBM is committed to containerization across its enterprise software portfolio.

While still supporting traditional packaging and deployment models (installing software as usual on a supported operating system), an increasing number of products are available as container images. As mentioned, deploying containerized software in a manner suitable for production requires more than an image.

IBM Cloud Paks provide enterprise software container images that are pre-packaged in production-ready configurations that can be quickly and easily deployed to IBM's container platforms, with support for resiliency, scalability, and integration with core platform services, like monitoring or identity management. For customers who don't want to operate the software and its underlying infrastructure, in containers or otherwise, IBM also makes many of its products available as-a-Service, where they are hosted and maintained by IBM in its public cloud.

Figure 1-17 shows the three ways IBM software is delivered and consumed as containers. Not all IBM products are available in all three delivery models.




Ad hoc Containers	IBM Containers	IBM Cloud Paks
		
Client obtains IBM software binaries Creates their own containers	Client obtains IBM Software container, from Docker Hub/Store or Passport Advantage	Client obtains a complete package Includes the Container, Automated Deployment, Best Practice configuration, Built-in Management and Full stack support on IBM platforms

Figure 1-17 IBM software as containers

IBM Cloud Paks use Helm charts to describe how IBM software should be deployed in a Kubernetes environment. These resource definitions can be easily customized during deployment, and upgrades can be easily rolled out or rolled back using the management interfaces provided by IBM Cloud Private or IBM Cloud Kubernetes Service.

An IBM Cloud Pak is more than a simple Helm chart. IBM Cloud Paks accelerate time to value and improve enterprise readiness at lower cost than containers alone.

IBM Cloud Pak is a collection of assets for container native applications, many based on open technologies. IBM Cloud Pak delivers higher value than containers alone. Certified IBM Cloud Pak are enterprise ready out of the box.

IBM Cloud Paks are identified in the Catalog by one of two badges with the entry. An entry with an *IBM Cloud Pak* badge meets the criteria for that badge. An entry that displays a *Certified IBM Cloud Pak* badge indicates that it meets the requirements of the *Certified IBM Cloud Pak* badge, which are more stringent than what is required for the *IBM Cloud Pak* badge.

IBM Cloud Paks can be created by IBM or 3rd party software solutions that are offered by IBM Partners. Figure 1-18 shows the comparison between IBM Cloud Pak and Certified IBM Cloud Pak.

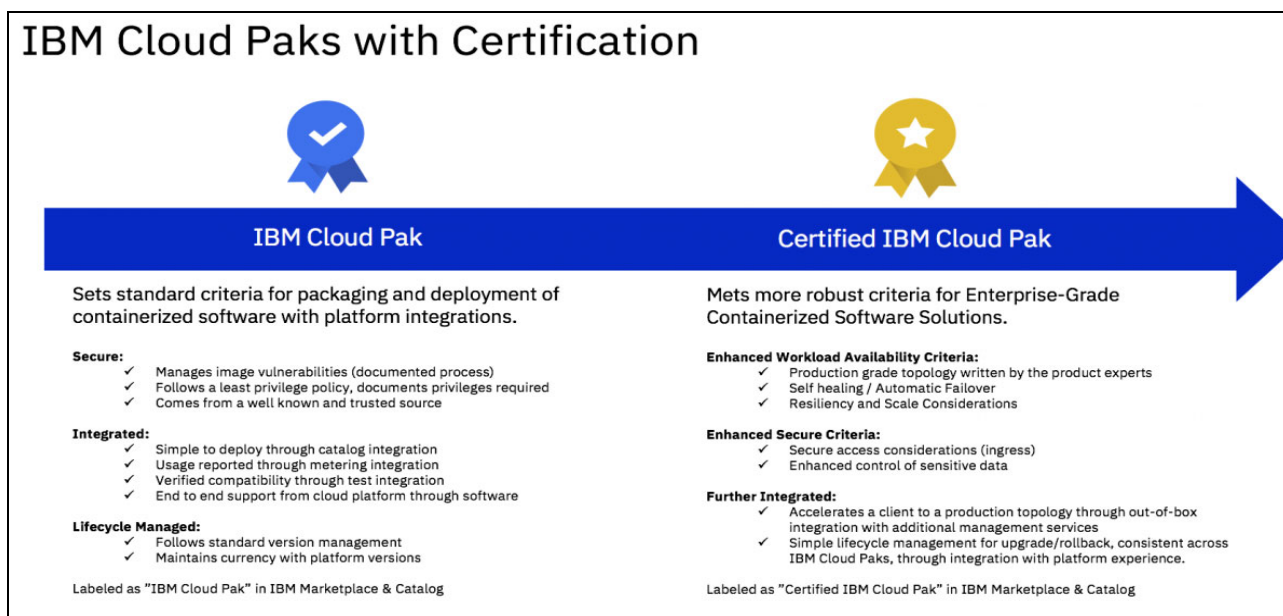


Figure 1-18 Comparison between IBM Cloud Pak and Certified IBM Cloud Pak

For more information visit the IBM Knowledge Center link https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/app_center/cloud_paks_over.html.

1.8 IBM Cloud Private Editions

The following are the IBM Cloud Private Editions:

- ▶ *IBM Cloud Private* delivers a customer-managed container solution for enterprises. It is also in a community edition, IBM Cloud Private-Community Edition, which provides a limited offering that is available at no charge and is ideal for test environments.
- ▶ *IBM Cloud Private Cloud Native Edition* is licensed and provides an high availability topology for an enterprise production runtime. Cloud Automation Manager, Vulnerability Advisor is also packaged as part of Cloud Native edition.
- ▶ *IBM Cloud Private Enterprise Edition* has all the offerings of IBM Cloud Private Native Edition plus IBM WebSphere Application Server Network Deployment MQ Advanced, APICConnect.

Figure 1-19 shows the IBM Cloud Private Editions.

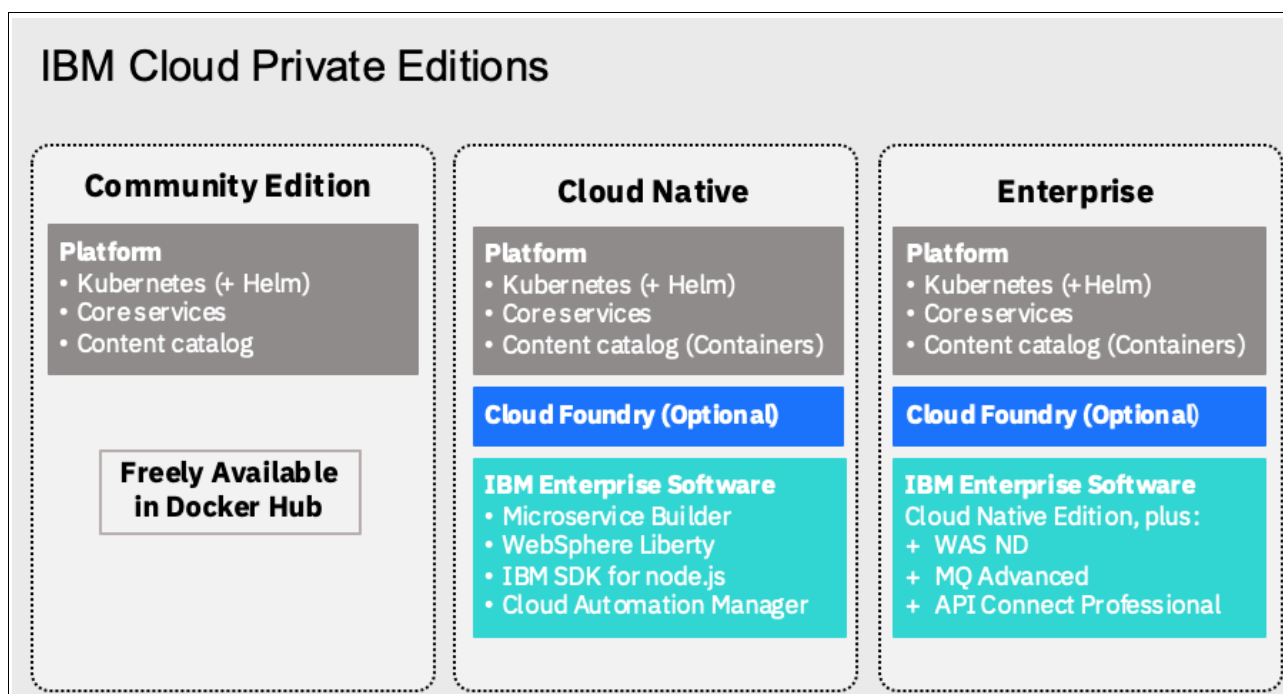


Figure 1-19 IBM Cloud Private Editions

IBM Cloud Private for Data: In addition to these IBM Cloud Private Editions, there is another related product called IBM Cloud Private for Data. This product is a native cloud solution that enables you to put your data to work quickly and efficiently. It lets you do both by enabling you to connect to your data (no matter where it lives), govern it, find it, and use it for analysis. IBM Cloud Private for Data also enables all of your data users to collaborate from a single, unified interface, so your IT department doesn't need to deploy and connect multiple applications.

We will not discuss this product in this document. You can see the product page at <https://www.ibm.com/analytics/cloud-private-for-data> for more information.

1.9 Persistent volumes

Persistent volume is a storage resource within the cluster. Persistent volumes have a lifecycle independent of any individual pod that uses it. This API object encapsulates the details of the storage implementation or cloud-provider-specific storage system, as shown in Figure 1-20 on page 25.

A persistent volume claim is a storage request, or claim, made by the developer. Claims request specific sizes of storage, as well as other aspects such as access modes.

A StorageClass describes an offering of storage and allow for the dynamically provisioning of PVs and PVCs based upon these controlled definitions.

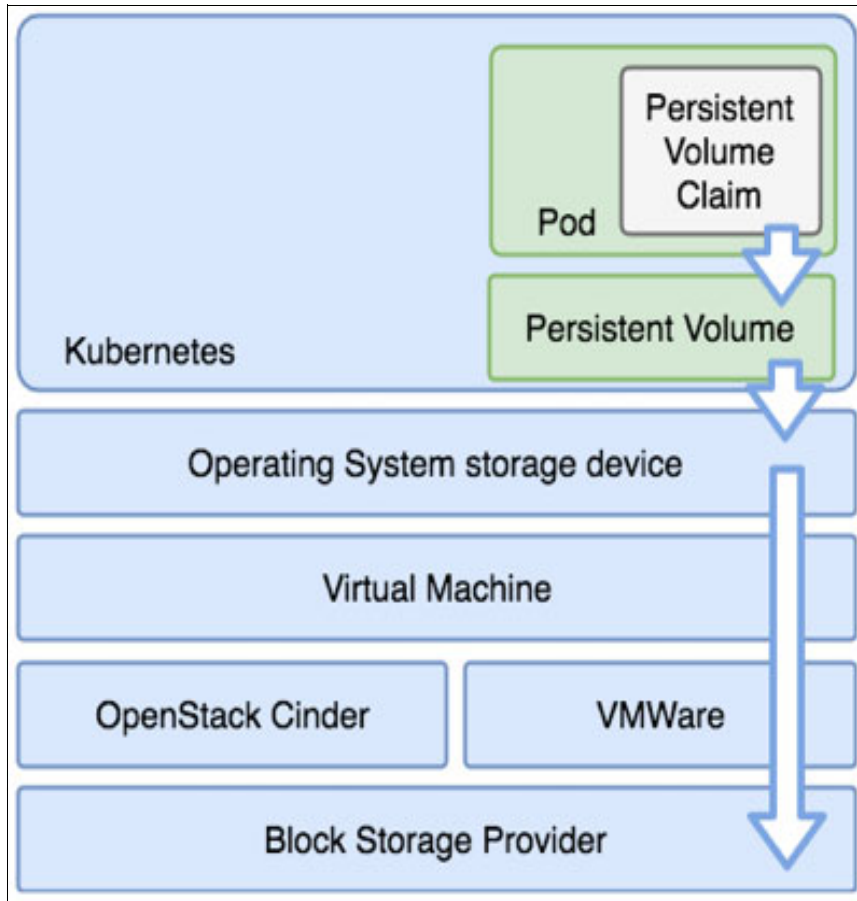


Figure 1-20 Persistent volumes

1.9.1 Volume and claim lifecycle

The Reclaim policy informs the cluster what to do with the volume after it has been released from its claim

Retain allows for the manual reclamation of the storage asset. The PVC is deleted but the PV remains.

Delete reclaim policy removes both the objects from within the cluster, as well as the associated storage asset from the external infrastructure.

Recycle has been deprecated and should no longer be used.

Access Modes define how volumes can be mounted in the manner supported by the storage provider

- ▶ ReadWriteOnce (RWO) can be mounted as read-write by a single node and pod
- ▶ ReadOnlyMany (ROX) can be mounted read-only by many nodes and pods
- ▶ ReadWriteMany (RWX) can be mounted as read-write by many nodes and pods

Note: Reclaim policy and Access Modes may be defined differently by each storage provider implementation.

1.9.2 IBM Cloud Private Storage providers

Kubernetes and IBM Cloud Private offer many options for managing persistent storage within the cluster. IBM Cloud Private features the following:

- ▶ GlusterFS enterprise grade of storage to K8s pods offering 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 not hosted in VMware provides direct access to IBM block storage via 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.
- ▶ Ceph (Rook) is an industry proven option that can provide several storage options along with persistent volumes for Kubernetes.

See Chapter 4, “Managing persistence in IBM Cloud Private” on page 115 for more information on managing storage in IBM Cloud Private.

1.10 IBM Cloud Private terms

The following IBM Cloud Private terms are used throughout this book. Table 1-1 shows the definition of these terms.

Table 1-1 IBM Cloud Private terms

IBM Cloud Private term	Definition
airgap environment	A network environment that does not have internet access.
API key	A unique code that is passed to an API to identify the calling application or user. An API key is used to track and control how the API is being used, for example, to prevent malicious use or abuse of the API.
application log (applog)	A log that is produced from applications that are deployed in the Cloud Foundry environment.
application	One or more computer programs or software components that provide a function in direct support of a specific business process or processes.
audit log	A log file containing a record of system events and responses.
Availability Zone	An operator-assigned, functionally independent segment of network infrastructure.
boot node	A node that is used for running installation, configuration, node scaling, and cluster updates.

IBM Cloud Private term	Definition
buildpack	A collection of scripts that provide framework and runtime support for apps.
catalog	A centralized location that can be used to browse for and install packages in a cluster.
Cloud Foundry deployment tool	The user interface that is used to manage the deployment of Cloud Foundry.
cluster	A set of resources, worker nodes, networks, and storage devices that keep apps highly available and ready to deploy in containers.
container image	In Docker, standalone, executable software, including code and system tools, that can be used to run an application.
container	A system construct that allows users to simultaneously run separate logical operating system instances. Containers use layers of file systems to minimize image sizes and promote reuse.
deployment	A process that retrieves the output of a build, packages the output with configuration properties, and installs the package in a pre-defined location so that it can be tested or run.
Docker	An open platform that developers and system administrators can use to build, ship, and run distributed applications.
ELK stack	The three products, Elasticsearch, Logstash, and Kibana, that comprise a stack of tools that stream, store, search, and monitor data, including logs.
endpoint	A network destination address that is exposed by Kubernetes resources, such as services and ingresses.
extension	A package that contains a deployment process and its required scripts and files.
fault tolerance	The ability of a system to continue to operate effectively after the failure of a component part.
Helm chart	A Helm package that contains information for installing a set of Kubernetes resources into a Kubernetes cluster.
Helm release	An instance of a Helm chart that runs in a Kubernetes cluster.
Helm repository	A collection of charts.

IBM Cloud Private term	Definition
high availability	The ability of IT services to withstand all outages and continue providing processing capability according to some predefined service level. Covered outages include both planned events, such as maintenance and backups, and unplanned events, such as software failures, hardware failures, power failures, and disasters.
image manager	A centralized location for managing images inside a cluster.
image	A file system and its execution parameters that are used within a container runtime to create a container. The file system consists of a series of layers, combined at runtime, that are created as the image is built by successive updates. The image does not retain state as the container executes.
inception container	See “installer container.”
ingress	A collection of rules to allow inbound connections to the Kubernetes cluster services.
installer container	The Docker container that runs the configuration manager and the Cloud Foundry deployment tool.
isolation segment	A division that can be used to separate applications as if they were in different deployments without the need for redundant management and network complexity.
isolation	The process of confining workload deployments to dedicated virtual and physical resources to achieve multi-tenancy support.
Kubernetes	An open-source orchestration tool for containers.
layer	A changed version of a parent image. Images consist of layers, where the changed version is layered on top of the parent image to create the new image.
load balancer	Software or hardware that distributes workload across a set of servers to ensure that servers are not overloaded. The load balancer also directs users to another server if the initial server fails.
machine type	A configuration that is used to instantiate a virtual machine.
management console	The graphical user interface for IBM Cloud Private.
management logging service	An ELK stack that is used to collect and store all Docker-captured logs.

IBM Cloud Private term	Definition
management node	An optional node that only hosts management services such as monitoring, metering, and logging and can be used to prevent the master node from becoming overloaded.
marketplace	A list of enabled services from which users can provision resources.
master node	A node that provides management services and controls the worker nodes in a cluster. Master nodes host processes that are responsible for resource allocation, state maintenance, scheduling, and monitoring.
mesh	A network topology in which devices are connected with many redundant interconnections between network nodes. Every node has a connection to every other node in the network.
microclimate	An end-to-end, cloud-native solution for creating, building, testing, and deploying applications.
microservice	A set of small, independent architectural components, each with a single purpose, that communicate over a common lightweight API.
multicloud	A cloud computing model in which an enterprise uses a combination of on-premises, private cloud, and public cloud architecture.
Network File System	A protocol that allows a computer to access files over a network as if they were on its local disks.
organization (org)	In Cloud Foundry, the top-most meta object within the infrastructure that is managed by an account with administrative privileges.
persistent volume claim	A request for cluster storage.
persistent volume	Networked storage in a cluster that is provisioned by an administrator.
placement policy	A policy that defines where the application components should be deployed and how many replicas there should be.
Pod Security Policy	A policy that is used to set up cluster-level control over what a pod can do or what it can access.
pod	A group of containers that are running on a Kubernetes cluster. A pod is a runnable unit of work, which can be either a stand-alone application or a microservice.
proxy node	A node that transmits external requests to the services that are created inside a cluster.
registry	A public or private container image storage and distribution service.
repository	A persistent storage area for data and other application resources.

IBM Cloud Private term	Definition
resource	A physical or logical component that can be provisioned or reserved for an application or service instance. Examples of resources include database, accounts, and processor, memory, and storage limits.
role-based access control	The process of restricting integral components of a system based on user authentication, roles, and permissions.
service broker	A component of a service that implements a catalog of offerings and service plans, and interprets calls for provisioning and deprovisioning, binding and unbinding.
storage node	A node that is used to provide the backend storage and file system to store the data in a system.
system log (syslog)	A log that is produced by Cloud Foundry components.
taint	To mark a particular input, such as a variable, as being unsafe in order to subject it to security checking.
team	An entity that groups users and resources.
vCPU	A virtual core that is assigned to a virtual machine or a physical processor core if the server isn't partitioned for virtual machines.
Virtual Machine File System (VMFS)	A cluster file system that allows virtualization to scale beyond a single node for multiple VMware ESX servers.
virtual storage area network (VSAN)	A fabric within the storage area network (SAN).
worker node	In a cluster, a physical or virtual machine that carries the deployments and services that make up an app.
workload	A collection of virtual servers that perform a customer-defined collective purpose. A workload generally can be viewed as a multitiered application. Each workload is associated with a set of policies that define performance and energy consumption goals.



High availability installation

This chapter provides a step by step guide on how to install IBM Cloud Private in a high availability (HA) configuration. It also discusses the recommended HA configuration options.

This chapter covers the following topics:

- ▶ 2.1, “High availability considerations” on page 32
- ▶ 2.2, “High Availability models for IBM Cloud Private cluster” on page 35
- ▶ 2.3, “Performance considerations for IBM Cloud Private setup” on page 38
- ▶ 2.4, “Step-by-step installation guide using Terraform” on page 40
- ▶ 2.5, “Post installation verification” on page 53
- ▶ 2.6, “Installing IBM Cloud Private on other Cloud platforms” on page 63
- ▶ 2.7, “Setting up IBM Cloud Private catalog in an airgap environment” on page 65
- ▶ 2.8, “Changing certificates post installation” on page 67

2.1 High availability considerations

In a container-as-a-Service (CaaS) platform such as IBM Cloud Private, two levels of high availability exists for resiliency:

- ▶ High availability for platform components.
- ▶ High availability of the workloads and applications hosted on this platform.

High availability installations of IBM Cloud Private platform are only supported through Cloud Native and Enterprise Edition only. See section 1.3, “IBM Cloud Private architecture” on page 10 for details on the IBM Cloud Private architecture and section 1.2, “IBM Cloud Private node types” on page 6 for a discussion of the IBM Cloud Private node types.

System administrators should determine the high availability requirements of the IBM Cloud Private platform installation before the software is installed.

Important: It is not possible to convert a standard installation into a high availability installation, or add new master nodes to a highly available installation.

Kubernetes technology provides built-in functions to support the resiliency of a cluster. When administrators install IBM Cloud Private, the installation process installs all the components that they need. However, it is always good to know how Kubernetes works. Administrators can deploy the master nodes and the proxy nodes in plurality for achieving high availability. System administrators can configure high availability for only the master nodes, only the proxy nodes, or for both types of nodes.

Note: Every instance of the master node has its own etcd database and runs the API Server. The etcd database contains vital data of the cluster used in orchestration. Data in etcd is replicated across multiple master nodes. This applies incase the etcd is included within the masters, you can also separate etcd nodes from the master nodes.

Note: Scheduler and Control managers of Kubernetes are in all the master nodes, but they are active only in one master node. They work in *leader election mode* so that only one is active. If a failure occurs, another instance of master node takes over.

2.1.1 Fault tolerance

System administrators must have an odd number of masters in the IBM Cloud Private cluster. Having an odd master size does not change the numbers needed for majority. Majority is the number of master nodes needed for the cluster to be able to operate. However, adding extra master nodes provide a higher tolerance for failure. For N number of masters in a cluster, the cluster can tolerate up to $(N-1)/2$ permanent failures. For example, in a cluster that has three masters, if one master fails, then the fault tolerance is as $(3-1)/2=1$. System administrators must aim for a fault tolerance of one or more.

Table 2-1 shows how fault tolerance in a cluster is affected by even and odd sized clusters.

Table 2-1 Fault tolerance for the HA clusters

Cluster Size	Majority	Fault Tolerance
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

2.1.2 Considerations for sizing the IBM Cloud Private cluster

System administrators should consider following points while sizing the cluster.

Worker node considerations

Consider the following when planning your worker nodes:

- ▶ If your cluster has a few worker nodes, consider increasing the number of worker nodes while decreasing the size of the nodes for adequate headspace, efficiency, mobility, and resiliency.
- ▶ Accommodate the workload mobility.
- ▶ Consider the memory that is required for a specific type of workload.
- ▶ Consider the memory that is required for other application frameworks.
- ▶ The maximum pod per node is 500 and the maximum pod per CPU core is 10 (This is for Intel based workload).
- ▶ The cluster size depends on the worker node number. The pod number depends on the application type and the worker node's configuration.

Proxy node considerations

The following needs to be considered when planning your proxy nodes:

- ▶ If your cluster has a few worker nodes, consider increasing the number of worker nodes while decreasing the size of the nodes for adequate headspace, efficiency, mobility, and resiliency.
- ▶ Accommodate the workload mobility.
- ▶ Consider the memory that is required for a specific type of workload.
- ▶ Consider the memory that is required for other application frameworks.
- ▶ The maximum pod per node is 500 and the maximum pod per CPU core is 10.
- ▶ The cluster size depends on the worker node number. The pod number depends on the application type and the worker node's configuration.

Management node considerations

Larger clusters with more workload require larger management nodes. Management nodes can be added at any time if they were originally externalized. For proxy nodes, consider the headspace requirements to carry the workload due to a node failure.

Large cluster considerations

The following are the large cluster considerations:

- ▶ The node-to-node mesh starts to fail with 700 nodes in the cluster. You must create a router reflector for BGP (Border Gateway Protocol) daemons.
- ▶ Consider to use etcd outside of your master nodes if you plan on having a cluster with several hundred worker nodes. A separated etcd cluster is ideal to reduce the impact on the master node.
- ▶ Be sure to implement load balancing on your master node.
- ▶ The number of services in your cluster affects the load on each node. In large clusters with more than 5000 services, you must run your nodes in IP Virtual Server (IPVS) mode.

2.1.3 Sample sizing for your IBM Cloud Private cluster

This section gives some sample configurations for sizing your IBM Cloud Private cluster.

Small size environment with medium resiliency

Table 2-2 shows sizing of small sized IBM Cloud Private cluster environment.

Section 2.4, “Step-by-step installation guide using Terraform” shows installation of IBM Cloud Private cluster with number of nodes of each type, shown in Table 2-2 with an additional vulnerability advisor node.

Table 2-2 Small size environment with medium resiliency

Node Type	Number of Nodes	CPU	Memory (Gb)	Disk (Gb)
Boot	1	2	8	250
Master	3	16	32	500
Management	2	8	16	500
Proxy	2	4	16	400
Worker	3+	8	32	400

Medium size environment with medium resiliency

Table 2-3 below shows sizing of medium sized IBM Cloud Private cluster environment.

Table 2-3 Medium size environment with medium resiliency

Node Type	Number of Nodes	CPU	Memory (Gb)	Disk (Gb)
Boot	1	2	8	250
Master	3	16	32	500
Management	2	16	32	500

Node Type	Number of Nodes	CPU	Memory (Gb)	Disk (Gb)
Proxy	3	4	16	400
Worker	3+	8	32	400
Vulnerability advisor	3	6	24	500

Large size environment (worker nodes < 500) with high resiliency

Table 2-4 below shows sizing for large sized IBM Cloud Private cluster environment with 500 or less worker nodes.

Table 2-4 Large size environment with 500 worker nodes

Node Type	Number of Nodes	CPU	Memory (Gb)	Disk (Gb)
Boot	1	4	8	250
Master	3	16	128	500
Management	2	16	128	500
Proxy	2	4	16	400
Worker	500	8	32	400
Vulnerability advisor	3	6	48	500

Large size environment (worker nodes < 1000) with high resiliency

Table 2-5 shows sizing for large sized IBM Cloud Private cluster environment with 1000 or less worker nodes.

Table 2-5 Large size environment with 1000 worker nodes

Node Type	Number of Nodes	CPU	Memory (Gb)	Disk (Gb)
Boot	1	4	8	250
Master	3	16	128	500
Management	5	16	128	500
Proxy	2	4	16	400
Worker	1000	8	32	400
Vulnerability advisor	3	6	48	500

2.2 High Availability models for IBM Cloud Private cluster

Listed below are three classification models for highly available IBM Cloud Private cluster:

- ▶ Intra Cluster
- ▶ Intra Cluster with multiple availability zones
- ▶ Inter Cluster with federation on different availability zones

2.2.1 Intra cluster

A cluster is composed of at least master nodes and worker nodes. This model consists of HA inside an IBM Cloud Private cluster. The redundancy is implemented by deploying multiple nodes for master and for workers that are distributed in a single site.

This scenario uses Kubernetes functions but can't protect applications from site failure. If system administrators need site-based protection, they should combine this model with other protection solutions, including a disaster recovery solution. Figure 2-1 shows the IBM Cloud Private intra cluster topology.

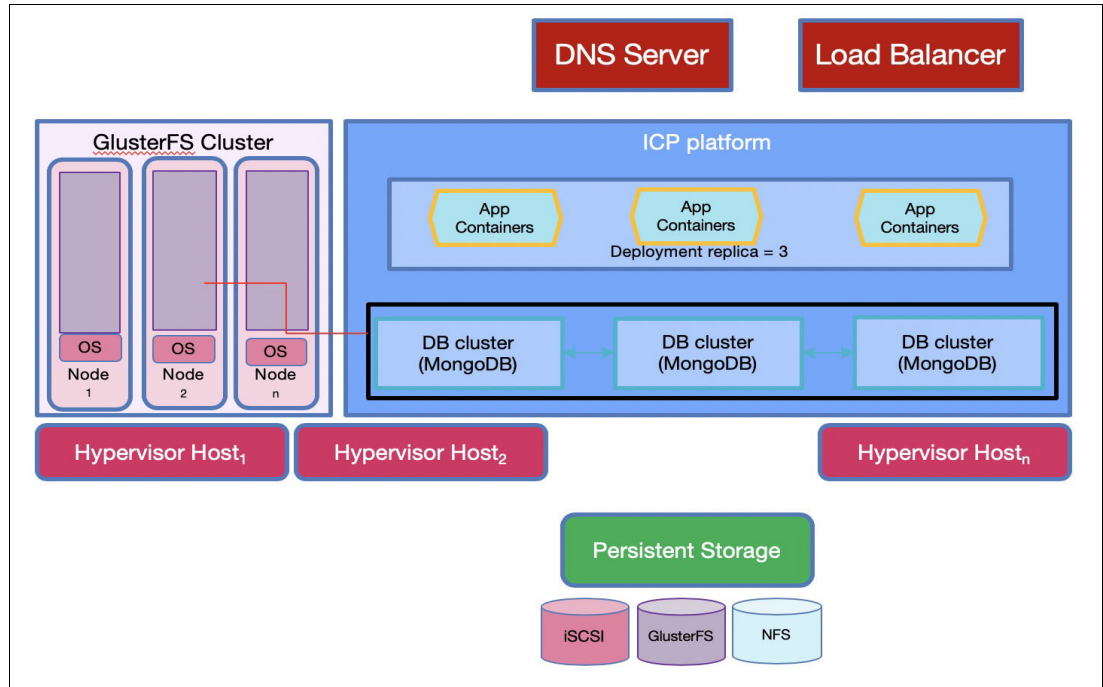


Figure 2-1 Intra cluster topology

2.2.2 Intra cluster with multiple availability zones

This kind of scenario is often referred to as *business continuity*. It combines intra cluster HA with the capability to protect from a site failure.

The cluster is distributed among multiple zones. For example, you might have three, or five, or seven master nodes and several worker nodes distributed among three zones. Zones are sites on the same campus or sites that are close to each other. If a complete zone fails, the master still survives and can move the pods across the remaining worker nodes.

Note: This scenario is possible, but might present a few challenges. It must be implemented carefully.

Potential challenges in this form of workload deployment

- The spread across multiple zones must not introduce latency. A high latency can compromise the overall Kubernetes work with unpredictable results. For example, because of latency, the master might consider a group of workers as unreachable and start to uselessly move pods. Or, one of the master nodes might be considered down only because of a long latency.

- In any failure condition, make sure that most of the master nodes survive. Distributing the cluster in only two zones is almost useless. However, configuring three zones implies more costs and complexity.

Figure 2-2 shows the intra cluster with multiple availability zone topology.

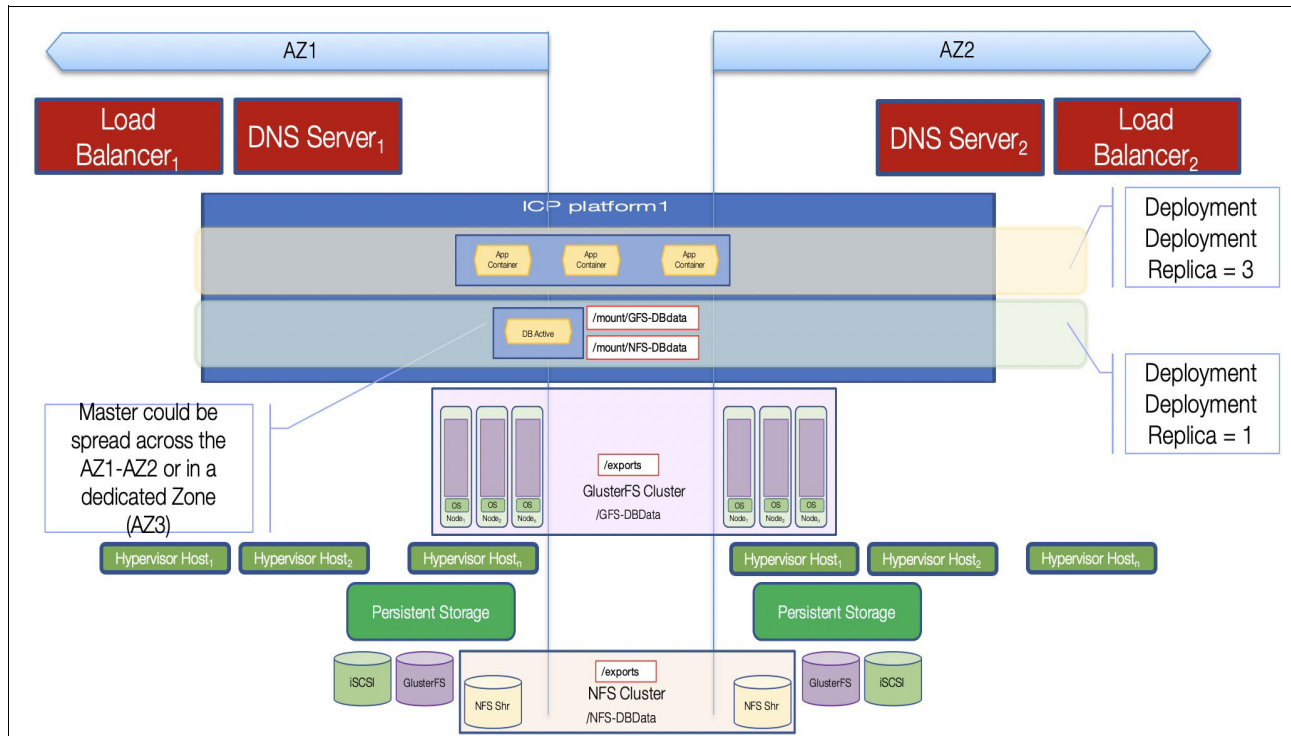


Figure 2-2 Intra cluster with multiple availability zone topology

2.2.3 Inter Cluster with federation on different availability zones

For this model, think of two Kubernetes clusters, each with its own master and worker nodes. If one cluster fails, its pods are run on the other cluster. Kubernetes supports this model by implementing a cluster federation. A higher-level master is deployed as a federated control plane. If a cluster fails, the master control plane instructs the masters of the surviving cluster to redeploy the failed pods.

The federation model is possible; however, beyond the orchestrator, you must consider all the other components of IBM Cloud Private to recover. For example, you must recover all the tools to manage the logs and to monitor your platform and workloads.

While the federation for Kubernetes is a built-in feature, you still must take care of all the other components. As in the “Intra cluster with multiple zones” model, you must also be aware of possible latency problems.

Support for federation is relatively new in Kubernetes. Before you apply federation to business-critical workloads, look at its evolution and maturity.

Figure 2-3 shows the inter cluster with federation on different availability zones topology.

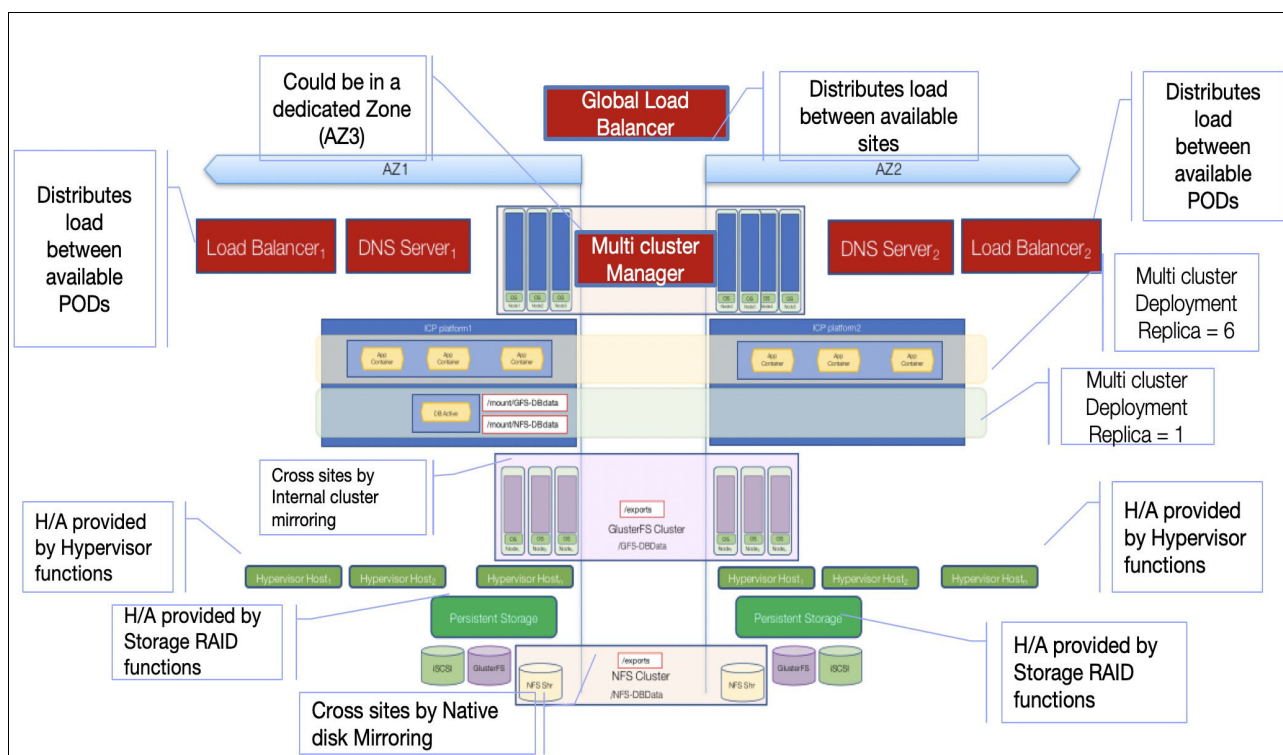


Figure 2-3 Inter cluster with federation on different availability zones topology

2.3 Performance considerations for IBM Cloud Private setup

This chapter covers some of best practises to consider in an IBM Cloud Private cluster setup. These are baselines and the optimal varies based on specific environment needs.

2.3.1 Nodes considerations

This section gives summary of the nodes considerations that we have discussed in the previous section.

- Boot node considerations:
 - Externalizing your boot node alleviates some resource constraints during cluster installation.
- Master node considerations:
 - At least 3 nodes ensures a quorum can be reached upon failure.
 - Additional nodes help ensure the ability to recovery from failure.
 - Consider minimum recommendations to be applicable to only non-production environments.
 - Can Load Balance masters, but only one is the primary master.
 - Only one master node is active at any point in time:

- ▶ Management node considerations:
 - Larger clusters with more workload thus require larger management nodes.
 - Fewer large nodes will have the same impact as many small nodes.
 - Your production clusters should use independent management nodes.
- ▶ Proxy node considerations:
 - For compute sizing consider total resource sizing versus the # of nodes.
 - You can tune your ingress controller to your workload.
 - Your proxy VIP will point only to a single node at a time.
 - Consider optionally load balancer to spread workload to your proxy nodes.
 - Understand your workload to tune and size appropriately.
- ▶ Worker node considerations:
 - If your cluster has a small amount of workload consider increasing the number of worker nodes while decreasing the size of the nodes (for adequate headspace, efficiency, mobility, resiliency).
 - If you have large pods your worker nodes should be larger to accommodate workload mobility.
 - Java workloads typically use 4 x CPU for memory.
 - Other application frameworks may be closer to 2 x CPU = memory.
 - Consider your workload to size your worker nodes.

2.3.2 Tuning the IBM Cloud Private setup

The first thing with tuning is to look at the overhead of logging when talking about high transaction rates. It all depends on your environment, but especially in some network intensive loads this can accumulate an additional overhead that you should take into account. The overhead is mostly attributable to the ELK stack activity.

Impact of the proxy size and upstream keep-alive

Proxy node capacity is vital, ensure yours is sufficient. For network intensive workloads, bigger proxy node can be more efficient than multiple smaller proxy nodes. Default keep-alive is 64: consider tuning this based upon your workload. For more information, see the following IBM Knowledge Center link

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.0/installing/proxy_resource.html.

IBM Cloud Private application logging

Configure log rotate for Docker (max-size, max-file). Without log rotate disk may fill and trigger pod eviction and image garbage collection as shown in Figure 2-4.

```
## The maximum size of the log before it is rolled
docker_log_max_size: 50m
## The maximum number of log files that can be present
docker_log_max_file: 10
```

Figure 2-4 Docker log rotation

If Docker is installed independently, update the Docker services with `--log-opt max-size=10m` `--log-opt max-file=10`. Add the config to `/etc/docker/daemon.json` as shown in Figure 2-5.

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m"
    "max-file": "10"
  }
}
```

Figure 2-5 `/etc/docker/daemon.json`

In summary:

- ▶ Consider your workload volumes. Do you have a high number of transactions? Is the workload variable?
- ▶ Consider your workload personality. Are your services network intensive? What are the programming frameworks you are using?
- ▶ Consider your performance requirements. How do your applications scale: horizontal, vertical, or sharding?

2.4 Step-by-step installation guide using Terraform

In this section, you learn how to install IBM Cloud Private step-by-step using Terraform. Terraform is a tool that allows provisioning and automating the infrastructure using scripts. The infrastructure can include the low level infrastructure as well as the software components on top of it.

In case of IBM Cloud Private, you use Terraform to perform the following automation:

- ▶ Create Security Groups to allow intercluster communication, allow specific port range for the public communication to the load-balancer, and to allow ssh between the boot node and all nodes for installation.
- ▶ Create two local load balancers in-front of the proxy nodes and the master nodes.
- ▶ Create virtual machines for boot node, master nodes, proxy nodes, management nodes, vulnerability advisor, and worker nodes.
- ▶ Create file storage for master nodes shared storage.
- ▶ Deploy IBM Cloud Private 3.1.2 Enterprise Edition on top of the provisioned infrastructure.

Figure 2-6 on page 41 shows an architecture diagram for the IBM Cloud Private highly available configuration. You learn how to deploy IBM Cloud Private based on the recommendations described in 2.1, “High availability considerations” on page 32. In this example, you deploy IBM Cloud Private on top of IBM Cloud infrastructure using the following configuration:

- ▶ 3 master nodes
- ▶ 2 management nodes
- ▶ 2 proxy nodes
- ▶ 1 vulnerability advisor node
- ▶ 3 worker nodes

- ▶ Endurance NFS storage for the directories that should be shared across all master nodes
- ▶ Local load balancer in front of the proxy node to allow public to the applications
- ▶ Local load balancer in front of the master nodes to allow access by admins to the IBM Cloud Private cluster.

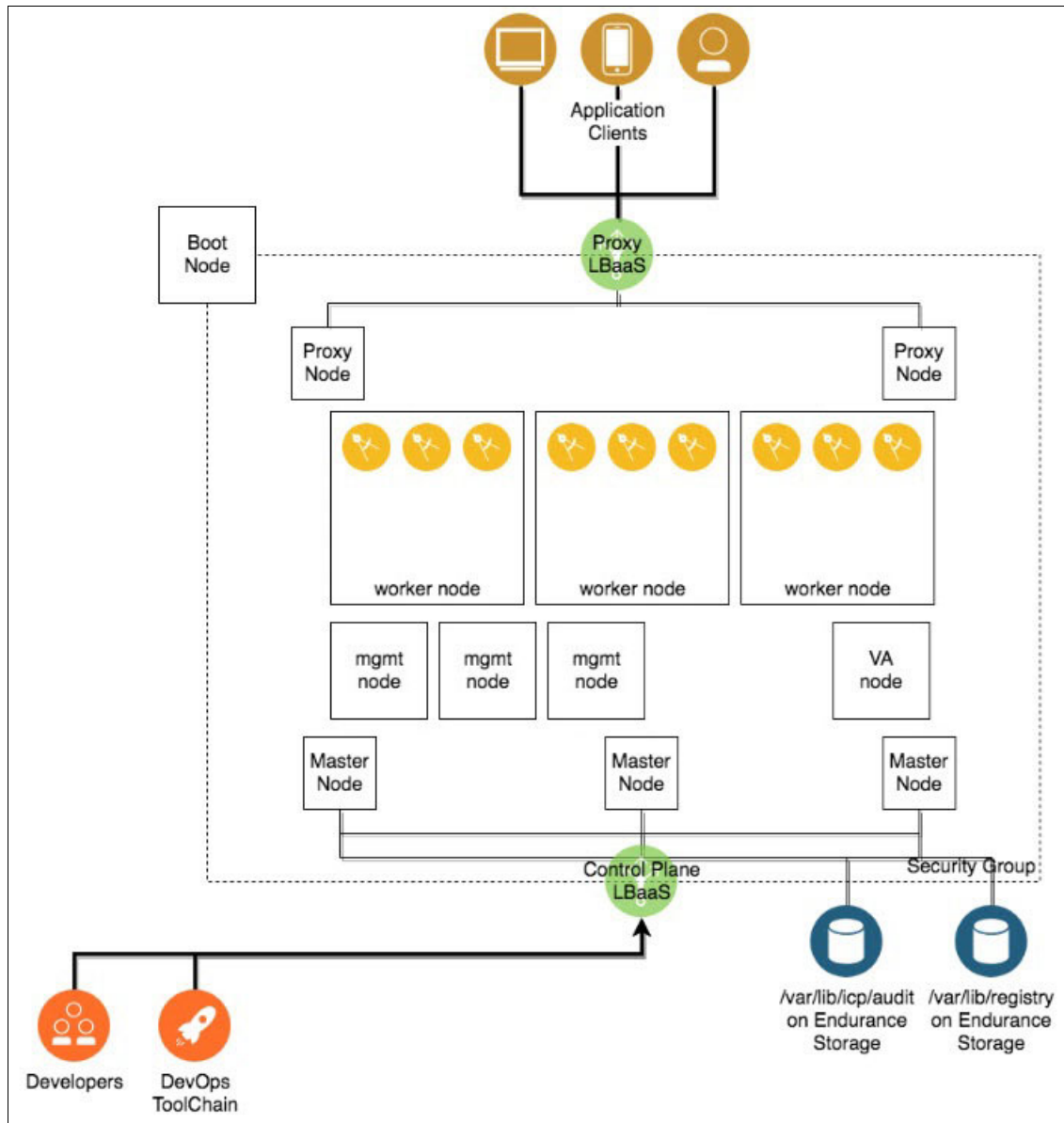


Figure 2-6 Architecture diagram for IBM Cloud Private cluster

You perform the following steps explained in this section:

1. Environment preparation.
2. Upload IBM Cloud Private installation binaries to the file server.
3. Configure the Terraform template.
4. Configure security.
5. Apply the Terraform template.

2.4.1 Environment preparation

In this section, you prepare your local environment with the pre-requisites needed to do the installation. You install Terraform, IBM Cloud provider for Terraform, and git client in this section. Skip this section in case you have these tools installed. You do this environment preparation on your local machine or in a Virtual Server Instance in IBM Cloud.

Note: These steps have been tested on MacOS and Ubuntu 18.04 Minimal LTS. Provision an Ubuntu 18.04 Virtual Server Instance in case you use other Operating System locally and run into a problem while preparing your environment or while applying the Terraform script. The specification of the machine needs to be at least 2 virtual cores with 4 GB of RAM.

Install Terraform

Install Terraform on your machine as you use it to run the scripts needed to install IBM Cloud Private. Perform the following steps to install Terraform:

1. Download the package corresponding to your operating system and architecture from the following URL:

<https://www.terraform.io/downloads.html>

Ubuntu: `wget`

`https://releases.hashicorp.com/terraform/0.11.11/terraform_0.11.11_linux_amd64.zip`

2. Extract the compressed folder. Perform the steps in Example 2-1 in case you have Ubuntu operating system.

Example 2-1 Extract the binaries of Terraform

```
sudo apt-get update
sudo apt-get install wget unzip
unzip terraform_0.11.11_linux_amd64.zip
```

3. Update your **PATH** environment variable to point to the directory that have the extracted binary or move the binary to binary directory:

Ubuntu: `sudo mv terraform /usr/local/bin/`

4. Verify that Terraform is installed successfully.

Open Terminal and write this command **terraform --version**, the Terraform version should appear if installed successfully.

Install IBM Cloud Provider plug-in for Terraform

In this step by step guidance, you perform installation of IBM Cloud Private on top of IBM Cloud infrastructure. For that Terraform needs to interact with IBM Cloud through its APIs. Terraform understands the provider APIs to access and manage its resources through Provider plug-ins.

Perform the following steps to install and configure IBM Cloud Provider plug-in:

1. Download the package corresponding to your operating system and architecture from the following URL:

<https://github.com/IBM-Cloud/terraform-provider-ibm/releases>


```
Ubuntu: wget
https://github.com/IBM-Cloud/terraform-provider-ibm/releases/download/v0.14.1/lin
ux_amd64.zip
```

2. Extract the compressed folder:

```
Ubuntu: unzip linux_amd64.zip
```

3. Create plugins directory in the following directory corresponding to your platform then move the extracted binary into it:

```
Windows: mkdir %APPDATA%\terraform.d\plugins
Linux/Unix/OS X: mkdir ~/.terraform.d/plugins
mv terraform-provider-ibm_v0.14.1 .terraform.d/plugins/
```

Install Git client

You use git client to clone the Terraform script. Perform the following steps to install Git client on your local machine.

1. Download and install the package corresponding to your operating system and architecture from the following URL:

<https://git-scm.com/download/>

```
Ubuntu: sudo apt install git
```

2. Verify that git is installed through running the command **git --version** from terminal.

2.4.2 Upload IBM Cloud Private binaries

The installation of IBM Cloud Private happens from the boot node. In this example, the boot node is a Virtual Server Instance (VSI) on IBM Cloud. That boot node needs access to the installation binaries.

Perform the steps in “Upload IBM Cloud Private binaries to a File Server” in case you run Terraform from your local machine in order to upload the binaries to File Server.

Upload IBM Cloud Private binaries to a File Server

In this section you upload the IBM Cloud Private installation binaries to a File Server accessible to the boot node on IBM Cloud. Skip the first two steps in case you have already a File Server.

1. Create Virtual Server Instance in IBM Cloud. You use this to install your File Server on top of it.
 - a. Navigate to <https://cloud.ibm.com/> and login as shown in Figure 2-7 on page 44.

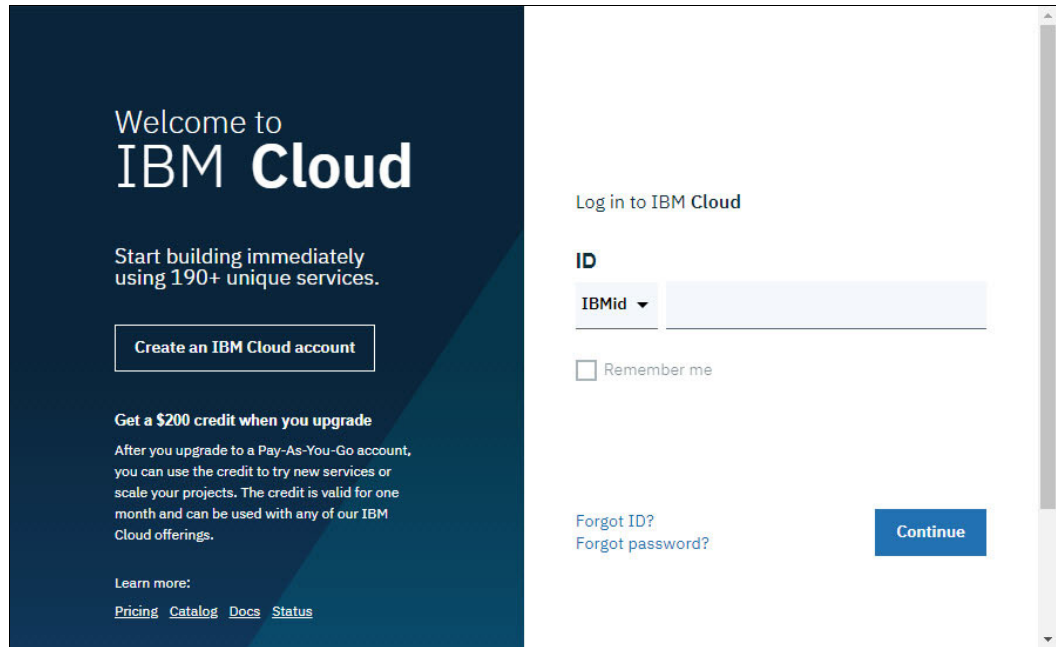


Figure 2-7 IBM Cloud console Login Page

- b. Click on **Catalog** from the above toolbar.
- c. Select **Virtual Server** from the catalog.
- d. Select **Public Virtual Server** then click **Continue**.
- e. Keep everything as default, and choose **Ubuntu** in Image Panel as shown in Figure 2-8.

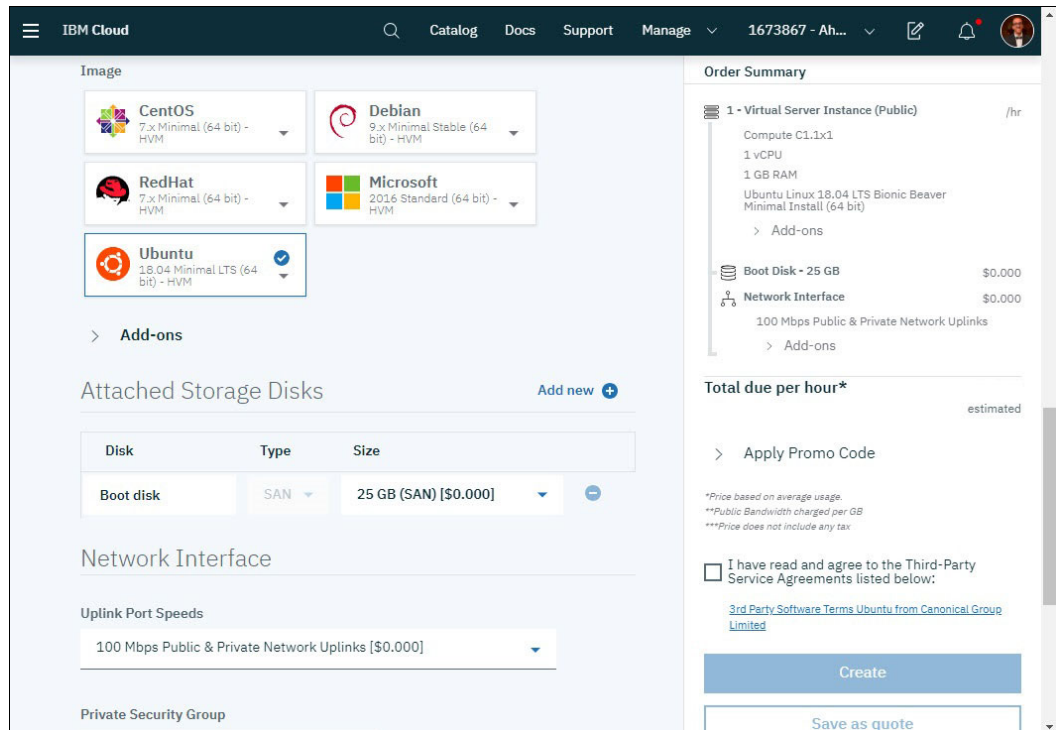


Figure 2-8 Create IBM Cloud Virtual Server Instance

- f. Read the terms and conditions then click on **I have read and agree to the Third-Party Service Agreements listed below:** then click on **Create**.
 - g. Wait till the virtual server is provisioned.
2. Install Apache HTTP Server on your Virtual Server Instance.

Note: The following steps shows the basic steps in order to create an Apache HTTP Server. Make sure to apply the security considerations to secure it. Skip this step if you have an existing HTTP Server or NFS server.

- a. SSH to your server with your root username and password.
- b. Update your local Linux packages:
`sudo apt update`
- c. Install apache2 package:
`sudo apt install apache2`
- d. Verify that the web server is running as shown in Example 2-2.

Example 2-2 Verify that the HTTP server is working

```
sudo systemctl status apache2
```

```
? apache2.service - The Apache HTTP Server
  Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor
  preset:
  Drop-In: /lib/systemd/system/apache2.service.d
           ..apache2-systemd.conf
  Active: active (running) since Tue 2019-02-19 17:54:06 UTC; 1min 12s ago
  Main PID: 2446 (apache2)
  Tasks: 55 (limit: 1145)
  CGroup: /system.slice/apache2.service
          ..2446 /usr/sbin/apache2 -k start
          ..2448 /usr/sbin/apache2 -k start
          ..2449 /usr/sbin/apache2 -k start
```

```
Feb 19 17:54:05 virtualserver01 systemd[1]: Starting The Apache HTTP
Server...
```

```
Feb 19 17:54:06 virtualserver01 systemd[1]: Started The Apache HTTP Server.
```

- e. Verify that the HTTP Server is accessible over the internet through navigating to `http://PUBLIC_IP` from your web browser. You should see the default page as shown in Figure 2-9.

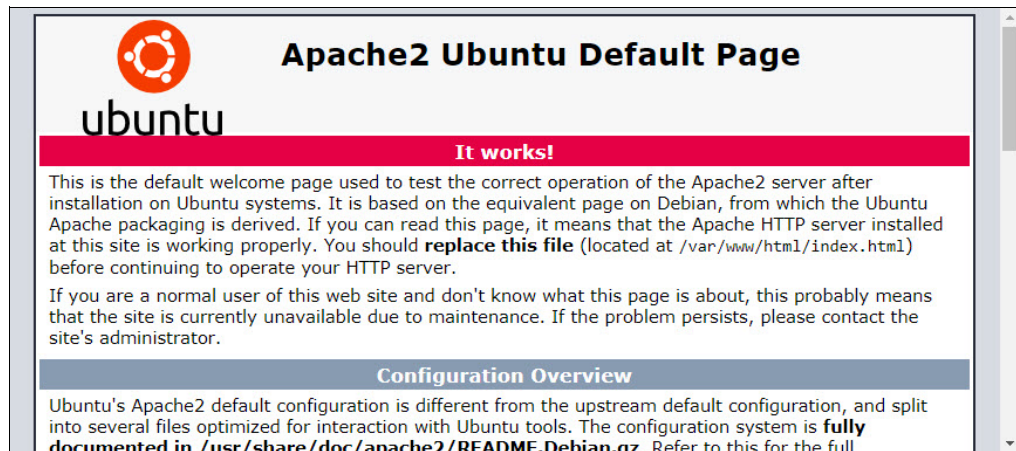


Figure 2-9 Apache HTTP server default page

3. Download IBM Cloud Private binaries from Passport Advantage such as `ibm-cloud-private-x86_64-3.1.2.tar.gz` for IBM Cloud Private Enterprise Edition V3.1.2.
4. Copy `ibm-cloud-private-x86_64-3.1.2.tar.gz` from your local machine to the HTTP Server on `/var/www/html/`.
5. Verify that you can access and download the binary file by navigating to `http://PUBLIC_IP/ibm-cloud-private-x86_64-3.1.2.tar.gz` from your web browser.

2.4.3 Configure the Terraform template

In this section, you configure the Terraform template with the instance specification. Perform the following steps in order to configure Terraform.

1. **git clone** the Terraform script project in your machine as shown in Example 2-3.

Note: The tag `3.1.2_redbook` is included in the **git clone** command. This is because the installation has been tested on this code base. Feel free to remove this tag in case you would like to work with the latest version of the script.

Example 2-3 Clone the GitHub repo that has the Terraform IBM Cloud Private installation script

```
git clone --branch 3.1.2_redbook \
https://github.com/ibm-cloud-architecture/terraform-icp-ibmcloud.git
```

```
Cloning into 'terraform-icp-ibmcloud'...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 215 (delta 31), reused 36 (delta 25), pack-reused 156
Receiving objects: 100% (215/215), 296.70 KiB | 334.00 KiB/s, done.
Resolving deltas: 100% (96/96), done.
Note: checking out '62eb61a8fd1b8434b7460591e73038e7b7635960'.
```

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

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

```
git checkout -b  
<new-branch-name>
```

Note: The terraform templates contained in this Github repository will be continuously updated by IBM team for upcoming IBM Cloud Private releases.

2. In order to allow Terraform to access your Cloud account, you need to provide the API username and API key of your IBM Cloud infrastructure account.

Note: In case you are not an account owner or Superuser, make sure you have the following permissions with IBM Cloud infrastructure:

- ▶ Manage Storage.
- ▶ Manage Security Groups.

- a. Navigate to your user profile on IBM Cloud infrastructure account through opening <https://control.softlayer.com/account/user/profile>.
- b. Copy the API Username and Authentication Key from the portal, as shown in Figure 2-10.

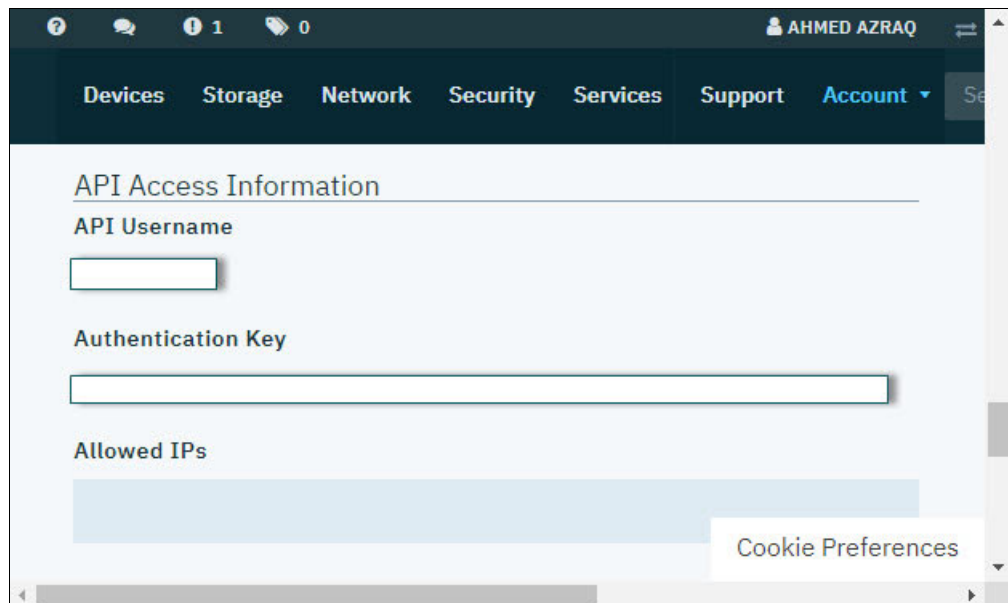


Figure 2-10 IBM Cloud infrastructure user profile

- c. On your local machine, navigate to `terraform-icp-ibmcloud/templates/icp-ee/`.
- d. Create a new file and name it `terraform.tfvars` that holds the configurations of your environment.

- e. Add in the file the two lines shown in Example 2-4. Replace `API_USERNAME` and `API_AUTH_KEY` with your IBM Cloud infrastructure Username and API key as retrieved in Figure 2-10.

Example 2-4 terraform.tfvars username and API key content

```
sl_username = "API_USERNAME"
sl_api_key = "API_AUTH_KEY"
```

3. Specify the data-center that will host all the Virtual Server Instances by add the following line to `terraform.tfvars`. Replace `dal10` with the intended data-center code.

```
datacenter = "dal10"
```

Note: Some customers have data residency requirements. Make sure that in these situations that the data-center configured in this step resides in the country of the customer. For example, if it is a Singaporean customer that has data-residency requirements, you will need to pick a data-center in Singapore, for example `SNG01` as shown in Figure 2-11.

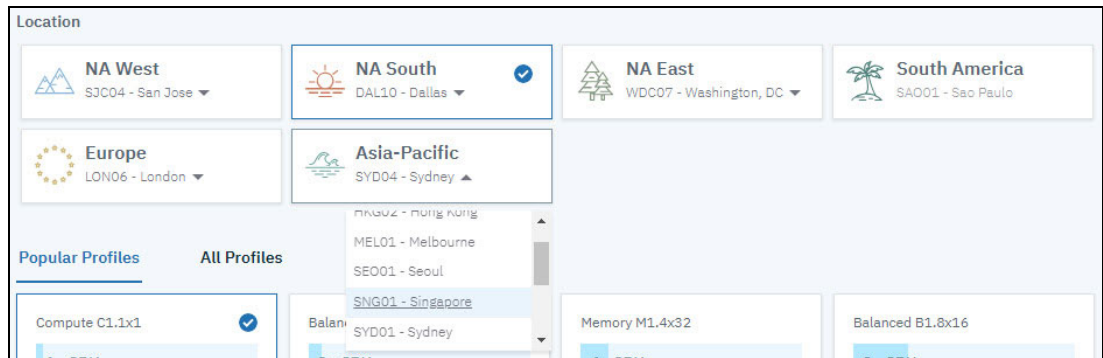


Figure 2-11 Data-Center code identification

4. Set the IBM Cloud Private Image to use for installation as shown. Get the image corresponding to your version from knowledge center.

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/installing/install_containers.html

For IBM Cloud Private Enterprise Edition 3.1.2, the image is `ibmcom/icp-inception-amd64:3.1.2-ee`. Add the following line to `terraform.tfvars`:

```
icp_inception_image = "ibmcom/icp-inception-amd64:3.1.2-ee"
```

5. Specify the location for the image by adding the following line to `terraform.tfvars`. Image Location can NFS, HTTP, or a private registry. Replace `IMAGE_LOCATION` with the location of your image that you have defined in 2.4.2, "Upload IBM Cloud Private binaries" on page 43.

```
image_location = "IMAGE_LOCATION"
```

6. Set IBM Cloud Private admin password by adding the following line to `terraform.tfvars`

```
icppassword = "IBM-Cloud-Private-Admin-Redbooks".
```

Note: Starting from IBM Cloud Private 3.1.2, by default the admin password should comply with a specific regular expression enforcement rule `'^([a-zA-Z0-9\-_]{32,})$'`. This regular expression can be changed during installation.

7. Specify the cluster name by adding the following line to terraform.tfvars:
deployment = "redbooks-deployment"
domain = "redbooks-deployment.icp"
8. Specify which management services will be disabled by adding the following line to terraform.tfvars. For IBM Cloud Private 3.1.2, the following management services can be disabled (custom-metrics-adapter, image-security-enforcement, istio, metering, logging, monitoring, service-catalog, storage-minio, storage-glusterfs, and vulnerability-advisor).
disabled_management_services = ["storage-glusterfs", "storage-minio"]
9. Specify the number and configuration of nodes as described in 2.1, “High availability considerations” on page 32. Add the following lines shown in Example 2-5 to terraform.tfvars with the specifications.

Example 2-5 IBM Cloud Private nodes specification

```
master = {  
  nodes = "3"  
  cpu_cores = "8"  
  memory = "32768"  
  disk_size = "100" // GB  
  docker_vol_size = "400" // GB  
}  
  
mgmt = {  
  nodes = "2"  
  cpu_cores = "8"  
  memory = "16384"  
  disk_size = "100" // GB  
  docker_vol_size = "400" // GB  
}  
  
va = {  
  nodes = "1"  
  cpu_cores = "8"  
  memory = "16384"  
  disk_size = "100" // GB  
  docker_vol_size = "400" // GB  
}  
  
proxy = {  
  nodes = "2"  
  cpu_cores = "4"  
  memory = "16384"  
  disk_size = "100" // GB  
  docker_vol_size = "300" // GB  
}  
  
worker = {  
  nodes = "3"  
  cpu_cores = "8"  
  memory = "32768"  
  disk_size = "100" // GB  
  docker_vol_size = "400" // GB  
}
```

10. Check `variables.tf` to know the other options that you can change by overriding in `terraform.tfvars`.
11. In case you plan to run `glusterfs` or `ceph` in your worker nodes, then you need to have a dedicated disk for each to run. Perform the following steps to add two additional disks on the worker nodes.

- a. Open `instances.tf` and locate `disks` variable under resource `"ibm_compute_vm_instance" "icp-worker"`
- b. Add the following two lines to define a pointer for the two new variables defining the additional disks for worker nodes:

```
"${var.worker["disk3_size"]}",
"${var.worker["disk4_size"]}"
```

- c. Your `disks` variable should look as Example 2-6.

Example 2-6 disks variable inside "ibm_compute_vm_instance" "icp-worker"

```
disks = [
    "${var.worker["disk_size"]}",
    "${var.worker["docker_vol_size"]}",
    "${var.worker["disk3_size"]}",
    "${var.worker["disk4_size"]}"
]
```

- d. Notice in the previous step, you added a reference to `worker["disk3_size"]`, and `"${var.worker["disk4_size"]}"` variables. Define these variables in `variables.tf` inside variable `"worker"` block and add a default value for it. Your variable `"worker"` block should now look as shown in Example 2-7.

Example 2-7 Variable worker in variables.tf after adding the two additional variables

```
variable "worker" {
    type = "map"

    default = {
        nodes      = "3"

        cpu_cores   = "4"
        memory      = "16384"

        disk_size    = "100" // GB
        docker_vol_size = "100" // GB
        local_disk   = false

        network_speed = "1000"

        hourly_billing = true

        disk3_size = "100" // GB
        disk4_size = "100" //GB
    }
}
```

- e. Override the values in `terraform.tfvars` with the intended size of disk 3 and disk 4 as shown in Example 2-8.

Example 2-8 Defining the size of disk 3 and disk 4 for worker nodes

```
worker = {  
  disk3_size = "200"  
  disk4_size = "200"  
}
```

2.4.4 Apply the Terraform template

Perform the following steps in order to apply the Terraform template:

1. Initialize Terraform in the script by writing the command shown in Example 2-9.

Example 2-9 Init Terraform

terraform init

```
Initializing modules...  
- module.icpprovision  
  Getting source  
"git::https://github.com/IBM-CAMHub-Open/template_icp_modules.git?ref=2.3//publ  
ic_cloud"
```

```
Initializing provider plugins...  
- Checking for available provider plugins on https://releases.hashicorp.com...  
- Downloading plugin for provider "tls" (1.2.0)...  
- Downloading plugin for provider "null" (2.0.0)...  
- Downloading plugin for provider "random" (2.0.0)...
```

The following providers do not have any version constraints in configuration, so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking changes, it is recommended to add version = "... constraints to the corresponding provider blocks in configuration, with the constraint strings suggested below.

```
* provider.ibm: version = "~> 0.14"  
* provider.null: version = "~> 2.0"  
* provider.random: version = "~> 2.0"  
* provider.tls: version = "~> 1.2"
```

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

2. Run **terraform plan** to know the changes that Terraform will make. The output you receive should be similar to Example 2-10.

Example 2-10 terraform plan output

Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be persisted to local or remote state storage.

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

....
....

Plan: 66 to add, 0 to change, 0 to destroy.

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

3. Run **terraform apply** to start the deployment then confirm the action by typing yes. It takes around 3 hours.

Troubleshooting: In case the apply fails for any reason like network failure, you can perform **terraform apply** again; terraform preserves the state and will *not* start from scratch.

In some cases you may need to delete a resource manually, make any needed modifications, and perform **terraform apply** again. For example in case you would like to change the image location, you need to delete the resource `image_load` using **terraform destroy -target null_resource.image_load**. This is to make sure it reloads the image from the new path.

4. Example 2-11 shows a successful completion.

Example 2-11 Successful completion of the IBM Cloud Private installation

Apply complete! Resources: 66 added, 0 changed, 0 destroyed.

Outputs:

ICP Admin Password = IBM-Cloud-Private-Admin-Redbooks
ICP Admin Username = admin
ICP Console URL =
`https://redbooks-deployment-0665b912-1673867-dal10.1b.bluemix.net:8443`
ICP Console load balancer DNS (external) =
`redbooks-deployment-0665b912-1673867-dal10.1b.bluemix.net`
ICP Kubernetes API URL =
`https://redbooks-deployment-0665b912-1673867-dal10.1b.bluemix.net:8001`
ICP Proxy load balancer DNS (external) =
`redbooks-deployment-proxy-0665b912-1673867-dal10.1b.bluemix.net`

ICP Registry URL =
redbooks-deployment-0665b912-1673867-dal10.1b.bluemix.net:8500

Note: You can use similar steps to install IBM Cloud Private Community Edition in case you don't have a license for IBM Cloud Private. The Community Edition is free of charge and its terraform template exist in the same GitHub repository under folder icp-ce-minimal.

<https://github.com/ibm-cloud-architecture/terraform-icp-ibmcloud/tree/master/templates/icp-ce-minimal>.

You can also install IBM Cloud Private Community Edition locally to your PC through Vagrant.

<https://github.com/IBM/deploy-ibm-cloud-private/blob/master/docs/deploy-vagrant.md>.

2.5 Post installation verification

As a sanity validation step, perform the following steps in order to verify that your IBM Cloud Private cluster is installed successfully. You do the verification through the IBM Cloud Private Command Line Interface and IBM Cloud Private Console User Interface.

2.5.1 IBM Cloud Private command line interface

In this section, you verify that you can login to the IBM Cloud Private CLI and do some sanity commands.

For more details on how to install these tools, see Appendix A, “Command line tools” on page 347.

1. Perform the instructions in this Knowledge Center to install IBM Cloud Private CLI (Cloudctl):

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/manage_cluster/install_cli.html.

2. Follow the instructions in this Knowledge Center to download and install Kubernetes CLI (Kubectl):

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/manage_cluster/install_kubectl.html.

3. Verify that Cloudctl and Kubectl are installed by performing the following commands:

```
cloudctl version
kubectl version
```

4. Log in to your cluster with the following command, where `ibm_cloud_private_console_url` is the external host name or IP address for your master:

```
cloudctl login -a https://<ibm_cloud_private_console_url>:8443
--skip-ssl-validation
```

5. Make sure that there are no errors during login.

6. Make sure that all pods are in either status Running or Completed in kube-system through performing the command shown in Example 2-12.

Example 2-12 Verifying that pods are running

```
kubect1 get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
audit-logging-fluentd-ds-2fdws	1/1	Running	1	8d
audit-logging-fluentd-ds-6bcp6	1/1	Running	0	8d
audit-logging-fluentd-ds-dt6pd	1/1	Running	0	8d
audit-logging-fluentd-ds-hd7lp	1/1	Running	0	8d
audit-logging-fluentd-ds-m7nlc	1/1	Running	0	8d
audit-logging-fluentd-ds-mhkw5	1/1	Running	0	8d
audit-logging-fluentd-ds-tvsx9	1/1	Running	0	8d
audit-logging-fluentd-ds-v4v9f	1/1	Running	0	8d
audit-logging-fluentd-ds-xspfz	1/1	Running	0	8d
auth-apikeys-f45sd	1/1	Running	0	8d
auth-apikeys-sllwj	1/1	Running	1	8d
auth-apikeys-wq9dk	1/1	Running	0	8d
auth-idp-6p52r	4/4	Running	2	8d
auth-idp-fsnw4	4/4	Running	3	8d
auth-idp-k6znh	4/4	Running	7	8d
auth-pap-7wpdp	2/2	Running	3	8d
auth-pap-jhp7f	2/2	Running	0	8d
auth-pap-sk5j	2/2	Running	0	8d
auth-pdp-6q76t	2/2	Running	0	8d
auth-pdp-cd9wm	2/2	Running	3	8d
auth-pdp-lcm8t	2/2	Running	0	8d
calico-kube-controllers-85676f8998-dt2dv	1/1	Running	0	8d
calico-node-2l4ld	2/2	Running	1	8d
calico-node-8c65m	2/2	Running	0	8d
calico-node-8txd6	2/2	Running	2	8d
calico-node-9z28f	2/2	Running	0	8d
calico-node-cbrmm	2/2	Running	0	8d
calico-node-djvlf	2/2	Running	0	8d
calico-node-k4nlq	2/2	Running	0	8d
calico-node-wn9g9	2/2	Running	0	8d
calico-node-zcpg4	2/2	Running	0	8d
catalog-ui-9s2pc	1/1	Running	0	8d
catalog-ui-qfqsm	1/1	Running	2	8d
catalog-ui-zt2vx	1/1	Running	0	8d
custom-metrics-adapter-597f7645cf-74n5d	1/1	Running	0	8d
default-http-backend-54987cd76b-2rvbq	1/1	Running	0	8d
heapster-545458b4ff-q8d8k	1/1	Running	0	28h
heapster-545458b4ff-rd8pq	1/1	Running	0	8d
heapster-545458b4ff-tgv9f	1/1	Running	0	8d
helm-api-56fb65c854-tzgsd	3/3	Running	0	8d
helm-repo-847fbc5cd8-mc7q6	2/2	Running	0	8d
ibmcloud-image-enforcement-6d47d6d485-mwpq8	1/1	Running	0	8d
ibmcloud-image-enforcement-6d47d6d485-nkz9w	1/1	Running	0	28h
ibmcloud-image-enforcement-6d47d6d485-w6jnp	1/1	Running	0	8d
icp-management-ingress-b2tl8	1/1	Running	0	8d
icp-management-ingress-g94gj	1/1	Running	1	8d
icp-management-ingress-th2w8	1/1	Running	0	8d
icp-mongodb-0	1/1	Running	2	8d

icp-mongodb-1	1/1	Running	1	8d
icp-mongodb-2	1/1	Running	1	8d
image-manager-0	2/2	Running	2	8d
image-manager-1	2/2	Running	3	8d
image-manager-2	2/2	Running	1	8d
k8s-etcd-10.177.233.177	1/1	Running	0	8d
k8s-etcd-10.177.233.184	1/1	Running	0	8d
k8s-etcd-10.93.221.119	1/1	Running	1	8d
k8s-kmsplugin-10.177.233.177	1/1	Running	0	8d
k8s-kmsplugin-10.177.233.184	1/1	Running	0	8d
k8s-kmsplugin-10.93.221.119	1/1	Running	1	8d
k8s-master-10.177.233.177	3/3	Running	3	8d
k8s-master-10.177.233.184	3/3	Running	1	8d
k8s-master-10.93.221.119	3/3	Running	4	8d
k8s-proxy-10.171.37.135	1/1	Running	0	8d
k8s-proxy-10.177.233.176	1/1	Running	0	8d
k8s-proxy-10.177.233.177	1/1	Running	0	8d
k8s-proxy-10.177.233.181	1/1	Running	0	8d
k8s-proxy-10.177.233.184	1/1	Running	0	8d
k8s-proxy-10.93.221.105	1/1	Running	0	8d
k8s-proxy-10.93.221.119	1/1	Running	1	8d
k8s-proxy-10.93.221.68	1/1	Running	0	8d
k8s-proxy-10.93.221.97	1/1	Running	0	8d
key-management-api-787d976f79-7qnjz	1/1	Running	0	8d
key-management-crypto-558dff975-82kgj	1/1	Running	0	8d
key-management-lifecycle-5bdfdf7978c-18bfc	2/2	Running	0	8d
key-management-onboarding-pgwvg	0/1	Completed	0	8d
key-management-pep-7b864d6c88-9dlts	1/1	Running	0	8d
key-management-persistence-75ff587647-9kw5g	1/1	Running	0	8d
kube-dns-29r6d	1/1	Running	0	8d
kube-dns-5td8z	1/1	Running	1	8d
kube-dns-klbk7	1/1	Running	0	8d
logging-elk-client-564c6c7894-b7t4b	2/2	Running	0	8d
logging-elk-data-0	1/1	Running	0	8d
logging-elk-elasticsearch-curator-1550619000-7n6cc	0/1	Completed	0	45h
logging-elk-elasticsearch-curator-1550705400-s8wvp	0/1	Completed	0	21h
logging-elk-elasticsearch-pki-init-kml4g	0/1	Completed	0	8d
logging-elk-filebeat-ds-26g4l	1/1	Running	0	8d
logging-elk-filebeat-ds-9ns27	1/1	Running	0	8d
logging-elk-filebeat-ds-f9dt2	1/1	Running	0	8d
logging-elk-filebeat-ds-fb9bt	1/1	Running	1	8d
logging-elk-filebeat-ds-jr26q	1/1	Running	0	8d
logging-elk-filebeat-ds-k8f4x	1/1	Running	1	8d
logging-elk-filebeat-ds-nhf5v	1/1	Running	0	8d
logging-elk-filebeat-ds-zdxv6	1/1	Running	1	8d
logging-elk-filebeat-ds-zt7s5	1/1	Running	0	8d
logging-elk-kibana-578c8dcc6c-q58st	2/2	Running	0	8d
logging-elk-kibana-init-q8gw2	0/1	Completed	6	8d
logging-elk-logstash-b54c9bb89-4glgr	1/1	Running	0	8d
logging-elk-master-9d5866588-xrpsw	1/1	Running	0	8d
mariadb-0	2/2	Running	0	8d
mariadb-1	2/2	Running	12	8d
mariadb-2	2/2	Running	0	8d
metering-dm-57b569b5c9-jxf2t	1/1	Running	0	8d
metering-reader-7c74p	1/1	Running	0	8d

metering-reader-7fj9f	1/1	Running	0	8d
metering-reader-g5767	1/1	Running	0	8d
metering-reader-gcd47	1/1	Running	0	8d
metering-reader-jx8zx	1/1	Running	0	8d
metering-reader-l8d29	1/1	Running	0	8d
metering-reader-lwzxv	1/1	Running	3	8d
metering-reader-tlq52	1/1	Running	0	8d
metering-reader-v546l	1/1	Running	0	8d
metering-ui-7d68d4dc79-r5fhx	1/1	Running	0	8d
metrics-server-6d47b6dc6b-przd1	1/1	Running	0	8d
mgmt-repo-7c9cf9dc89-2djtq	2/2	Running	0	8d
monitoring-grafana-5fc5676bdf-794bd	3/3	Running	0	8d
monitoring-prometheus-56988655cf-j66ht	4/4	Running	0	8d
monitoring-prometheus-alertmanager-8c5fcb784-f55bs	3/3	Running	0	8d
monitoring-prometheus-collectdexporter-cc9f4656c-d7bxq	2/2	Running	0	8d
monitoring-prometheus-elasticsearchexporter-6565fb986-z7df8	2/2	Running	0	8d
monitoring-prometheus-kubestatemetrics-5fb65cd8b-gwtv8	2/2	Running	0	8d
monitoring-prometheus-nodeexporter-28z4n	2/2	Running	0	8d
monitoring-prometheus-nodeexporter-845cl	2/2	Running	2	8d
monitoring-prometheus-nodeexporter-fbd5s	2/2	Running	0	8d
monitoring-prometheus-nodeexporter-hfxtw	2/2	Running	0	8d
monitoring-prometheus-nodeexporter-mns4t	2/2	Running	0	8d
monitoring-prometheus-nodeexporter-qrjtp	2/2	Running	0	8d
monitoring-prometheus-nodeexporter-sxgl9	2/2	Running	0	8d
monitoring-prometheus-nodeexporter-w7f5t	2/2	Running	0	8d
monitoring-prometheus-nodeexporter-xq8zj	2/2	Running	0	8d
nginx-ingress-controller-lcz9k	1/1	Running	0	8d
nginx-ingress-controller-znqr7	1/1	Running	0	8d
nvidia-device-plugin-5gpxt	1/1	Running	0	8d
nvidia-device-plugin-7g7p4	1/1	Running	1	8d
nvidia-device-plugin-82dp7	1/1	Running	0	8d
nvidia-device-plugin-cb9lw	1/1	Running	0	8d
nvidia-device-plugin-hgrnx	1/1	Running	0	8d
nvidia-device-plugin-jt4j6	1/1	Running	0	8d
nvidia-device-plugin-mb4x5	1/1	Running	0	8d
nvidia-device-plugin-qvgsp	1/1	Running	0	8d
nvidia-device-plugin-xmlf7	1/1	Running	0	8d
oidc-client-registration-x7hgh	0/1	Completed	0	8d
platform-api-758ff75d7b-b8wlk	2/2	Running	0	8d
platform-api-758ff75d7b-cg5lt	2/2	Running	0	28h
platform-api-758ff75d7b-j6ggv	2/2	Running	0	8d
platform-deploy-6744cfb478-77stf	1/1	Running	0	8d
platform-deploy-6744cfb478-pk48z	1/1	Running	0	28h
platform-deploy-6744cfb478-qng4v	1/1	Running	0	8d
platform-ui-kz6jn	1/1	Running	0	8d
platform-ui-mmvlq	1/1	Running	0	8d
platform-ui-vffd4	1/1	Running	1	8d
secret-watcher-777978d867-qvg78	1/1	Running	0	8d
security-onboarding-bk4qv	0/1	Completed	0	8d
service-catalog-apiserver-5n5rn	1/1	Running	1	8d
service-catalog-apiserver-bk5k6	1/1	Running	0	8d
service-catalog-apiserver-jlqr7	1/1	Running	0	8d
service-catalog-controller-manager-6cd6684cf7-jztjh	1/1	Running	0	28h
tiller-deploy-854fc7d5d-pkbv7	1/1	Running	0	28h
unified-router-86jnr	1/1	Running	0	8d

unified-router-8zbh4	1/1	Running	0	8d
unified-router-zvf6p	1/1	Running	1	8d
web-terminal-cd5674776-pmxf9	1/1	Running	0	8d

7. Verify that all nodes are in Ready Status through performing the command in Example 2-13.

Example 2-13 Verify that all nodes are created

kubect1 get nodes

NAME	STATUS	ROLES	AGE	VERSION
10.171.37.135	Ready	proxy	8d	v1.12.4+icp-ee
10.177.233.176	Ready	worker	8d	v1.12.4+icp-ee
10.177.233.177	Ready	etcd,master	8d	v1.12.4+icp-ee
10.177.233.181	Ready	worker	8d	v1.12.4+icp-ee
10.177.233.184	Ready	etcd,master	8d	v1.12.4+icp-ee
10.93.221.105	Ready	management	8d	v1.12.4+icp-ee
10.93.221.107	Ready	management	8d	v1.12.4+icp-ee
10.93.221.119	Ready	etcd,master	8d	v1.12.4+icp-ee
10.177.233.171	Ready	worker	8d	v1.12.4+icp-ee
10.93.221.68	Ready	proxy	8d	v1.12.4+icp-ee
10.93.221.97	Ready	va	8d	v1.12.4+icp-ee

2.5.2 IBM Cloud Private Console user interface

In this section, you make sure that you can access the Console UI, access the catalog, and perform a sample application deployment.

1. Navigate to the URL of your IBM Cloud Private Console as displayed at the end of the installation output in `ibm_cloud_private_console_url`.

2. In the log in page, enter the admin username and password that you have specified during installation then click **Log in** as shown in Figure 2-12.

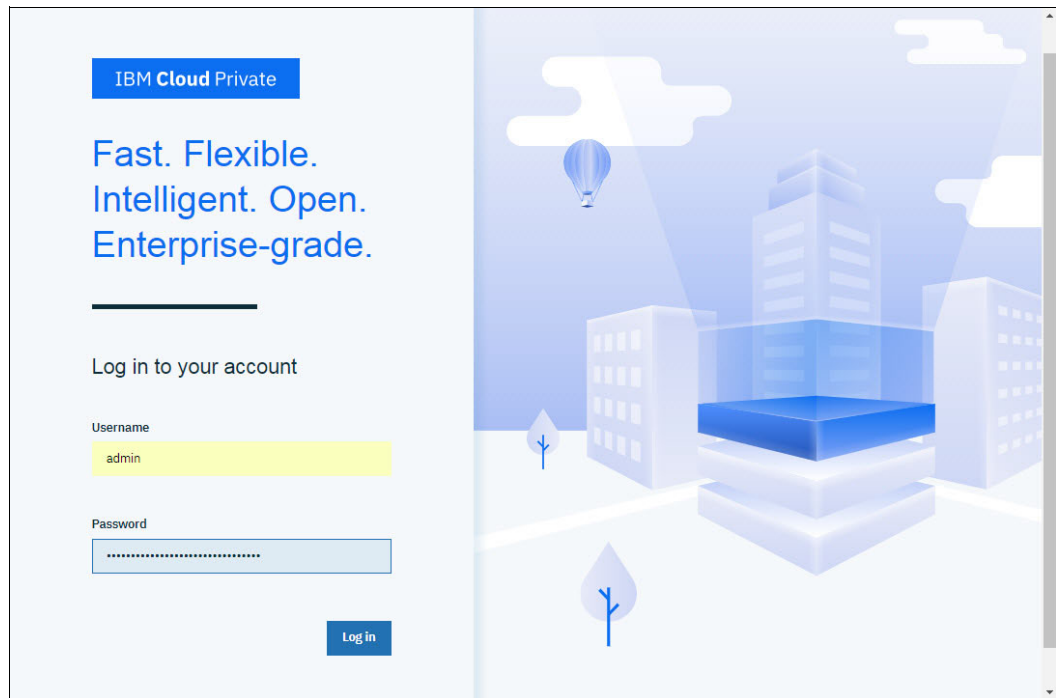


Figure 2-12 IBM Cloud Private login screen

3. After you login, you should be able to see “Welcome to IBM Cloud Private” screen as shown in Figure 2-13.

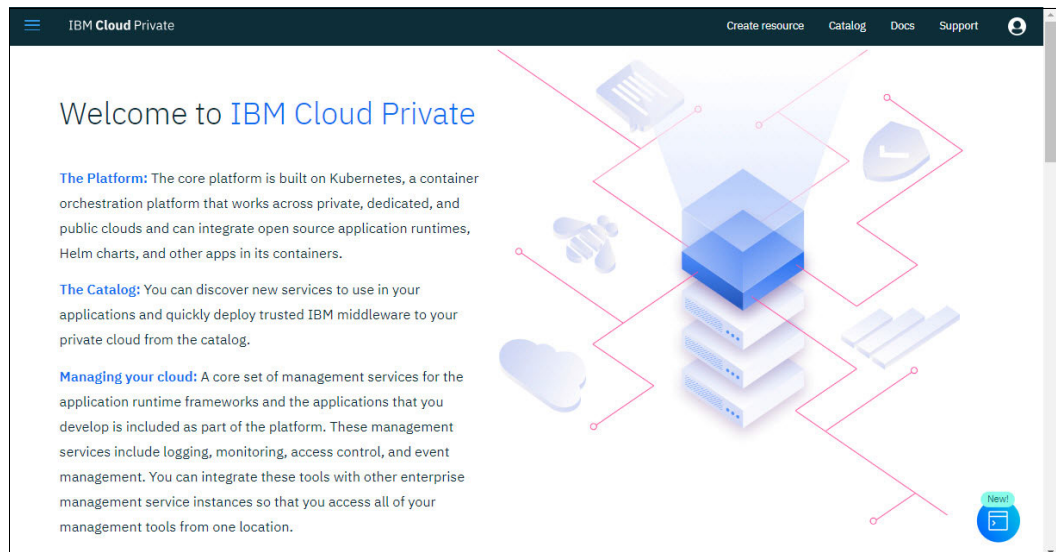


Figure 2-13 IBM Cloud Private welcome screen

4. Click on **Catalog** in the above toolbar and make sure that the Helm charts are loaded as shown Figure 2-14.

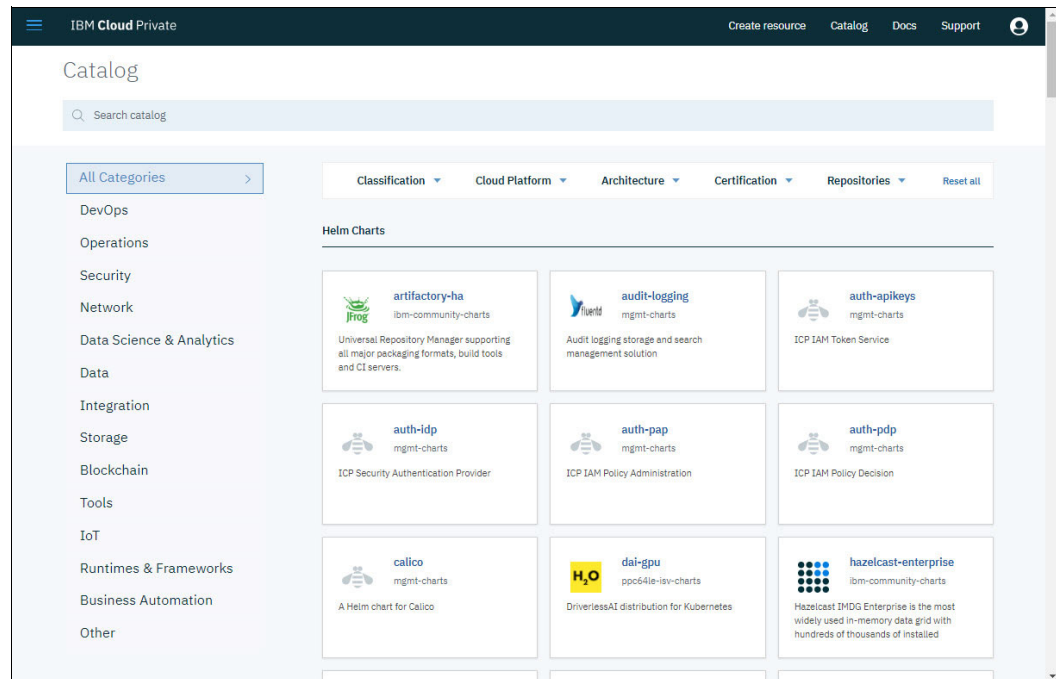


Figure 2-14 IBM Cloud Private Catalog

5. Try to deploy a sample helm-chart like `ibm-nodejs-sample`. Select `ibm-nodejs-sample` from the catalog.

6. The documentation of the helm-chart should appear as shown in Figure 2-15. Click **Configure** to proceed.

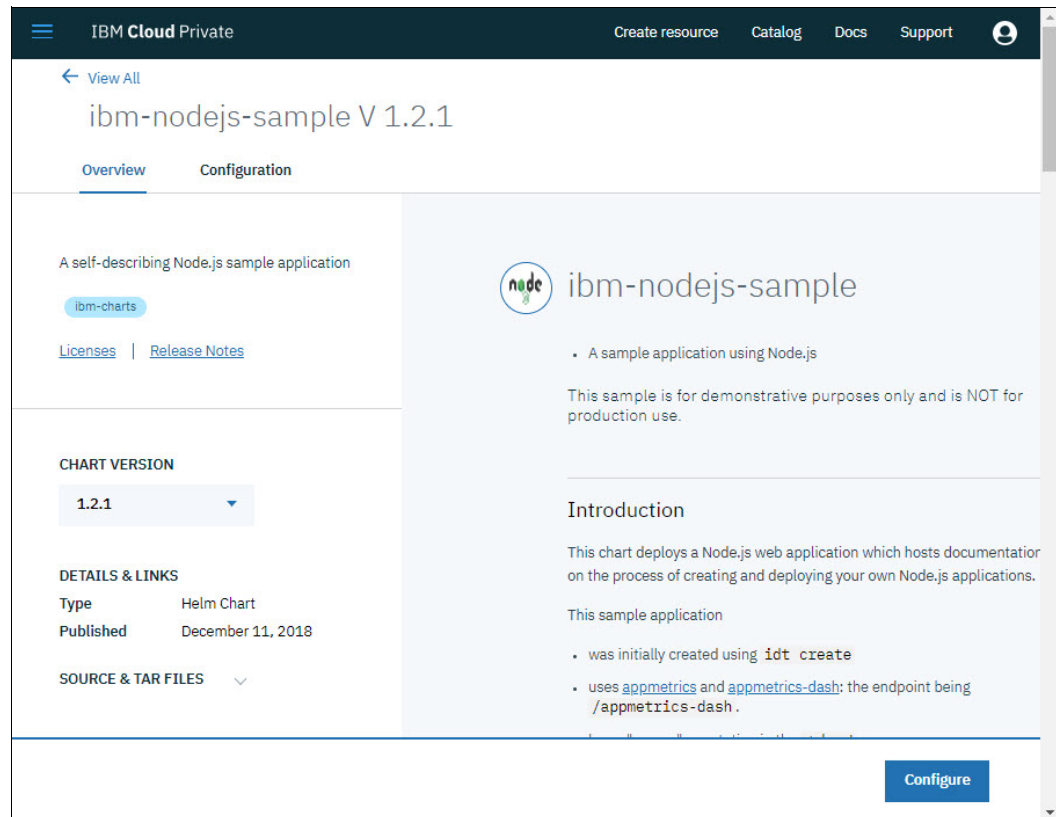


Figure 2-15 *ibm-nodejs-sample Info page*

7. Input `nodejs-test` in Helm release name and choose default in Target namespace then click on **Install** as shown in Figure 2-16.

The screenshot shows the IBM Cloud Private interface for configuring a Helm chart. The top navigation bar includes 'IBM Cloud Private', 'Create resource', 'Catalog', 'Docs', 'Support', and a user profile icon. Below the navigation bar, there is a 'View All' link and the chart title 'ibm-nodejs-sample V 1.2.1'. The 'Configuration' tab is selected, showing a 'Configuration' section with a description: 'A self-describing Node.js sample application. Edit these parameters for configuration.' The 'Helm release name' field is set to 'nodejs-test' and the 'Target namespace' dropdown is set to 'default'. The 'License' section has a checked checkbox for 'I have read and agreed to the License agreement'. The 'Pod Security' section contains a warning message: 'This chart does not specify a pod security policy. Select a Namespace with `ibm-anyuid-hostpath-psp`, or reference the [chart security reference table](#) for a list of charts with known pod security policies defined.' At the bottom right, there are 'Cancel' and 'Install' buttons.

Figure 2-16 Configure Helm chart

8. Click on **View Helm Release** as shown in Figure 2-17 to see the status of the deployment.

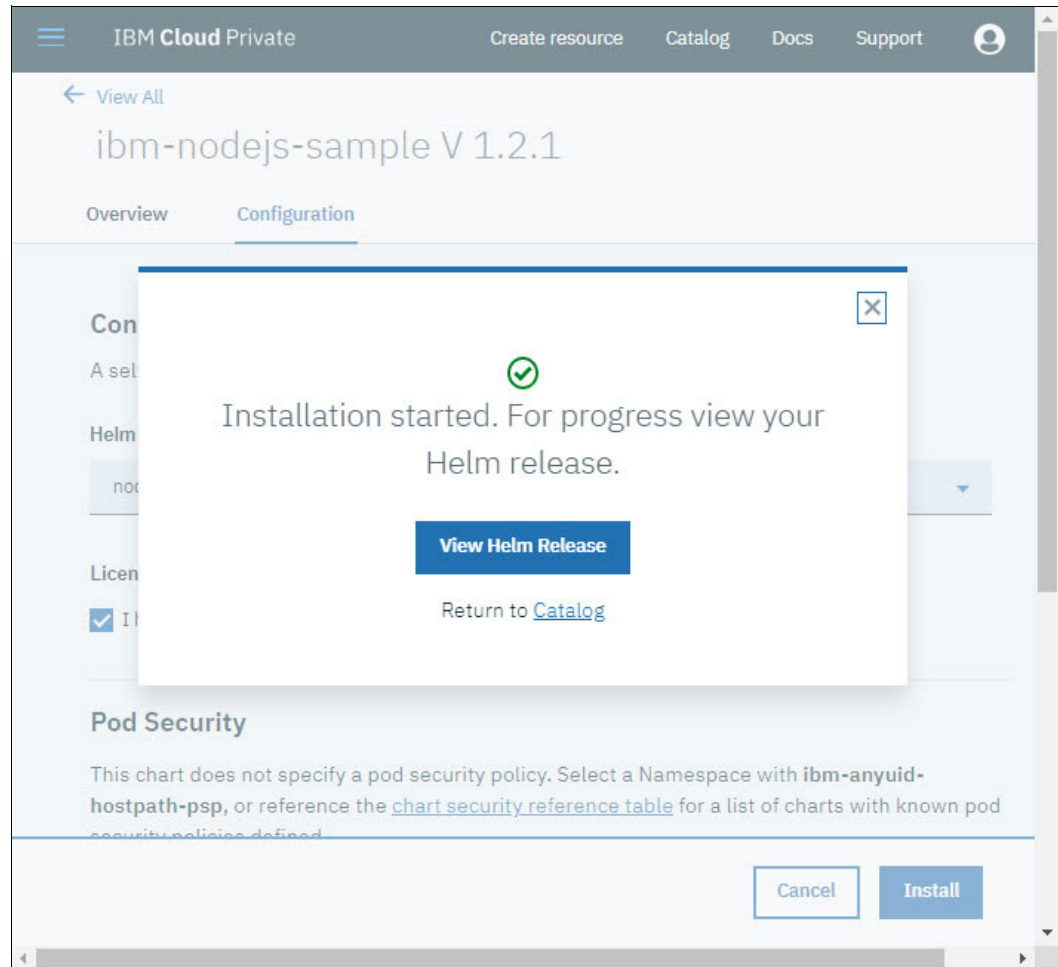


Figure 2-17 View Helm release dialog

- Wait till the deploy is successful, your screen should be similar to Figure 2-18. Make sure that the available deployment is 1, Pod is in running status.

The screenshot shows the IBM Cloud Private interface for a Helm release named 'nodejs-test'. The release is in a 'Deployed' state, updated on February 20, 2019, at 12:36 PM. A 'Launch' button is visible in the top right.

Details and Upgrades

CHART NAME	CURRENT VERSION	AVAILABLE VERSION	
ibm-nodejs-sample	1.2.1 <small>Installed: February 20, 2019 → Release Notes</small>	1.2.1 <small>Released: December 11, 2018 → Release Notes</small>	<div>Upgrade</div> <div>Rollback</div>
NAMESPACE	default		

Deployment

NAME	DESIRED	CURRENT	UP TO DATE	AVAILABLE	AGE
nodejs-test-nodejs-sample-nodejs	1	1	1	1	2h

Pod

NAME	READY	STATUS	RESTARTS	AGE	
nodejs-test-nodejs-sample-nodejs-866557c4bb-pc98m	1/1	Running	0	2h	View Logs

Service

NAME	TYPE	CLUSTER IP	EXTERNAL IP	PORT(S)	AGE
nodejs-test-nodejs-sample-nodejs	NodePort	172.21.158.69	<none>	3000:30234/TCP	2h

Figure 2-18 Helm release status

2.6 Installing IBM Cloud Private on other Cloud platforms

Kubernetes has become the de facto standard in managing container-based workloads. IBM Cloud Private builds on top of the Kubernetes orchestrator and provides a private cloud platform for developing and running a container workload solely for one organization. It is a reliable way of running containerized IBM middleware given the knowledge and verification test applied to both.

2.6.1 Typical scenario of running IBM Cloud Private on other Cloud platforms

Below are four “why” scenarios for clients considering IBM Cloud Private running on other Cloud infrastructure:

- ▶ **Multicloud strategy:** Customers would like to leverage the strength and unique offerings from different cloud vendors, but also want to have a consistent operation and runtime environment that they can achieve portability without cloud platform lock-in. For example, running digital innovation on both IBM public cloud and AWS.
- ▶ **Cloud bursting:** If you have private cloud environments running on-prem and would like to expand the cluster or private cloud to external infrastructure only in certain special condition or bursting workload.
- ▶ **Disaster recovery:** Since the same workload can be easily and quickly provisioned, external cloud providers can also be a great place to act as a disaster recovery data center. IBM Cloud Private on other cloud provider fits this use case nicely.
- ▶ **Existing other cloud users with IBM Middleware workload:** IBM middleware investments are further extended with an app modernization strategy. IBM's middleware also benefits from associated data analytic services, such as IBM Cognos Analytics to gain a deeper understanding of your business and IBM SPSS statistical analysis to predict outcomes; you can continue deploying them in your cloud environment by leveraging ICP. This approach gives the best of both worlds.

2.6.2 Installing IBM Cloud Private on AWS using Terraform

This repository url below contains a collection of Terraform templates to install IBM Cloud Private on Google Cloud.

<https://github.com/ibm-cloud-architecture/terraform-icp-aws.git>

2.6.3 Installing IBM Cloud Private on Microsoft Azure using Terraform

This repository url below contains a collection of Terraform templates to install IBM Cloud Private on Google Cloud.-CAMHub-Open/template_icp_azure.git

<https://github.com/ibm-cloud-architecture/terraform-icp-azure.git>

2.6.4 Installing IBM Cloud Private on Google Cloud using Terraform

This repository url below contains a collection of Terraform templates to install IBM Cloud Private on Google Cloud.

<https://github.com/ibm-cloud-architecture/terraform-icp-gcp.git>

2.6.5 Installing IBM Cloud Private on RedHat OpenShift

IBM and Red Hat have partnered to provide a joint solution that uses IBM Cloud Private and OpenShift. You can deploy IBM certified software containers running on IBM Cloud Private onto Red Hat OpenShift. Similar to IBM Cloud Private, OpenShift is a container platform built on top of Kubernetes. You can install IBM Cloud Private on OpenShift by using the IBM Cloud Private installer for OpenShift.

For more details, see the following IBM Knowledge Center URL:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/supported_environments/openshift/overview.html

Note: Pre-requisites to install IBM Cloud Private on other Cloud providers:

- ▶ Basic understanding of IBM Cloud Private
- ▶ Cloud account
- ▶ Access to IBM Cloud Private Images

2.6.6 Installing IBM Cloud Private on OpenStack Cloud provider

This repository url below contains a collection of Terraform templates to install IBM Cloud Private on OpenStack Cloud provider. This Terraform example configurations uses the OpenStack provider to provision virtual machines on OpenStack and TerraForm Module ICP Deploy to prepare VMs and deploy IBM Cloud Private on them.

<https://github.com/ibm-cloud-architecture/terraform-icp-openstack.git>

2.6.7 Installing IBM Cloud Private on VMware vSphere Cloud provider

This repository url below contains a collection of Terraform templates to install IBM Cloud Private on VMWare vSphere Cloud provider. This Terraform example configurations uses the VMware vSphere provider to provision virtual machines on VMware and TerraForm Module ICP Deploy to prepare VMs and deploy IBM Cloud Private on them. This Terraform template automates best practices learned from installing ICP on VMware at numerous client sites in production.

<https://github.com/ibm-cloud-architecture/terraform-icp-vmware.git>

2.6.8 Install IBM Cloud Private on existing Virtual Machines

This repository url below contains a collection of Terraform templates to install IBM Cloud Private on Vans already provisioned on-premise or on any cloud provider. This Terraform module can be used to deploy IBM Cloud Private on any supported infrastructure vendor. Tested on Ubuntu 16.04 and RHEL 7 on SoftLayer, VMware, AWS and Azure.

<https://github.com/ibm-cloud-architecture/terraform-module-icp-deploy.git>

2.7 Setting up IBM Cloud Private catalog in an airgap environment

An airgapped environment is an environment where a computer or network is prevented from establishing an external connection due to security reasons.

Generally post installing IBM Cloud Private on a internet connection enabled host, the IBM Charts will be synced automatically and all the IBM charts with its respective images are synced to the local IBM Cloud Private cluster.

However, in production environments due to client security policies, direct access to internet is disabled resulting in empty catalog on the IBM Cloud Private admin GUI. The below steps are documented in getting the desired chart to be available on IBM Cloud Private. IBM Liberty chart is taken as an example in the below steps.

2.7.1 Prerequisites

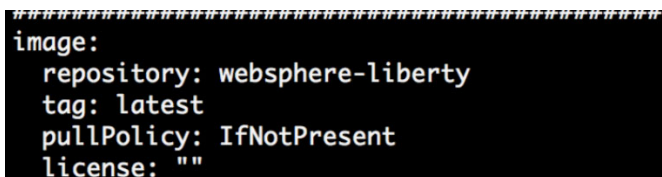
The following are the prerequisites for setting up IBM Cloud Private catalog in an airgap environment:

- ▶ Ensure you have git configured on the local system.
- ▶ Ensure Docker is installed on the local system, and has internet access.
- ▶ Install and configure Helm on the local system.
- ▶ Install IBM Cloud Private command line interface (CLI).

2.7.2 Steps to follow

Perform the following steps:

1. Obtain the Helm chart for the image. Clone the /IBM/Charts repository from git using: **git clone https://github.com/IBM/charts.git**.
2. Open the values.yaml file from the chart and locate the image that it uses. In this example, the repository is “websphere-liberty”, and tag is “latest” as shown in Figure 2-19.



```
image:
  repository: websphere-liberty
  tag: latest
  pullPolicy: IfNotPresent
  license: ""
```

Figure 2-19 Liberty chart

3. Pull the image locally using the repository and tag information from the values.yaml file:
docker pull <repository>:<tag>. In this example, it would be:
docker pull websphere-liberty:latest
4. Upload the image to the private image repository on IBM Cloud Private:
docker login <cluster_CA_domain>:8500
Tag the image: docker tag image_name.
<cluster_CA_domain:8500>/namespace/imagename:tagname. In our example:
docker tag websphere-liberty:latest
mycluster.icp:8500/default/websphere-liberty:latest.
5. Push the image to private repository:
docker push mycluster.icp:8500/default/websphere-liberty:latest
6. Open the values.yaml file again, and update the location of the Docker image, as shown in Figure 2-20 on page 67:
repository: <cluster_CA_domain>:8500/namespace/imagename tag: <tagname>


```
image:
  repository: mycluster.icp:8500/default/websphere-liberty
  tag: latest
  pullPolicy: IfNotPresent
  license: ""
```

Figure 2-20 Modify chart repository

7. Repackage the Helm chart: `helm package <path_to_helm_directory>` where `<path_to_helm_directory>` is the directory that contains the Helm chart. On running this, a `.tgz` file containing the Helm chart is created. In our example, the file `ibm-websphere-liberty-1.0.1.tar.tgz` was created.

8. Upload the new Helm chart to your IBM Cloud Private cluster:

```
bx pr login -a <ICP_master_ip>:8443 --skip-ssl-validation
```

To upload the Helm chart `ibm-websphere-liberty-1.0.1.tgz`, use the following command:

```
bx pr load-helm-chart --archive ibm-websphere-liberty-1.0.1.tgz
```

9. Update the package repository using the IBM Cloud Private management console. You should be able to see the Helm chart deployed as shown in Figure 2-21.

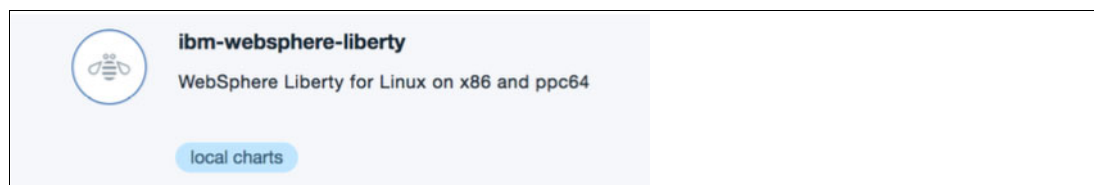


Figure 2-21 Catalog showing IBM Liberty chart

2.8 Changing certificates post installation

You can replace certificates for Image Manager (Docker Registry) and Management ingress created by the installer in your IBM Cloud Private environment. Before trying to replace certificate verify that your IBM Cloud Private cluster is running. Complete the following steps to replace certificates.

1. Generate a new certificate key pair for the certificate that you want to replace. The following sample creates an `icp-router` certificate and key pair. See Example 2-14.

Note: You must replace `mycluster.icp` with your own `cluster_CA_domain` that you configured in the `<installation_directory>/cluster/config.yaml` file.

Example 2-14 Generate a new certificate key pair

```
openssl req -x509 -newkey rsa:4096 -sha256 -nodes -keyout icp-router.key -x509
-days 36500 -out icp-router.crt -subj "/C=US/ST=California/L=Los
Angeles/O=Your Domain Inc/CN=mycluster.icp"
```

2. Log in to the boot node as a cluster administrator.

3. Change to the `cluster/cfc-certs/` directory. See Example 2-15.

Example 2-15 Change to the cluster/cfc-certs/ directory

```
cd <installation_directory>/cluster/cfc-certs/
```

4. Back up the original certificate directory or file. The following example (Example 2-16) backs up the router directory which contains the `icp-router.crt` and `icp-router.key`.

Example 2-16 Back up the original certificate directory

```
mv router router.bak
```

5. Replace `router/icp-router.key` and `router/icp-router.crt` with your own key pair and certificate that you generated in step 1. See Example 2-17.

Example 2-17 Replace router/icp-router.key and router/icp-router.crt

```
mkdir ./router  
cp <custom_certificate_directory>/icp-router.crt ./router/icp-router.crt  
cp <custom_certificate_directory>/icp-router.key  
./router/icp-router.key
```

6. Change to the cluster directory: `cd ..`
7. Run the following command to replace the certificate. See Example 2-18.

Example 2-18 Replace the certificate

```
docker run -t --net=host -v $(pwd):/installer/cluster -e LICENSE=accept  
ibmcom/icp-inception-$(uname -m | sed 's/x86_64/amd64/g'):3.1.2-ee  
replace-certificates -v
```



Part 2

IBM Cloud Private system administration tasks

This part includes common scenarios targeted for IBM Cloud Private system administrators.



Backup and restore of an IBM Cloud Private cluster

A backup and restore strategy plays an important role in any IBM Cloud Private cluster, as it defines how the whole environment, its configuration and its applications are restored to a working state in the event of a disaster.

This chapter describes the various options available to Cluster Administrators when thinking about backing up an IBM Cloud Private cluster, ranging from each individual core component to the infrastructure itself. It will discuss the various alternatives and considerations to a backup strategy, and provide several approaches to backing up and restoring the configuration to the same or a different instance of IBM Cloud Private 3.1.2.

This chapter has the following sections:

- ▶ 3.1, “The purpose of backing up a cluster” on page 72
- ▶ 3.2, “Backup versus high availability, disaster recovery, and continuous availability” on page 72
- ▶ 3.3, “Backup options” on page 73
- ▶ 3.4, “Backup and restore strategy” on page 76

3.1 The purpose of backing up a cluster

A properly defined resiliency strategy is a key part in keeping your applications running continuously, and most of the time a backup plan is one of the core features for any form of resiliency effort to provide peace of mind, knowing that your infrastructure and application data is able to be recovered in the event of some failure or disaster that disrupts day to day operations.

For IBM Cloud Private, there are three common options for defining a backup strategy

- ▶ Backup the infrastructure
- ▶ Backup the core components
- ▶ Backup using a preferred third party tool

Due to the vast range of tools available, backing up using third party software is not covered in detail in this book. Instead, this chapter will focus on how to effectively backup and restore the infrastructure and the core components.

In any case, regular backups should be taken so that you're able to restore to a recent point in time. This point in time entirely depends on your own policies, but backups are recommended to be taken immediately after installation and prior to any major changes or upgrades, where the risk of a cluster failure is higher than the normal day to day business operations.

Taking a full infrastructure backup can be quite storage intensive, so this chapter will provide a variety of methods to backup the core IBM Cloud Private components in a flexible manner, using both manual and automated methods to ensure the most efficient process is put in place on a regular basis without worrying about the manual effort involved.

The main purpose of a backup plan is to actually restore it, so in addition to backups, the relevant processes should be in place to test that the backups are actually good backups. This helps to avoid nasty surprises in the event something does go wrong and to provide peace of mind knowing that you can recover from failures.

3.2 Backup versus high availability, disaster recovery, and continuous availability

Whilst backup and restore is the main subject for this chapter, there are other alternatives that should be considered when designing the cluster architecture, and in all cases the architecture should be designed in such a way that gracefully caters for failure. This is applicable on both the infrastructure level and the application level, although the latter is less of a concern as the IBM Cloud Private platform takes care of keeping applications highly available. Before designing the processes to backup a cluster, consider the use of High Availability (HA), Continuous Availability (CA) and Disaster Recovery (DR) methodologies

HA is the ability to withstand failures, by eliminating any single points of failure from the whole system. This typically entails having two or more instances of any component so that the system is able to continue providing services by switching to instance 2, when instance 1 fails. CA is the ability to withstand both unplanned and planned downtime for any given component, providing greater flexibility for maintenance periods, or general component failures. DR on the other hand, is the capability to recover from a failure.

A common example used in DR discussions is the failure of an entire datacenter, and will typically involve all the hardware in one geographic location simultaneously shut down. A good DR strategy will be able to recover from this by recreating the same infrastructure (compute, networking and storage) in another geographical location within a specific Recovery Time Objective (RTO).

These methods can be combined with the backup strategy to ensure you are backing up the platform effectively to allow you to recover from any given failure. When doing the system architecture it is important to consider the backup strategy and also if there is a need for high availability (HA) or disaster recovery (DR). The following specifications must be taken into account: Recovery Point Objective (RPO) and Recovery Time Objective (RTO).

The backup strategy will grant that the system is restored to a given time when the backup was taken and in a case of the disaster, the data created between the time that the backup was taken and the moment of the failure will be lost in most of cases. This should be considered when creating the backup policy

The high availability environment will grant that the servers still responsive including when one or more nodes fail. So other servers will remain online and perform the tasks. When thinking about HA it is frequently referred as the local system redundancy and each of HA environments should be able to handle the full load. The main difference between the HA and backup is that if data gets corrupted in one of the nodes, it will be propagated to the other nodes, with that both nodes will having the failure, and will need the backup to resume to its normal operation.

The disaster recovery (DR) environment is a copy of the production (or running) environment and in case of failure this environment will take over the requests and all operation, when the main system is ready to be put back online the data should be replicated from the DR side to the main system.

3.3 Backup options

When considering the backup strategy, it is recommended to verify the most suitable types of backup to meet the requirements. This section will briefly discuss each of the components within the IBM Cloud Private infrastructure and platform, and if/why each should be backed up.

3.3.1 Infrastructure backups

Infrastructure backups are the most common method to ensure a full copy of an IBM Cloud Private node exists in the event of a failure. Suitable tools for infrastructure backups are usually available with the hypervisor hosting IBM Cloud Private.

At the time of writing, there are 6 different types of IBM Cloud Private nodes; boot, master, proxy, management, Vulnerability Advisor and worker nodes. Each node plays a specific role in the cluster, and each will have a different impact on the backup and restore strategy.

- **Boot nodes:** A boot (or bootstrap) node is used for running installation, configuration, node scaling, and cluster updates. Only one boot node is required for any cluster, and a single boot node can cater for multiple installations of IBM Cloud Private. This node stores the cluster configuration data and is important to at least backup the filesystem so that the original configuration and cluster certificates are able to be reused if necessary. In small clusters, boot nodes are typically combined with the master nodes.

- ▶ **Master nodes:** A master node provides management services and controls the worker nodes in a cluster. Master nodes host processes that are responsible for resource allocation, state maintenance, scheduling and monitoring. Because a high availability (HA) environment contains multiple master nodes, if the leading master node fails, failover logic automatically promotes a different node to the master role. Hosts that can act as the master are called master candidates. The master nodes hosts the most important core components, such as etcd (if etcd is not externalized) and MongoDB and should almost certainly be the main focus for platform level backups.
- ▶ **Proxy nodes:** A proxy node is a node that transmits external request to the services created inside your cluster. These are not considered important to backup, as they are generally easily replaced in any cluster.
- ▶ **Management nodes:** A management node hosts management services such as Istio, monitoring, metering, and logging. The importance of backing up this node entirely depends on how valuable you consider the data stored on it. For example, the logging data (by default) will retain the platform log data for one day, so it may not always make sense to keep infrastructure backups of the data if it will be deleted again shortly after restoration.
- ▶ **Vulnerability Advisor nodes:** A Vulnerability Advisor (VA) node is an optional node that is used for running the Vulnerability Advisor services. The importance of backing up this node entirely depends on how valuable you consider the data stored on it. VA data is a representation of the current cluster, so if a cluster fails and becomes unrecoverable, recovering the data to another cluster is not meaningful for reporting, but may be desired if you require all data to be retained.
- ▶ **Worker nodes:** A worker node is a node that provides a containerized environment for running tasks. It's generally not recommended to backup these nodes as they are easily replaced, but if applications running on the worker nodes depend on a file path on the host file system then an appropriate backup method (or a different storage method) should be considered.
- ▶ **etcd nodes:** An etcd node is an optional node that contains the etcd distributed key store components for the IBM Cloud Private cluster. The etcd components are hosted by the master nodes by default, but in large clusters with more than 100 worker nodes, it's recommended to separate etcd to reduce the volume of network traffic and resource requirements for heavy etcd usage.

Application storage considerations

As a Cluster Administrator, you need to think about the storage methods and how it fits in with the backup and restore strategy. There are a range of storage solutions for IBM Cloud Private that provide different storage capabilities and the most suitable solution(s) should be used to ensure you're able to backup the data from the PersistentVolume mounted to the application and restore it as necessary. This chapter does not provide a specific method to backing up and restoring PersistentVolume data and it's recommended to agree with the application owners on how to ensure the application is resilient to failures.

3.3.2 Platform backups

There are several components of the IBM Cloud Private platform that require persisted data to be backed up and retained in order restore the same working configuration to a fresh installation, either on the same infrastructure (local) or different infrastructure entirely (remote). In all cases, the backup data should be stored externally to the cluster outside the failure range (for example a different datacenter in a multiple datacenter environment), to ensure the data is globally available in the event of a disaster.

Platform components to back up

The following are components requiring a backup.

Cluster installation data

The `cluster` directory used for the IBM Cloud Private installation has several directories that contain core data generated during the cluster installation. It's important to back up this data as it will need to be reused in a situation where a cluster is considered unrecoverable, requiring a fresh installation of IBM Cloud Private on the same infrastructure. The following data should be backed up to a storage medium situated outside of the cluster.

etcd

`etcd` is a distributed key-value store that maintains all of the configuration data for an IBM Cloud Private cluster. Without a fully functioning and healthy `etcd`, the cluster will be inoperable. `etcd` is a core component of IBM Cloud Private that should be backed up at regular intervals so that a cluster can be restored to the most recent state in the event of a failure.

MongoDB

MongoDB is a database used in IBM Cloud Private to store data for the metering service, Helm repository, Helm API server, LDAP configuration and team/user/role information. This is a core component of IBM Cloud Private that should be backed up at regular intervals so that a cluster can be restored to the most recent state in the event of a failure.

MariaDB

MariaDB is a database used to store OpenIDConnect (OIDC) data used for the authentication to IBM Cloud Private. By default IBM Cloud Private refreshes the user access tokens every 12 hours, so the data in MariaDB is transient and therefore an optional component to back up. It's worth noting that whilst the content of the MariaDB database is not essential, it is worth backing up at least once to store the database structure, in the event that it becomes corrupt and the authentication modules in IBM Cloud Private cannot authenticate users.

Private image registry

The private image registry is a Docker image registry used to store all the images for the IBM Cloud Private platform. It contains all the system images and user images segregated by repositories. It is important that the image registry data is backed up to ensure that both the platform and user deployed workloads can reliably pull the same images regardless of whether the workloads are running on the same or a restored cluster. In a Highly Available configuration, the image repository is mounted on shared storage, so does not require a backup unless the cluster administrator wishes to do so.

Helm repository

The Helm repository stores all the Helm charts uploaded to the IBM Cloud Private catalog. This should be backed up to ensure the same Helm charts are available for deployment from the catalog.

Elasticsearch logging data

In IBM Cloud Private, an Elasticsearch cluster consists of Elasticsearch, Logstash and Kibana and is used to collect and process all the log data generated from all containers running on the platform. By default, the collected log data is stored for 24 hours, after which it is removed from the system and no longer available for querying. Whilst the Elasticsearch cluster itself is a core component and fundamental to cluster administrators and users that wish to view log data for their workloads, the log data stored is not essential to maintaining a functioning cluster, therefore it is optional to back up the logging data.

The Vulnerability Advisor and Mutation Advisor management services use Elasticsearch to store data, therefore in a cluster configuration where the Vulnerability Advisor and Mutation Advisor management services are enabled (and the cluster administrator deems the data valuable) then it is advised to back up the Elasticsearch log data.

Monitoring data

In IBM Cloud Private, the monitoring stack includes Alert Manager, Prometheus, and Grafana. These components provide the capabilities to expose metrics about the whole cluster and the applications running within it, trigger alerts based on data analysis and provide a graphical view of the collected metrics to the users. The monitoring stack is deployed, by default, with no persistent storage and therefore does not require backing up. After installation, if persistent storage is added to each component then backing up the Persistent Volume data is necessary to ensure any changes made can be restored.

Vulnerability Advisor

Vulnerability Advisor (VA) is an optional management service that actively scans the images used in an IBM Cloud Private cluster and identifies security risks in these images. The reporting data for VA is stored in Kafka and Elasticsearch. Therefore, to enable a time-based analysis of the cluster image security, the VA components should be backed up.

This means that the logging data also needs to be backed up to ensure VA data is available upon restoring VA functionality. The scans and reports are generated based on the current environment, so restoring this data to a new cluster means that the restored data is no longer meaningful, and depending on the requirements for VA reporting, it may not be necessary to restore the backup data at all.

Mutation Advisor

Mutation Advisor (MA) is an optional management service (installed with Vulnerability Advisor) that actively scans running containers for any changes in system files, configuration files, content files, or OS process within the container. Time-based reports are generated and stored in the same way as Vulnerability Advisor, and therefore the same conditions apply.

Application persistent volume data

Persistent volume data mounted by user deployed workloads can be hosted on a variety of storage solutions, and cluster administrators should ensure that the right tools/methods are used to backup the relevant storage platform accordingly. In a situation where a cluster becomes unrecoverable, and persistent volumes need to be recreated on a different cluster, it is the Cluster Administrators (or application owners, whichever is applicable) responsibility to know the data restoration process for the implemented storage solution(s).

3.4 Backup and restore strategy

The IBM Cloud Private backup and restore process described in this section is based on a best-effort basis. There are many different combinations of cluster configurations and user customizations that may impact how the whole infrastructure or individual platform components should be backed up, as well as different user preferences on exactly what data is considered 'meaningful' to restore.

Figure 3-1 shows each of the components to backup as mentioned earlier and their respective host.



Figure 3-1 Components to backup

The general strategy for successful IBM Cloud Private backup cycles is shown in Figure 3-2. It's a typical iterative approach during operations where periodic backups are taken at either regular intervals (for example daily) or before maintenance.

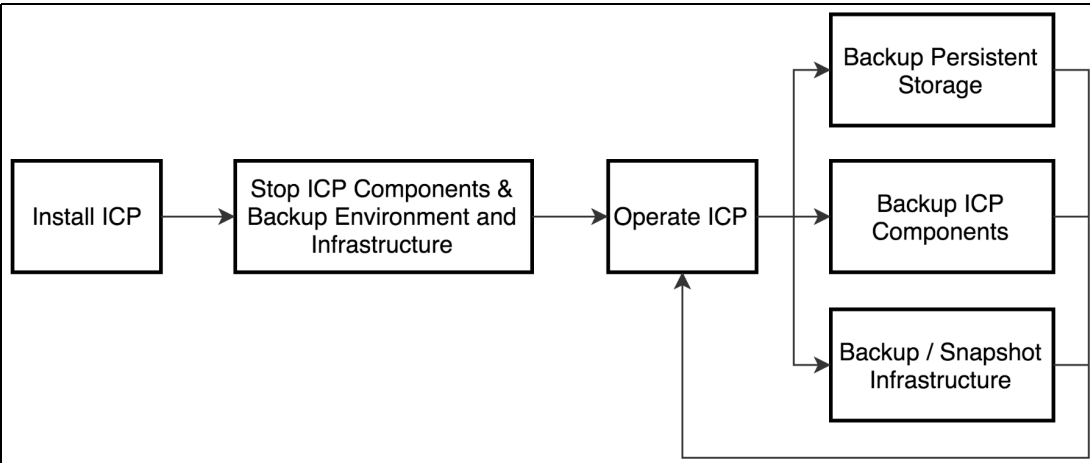


Figure 3-2 General strategy for IBM Cloud Private backup cycles

The backup process needs to be carefully planned to minimise disruption to end users. The following sections will explain the use of several options for backing up and restoring both infrastructure and platform components to allow cluster administrators to apply the most efficient process for their environment.

3.4.1 Infrastructure backup process

It is recommended to take an infrastructure level snapshot immediately after a successful installation of IBM Cloud Private, where there are no user deployed workloads running. This ensures that a clean cluster installation is always available in the event of failure and to reduce the time required to restore a working cluster.

Stopping an IBM Cloud Private node

To *gracefully* stop an IBM Cloud Private node the recommended approach is to stop kubelet and docker on the node first, then shut down the node. This ensures a clean shut down for Docker. For worker nodes containing user deployments that should be kept available, it may be necessary to drain the node first, using `kubectl drain <node>`. It's important to stop the Kubelet process prior to stopping Docker, because Kubelet will continue to run, attempting to access the containers that were previously running on the host:

1. First, stop Kubelet:

```
sudo systemctl stop kubelet
```

2. With Kubelet stopped, stop Docker:

```
sudo systemctl stop docker
```

3. Stopping Docker may take some time. When Docker is stopped (dockerd should no longer be running), shut down the node:

```
shutdown now
```

More information about taking nodes offline can be found in the IBM Knowledge Center:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_cluster/node_maintenance.html

Hot backups of an IBM Cloud Private cluster

Depending on the virtualization platform hosting the cluster nodes, it is possible to take a running (hot) backup of each IBM Cloud Private node and later use it for restoration. As hot backups can be taken while nodes are running, it is recommended to backup the nodes simultaneously, if possible, so that the cluster can be restored in a unified state in the event of failure. Hot backups are generally seen as a short term point-in-time snapshot and should not be used as the sole method of backing up cluster nodes.

Cold backups of an IBM Cloud Private cluster

When taking a cold backup of an IBM Cloud Private topology, shutting down the whole cluster at the same time is not required, but it's important to consider the order in which nodes are shut down. The master nodes contain core processes such as etcd, the Kubernetes api-server, controller manager and scheduler that will attempt to recover pods that have 'failed' due to the host becoming unavailable.

This recovery effort is normal behavior for a running system, but is not considered an organized state that provides a stable backup of the cluster. Therefore, stopping the master nodes first will provide the most stable backup to recover from.

The most stable approach, when taking cold backups of an IBM Cloud Private cluster, is to shut down all master nodes (and etcd nodes, if they are separate from the masters) at the same time, take a backup, then start up the node again. This ensures that all the core components are backed up in exactly the same state. However, the drawback to this approach is that the cluster is temporarily offline while backups are taken, and therefore is not suitable for production installation beyond the point of initial installation.

The recommended time to use this approach is immediately after installation with a stable cluster, so that it's possible to quickly revert back to a fresh installation if required. Generally, backups should be taken of all nodes after installation but proxy and worker nodes are easily replaced and only need backups if the workload data resides on the node itself. Management and Vulnerability Advisor nodes are also easily replaced, but many of the management services write data directly to the host filesystem, so at a minimum the management node filesystems should have a backup.

In a HA installation, it is possible that the cluster can be kept in a running state by taking node backups one by one. This ensures that the cluster is still available for end users, and also provides a stable state for etcd. It's crucial that etcd is kept in a stable state and still able to serve requests, as without a healthy etcd cluster, neither Kubernetes or IBM Cloud Private will function correctly. In topologies where etcd is separated from the master nodes, backups must also be in a similar fashion to ensure etcd is always kept in a stable state.

However, there is a caveat to this process; if too much time has passed in between taking a cold backup of each node, the risk of the etcd database becoming too inconsistent is increased. Furthermore, in a very active cluster, there may have been too many write operations performed on a particular etcd instance, which means etcd will not be able to properly recover.

The cluster administrator needs to consider the use of an active-active style architecture, as discussed earlier, if they require as close to zero-downtime as possible for running applications that rely on the cluster (or management) services on IBM Cloud Private (such as the api-server, or Istio). In most cases, workloads running on worker nodes (using the IBM Cloud Private proxy nodes for access to the applications) are not affected by master node downtime unless there is an event on the application host node that requires attention from the core IBM Cloud Private components whilst the master nodes are offline.

Based on the 1.3, “IBM Cloud Private architecture” on page 10, the following nodes contain core components that persist data and should be backed up regularly

- ▶ etcd/master
- ▶ management (optional but recommended)
- ▶ Vulnerability Advisor (optional but recommended)

The other nodes not mentioned in the list above (proxy and worker nodes) are generally replaceable without consequence to the health of the cluster. Information on replacing various IBM Cloud Private nodes can be found in the IBM Knowledge Center documentation:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/installing/add_node.html

3.4.2 Infrastructure restore process

Care should be taken when restoring IBM Cloud Private nodes. It's important to ensure that the backup process is followed carefully to maximise the chance of a successful cluster restore and attempting to restore a bad backup will almost certainly result in an unstable cluster. The etcd component carries the greatest weight to keep Kubernetes running and should always be validated immediately after restoration to validate the cluster integrity.

There are several other crucial components that also require quorum, such as MongoDB and MariaDB, but as they are Kubernetes resources it is always possible to recover it (assuming the host filesystem is intact). If etcd is permanently failed, so is the whole cluster, so the main focus in this section is retaining a healthy etcd cluster. The key thing to understand when restoring infrastructure backups is that etcd will rebuild its cluster state using the latest available data on the current “leader”, and the elected leader will have the highest raft index.

When the etcd/master nodes are brought back online, the etcd instances will continue to operate where they left off at the point in time where the backup was taken, requesting new elections and updating the current raft term. If any other etcd members making requests to the cluster (becoming a “follower”) have a different raft index that the leader does not recognise (for example a different copy of the data) and an inconsistent raft term then it will no longer be a member of the cluster.

Consider the following example: In a HA cluster with 3 masters, a cold backup is taken sequentially starting from master 1, then 2, then 3. Each master is powered off (using the recommended approach in the previous sections), backed up, then powered on again. During this time period, master 2 and 3 are still running, and the cluster is still fully operational for end users. When restoring, the same order is used so master 1 is restored first, then 2, then 3.

The issue here is that each backup for the master nodes is taken at a completely different time, with master 3 being the last. This means that master 3 will have different data when restored, and when brought online last it will attempt to communicate with the existing cluster. Therefore, it's essential to ensure that in the event of a cluster failure, and all 3 masters need to be restored from a backup, that the order of restoration is the *reverse* of the order of backup. This process applies to restoring both hot and cold backups.

Verifying a restored backup

Restoring a cluster entirely depends on the hosting virtualization platform. The examples in this section uses VMWare to host the IBM Cloud Private installation, and therefore uses the Virtual Machine (VM) snapshot capabilities provided with the platform. Note that a VMWare snapshot is not a complete backup mechanism and is used here to demonstrate the practical application of the backup methods described in this chapter.

Using the previous example of a HA cluster, master nodes 1, 2, and 3 were backed up using VMWare snapshots while the machines were offline (cold snapshot) in sequence starting from master 1. Nginx workloads were deployed to simulate user deployments so that the cluster state in etcd was changed. To simulate a disaster, masters 1 and 2 were destroyed. At this point etcd is now running as a single node cluster, in read-only mode and cannot accept any write requests from Kubernetes.

The CoreOS documentation (found at <https://coreos.com/etcd/docs/latest/op-guide/failures.html>) provides a good explanation about different etcd failure scenarios and what it can or can't tolerate. In this situation, quorum is lost and as you cannot currently replace IBM Cloud Private master nodes using different IP addresses, you cannot recover. Therefore, the snapshots must be used to deploy masters 1 and 2, and restore all 3 masters to a previously working state.

Restore all 3 masters to a previous snapshot, power on the nodes and allow some time for all the containers to start. To ensure the etcd cluster has started successfully and the cluster is healthy, etcd provides a useful command line utility to query the cluster status, which can be downloaded to your local machine or ran from the etcd container itself. To get the cluster health status, complete the following steps from the IBM Cloud Private boot node (or whatever system has cluster access with kubectl):

1. Run `kubectl -n kube-system get pods | grep k8s-etcd` to retrieve the etc pod name (the format is `k8s-etcd-<node-ip>`):

```
[root@icp-ha-boot ~]# kubectl -n kube-system get pods -o name | grep k8s-etcd
k8s-etcd-172.24.19.201
k8s-etcd-172.24.19.202
k8s-etcd-172.24.19.203
```

2. Use any one of the pod names returned to execute an `etcdctl cluster-health` command, using one of the master node IP addresses in the endpoint parameter:

```
[root@icp-ha-boot ~]# kubectl -n kube-system exec k8s-etcd-172.24.19.201 -- sh
-c "export ETCDCTL_API=2; etcdctl --cert-file /etc/cfc/conf/etcd/client.pem
--key-file /etc/cfc/conf/etcd/client-key.pem --ca-file
/etc/cfc/conf/etcd/ca.pem --endpoints https://172.24.19.201:4001
cluster-health"
```

```

member 8a2d3ec6df19666f is healthy: got healthy result from
https://172.24.19.201:4001
member ae708d12aa012fdc is healthy: got healthy result from
https://172.24.19.202:4001
member dd2afb46d331fdd2 is healthy: got healthy result from
https://172.24.19.203:4001
cluster is healthy

```

This outputs the cluster state for all etcd endpoints, and will either state `cluster is healthy`, or `cluster is unhealthy` as a final status. If the output is `cluster is healthy`, then restoring etcd was successful. If the output is `cluster is unhealthy`, then further investigation is needed.

It's also important to check the other core components, such as MongoDB and MariaDB to ensure they are running correctly. See the troubleshooting section for more information about verifying a successful IBM Cloud Private cluster state.

Further testing

This section provides some real-world testing of hot and cold backups where a cluster is under heavier stress from continuous workload deployment. The testing in this section was conducted in a VMWare lab environment using VMWare snapshot capabilities and does not represent a real customer deployment, it is only intended for simulation purposes.

To test the backup process resiliency for master nodes, the following test plan was used, as shown in Figure 3-3.

VMWare Snapshot Verification Procedure			
#	Action	Description	Result
1	Cluster Installation	Initial IBM Cloud Private Installation	cluster is healthy
2	Cold snapshot 1	Master nodes powered off, snapshot taken and nodes started	cluster is healthy
3	Deploy workloads	Create namespace 'ns-1' and deploy an Nginx deployment with 2 replicas	cluster is healthy
4	Hot snapshot 1	Snapshot manually taken sequentially of running master nodes from vSphere console	cluster is healthy
5	Deploy workloads	Create two namespaces; 'ns-2' and 'ns-3' and deploy an Nginx deployment with 2 replicas	cluster is healthy
6	Hot snapshot 2	Snapshot manually taken sequentially of running master nodes from vSphere console	cluster is healthy
7	Restore Cold snapshot 1	Restore master nodes to Cold snapshot 1	cluster is healthy
8	Restore Hot snapshot 2	Restore master nodes to Hot snapshot 2	cluster is healthy
9	Restore Hot snapshot 1 on master 1	Restore master 1 to Hot snapshot 1	cluster is not healthy. Master 1 etcd is not running
10	Restore Hot snapshot 1 on master 2 and 3	Restore masters 2 and 3 to Hot snapshot 1	cluster is healthy
11	Cold snapshot 2	Master nodes powered off, snapshot taken and nodes started sequentially whilst keeping cluster available	cluster is healthy
12	Deploy workloads	Rereate two namespaces; 'ns-2' and 'ns-3' and deploy an Nginx deployment with 2 replicas in each namespace	cluster is healthy
13	LDAP + Teams/Users Configured	Create 3 teams; 'team-1', 'team-2' and 'team-3' assigned to the respectively named namespace	cluster is healthy
14	Deploy workloads	Deploy an Nginx deployment with 10 replicas to each namespace	cluster is healthy
15	Hot snapshot 3	Snapshot manually taken sequentially of running master nodes from vSphere console	cluster is healthy
16	Install Vulnerability Advisor	Install Vulnerability Advisor	cluster is healthy
17	Restore Cold snapshot 2	Restore cluster back to initial state sequentially from master 1..3	cluster is not healthy. Master 3 etcd is not running
18	Restore Cold snapshot 2	Restore cluster back to initial state sequentially from master 3..1	cluster is healthy (LDAP & VA resources no longer deployed)
19	Deploy workloads	Automated workload deployment script used to generate workloads every 10 seconds	cluster is healthy
20	Hot snapshot 4	Snapshot manually taken sequentially of running master nodes from vSphere console	cluster is healthy
21	Cold snapshot 3	Master nodes powered off, snapshot taken and nodes started sequentially whilst keeping cluster available	cluster is healthy
22	Restore Hot snapshot 4	Restore cluster back to initial state sequentially from master 3..1	cluster is healthy
23	Restore Cold snapshot 3	Restore cluster back to initial state sequentially from master 3..1	cluster is healthy

Figure 3-3 VMWare snapshot simulation action list

This test plan took “hot” and “cold” snapshots of nodes at different times, In items 1 to 18, there was a minimal workload in the cluster, but this does not simulate an active cluster. At item 19, heavy workload was applied to the cluster, by deploying a random number of replicas of Nginx containers to different namespaces every 10 seconds.

Prior to taking backups, the etcd cluster was queried to retrieve information about the current database size, current raft term and current raft index. Below shows the output of an **etcdctl endpoint status** command.

```
[root@icp-ha-boot ~]# docker run --entrypoint=etcdctl -e ETCDCTL_API=3 -v
/etc/cfc/conf/etcd:/certs -v /var/lib/etcd:/var/lib/etcd -v /tmp:/data
mycluster.icp:8500/ibmcom/etcd:3.2.24 --cert=/certs/client.pem
--key=/certs/client-key.pem --cacert=/certs/ca.pem --endpoints
https://172.24.19.201:4001,https://172.24.19.202:4001,https://172.24.19.203:4001
-w table endpoint status
```

ENDPOINT	ID	VERSION	DB SIZE	IS LEADER	RAFT TERM	RAFT INDEX
https://172.24.19.201:4001	8a2d3ec6df19666f	3.2.24	30 MB	false	1617	1879006
https://172.24.19.202:4001	ae708d12aa012fdc	3.2.24	31 MB	true	1617	1879006
https://172.24.19.203:4001	dd2afb46d331fdd2	3.2.24	31 MB	false	1617	1879006

After 30 seconds, the same query was run to analyze the changes in data.

ENDPOINT	ID	VERSION	DB SIZE	IS LEADER	RAFT TERM	RAFT INDEX
https://172.24.19.201:4001	8a2d3ec6df19666f	3.2.24	30 MB	false	1617	1879291
https://172.24.19.202:4001	ae708d12aa012fdc	3.2.24	31 MB	true	1617	1879292
https://172.24.19.203:4001	dd2afb46d331fdd2	3.2.24	31 MB	false	1617	1879293

The leader and raft term is still the same, so there is no issues around frequent elections, but the raft index has changed, which represents changes in the data.

Master 1 was taken offline, snapshot, then left offline for some time to simulate maintenance. After 10 minutes, master 1 was brought back online, and the process repeated for master 2 and 3. Once all masters were available and running, the etcd status now shows the following.

ENDPOINT	ID	VERSION	DB SIZE	IS LEADER	RAFT TERM	RAFT INDEX
https://172.24.19.201:4001	8a2d3ec6df19666f	3.2.24	43 MB	true	1619	1931389
https://172.24.19.202:4001	ae708d12aa012fdc	3.2.24	43 MB	false	1619	1931391
https://172.24.19.203:4001	dd2afb46d331fdd2	3.2.24	43 MB	false	1619	1931395

This maintenance simulation process took place over 47 minutes, providing a realistic test for restoring the snapshots to a point in time before maintenance took place on the masters.

Prior to restoring the nodes, the endpoint status resembles the following.

ENDPOINT	ID	VERSION	DB SIZE	IS LEADER	RAFT TERM	RAFT INDEX
https://172.24.19.201:4001	8a2d3ec6df19666f	3.2.24	46 MB	true	1619	1945597
https://172.24.19.202:4001	ae708d12aa012fdc	3.2.24	46 MB	false	1619	1945597
https://172.24.19.203:4001	dd2afb46d331fdd2	3.2.24	46 MB	false	1619	1945597

To verify that restoring the masters in the original sequential order yields an inconsistent cluster, the masters were brought online starting from master 1. The endpoint status shows the following.

Failed to get the status of endpoint https://172.24.19.203:4001 (context deadline exceeded)

ENDPOINT	ID	VERSION	DB SIZE	IS LEADER	RAFT TERM	RAFT INDEX
https://172.24.19.201:4001	8a2d3ec6df19666f	3.2.24	44 MB	false	1638	1927603
https://172.24.19.202:4001	ae708d12aa012fdc	3.2.24	43 MB	true	1638	1927603

This test was repeated 5 times, of which every test produced consistent results; the third master could not join the cluster.

Upon restoring the nodes in reverse order (from master 3 to 1), the endpoint status resembles the following.

ENDPOINT	ID	VERSION	DB SIZE	IS LEADER	RAFT TERM	RAFT INDEX
https://172.24.19.201:4001	8a2d3ec6df19666f	3.2.24	43 MB	false	1661	1928697
https://172.24.19.202:4001	ae708d12aa012fdc	3.2.24	43 MB	false	1661	1928697
https://172.24.19.203:4001	dd2afb46d331fdd2	3.2.24	43 MB	true	1661	1928697

This test was repeated 5 times, of which every test produced consistent results; the etcd cluster recovered successfully.

Other combinations were also tested. The below tables show the results per varying combinations of restoration order for both hot and cold snapshots.

Cold Snapshots				Hot Snapshots			
Test #	# of Tests	Restore Order	Result	Test #	# of Tests	Order	Result
1	3	2	2/3 master 3 was	1	3	2	healthy cluster
		3	unavailable. 1/3			3	
		1	healthy cluster			1	
2	3	2	3/3 master 3 was unavailable	2	3	2	healthy cluster
		1				1	
		3				3	
3	3	1	healthy cluster	3	3	1	healthy cluster
		3				3	
		2				2	
4	3	3	healthy cluster	4	3	3	healthy cluster
		1				1	
		2				2	
				5	3	1	healthy cluster
						2	
						3	

Figure 3-4 Tables showing simple test results for hot and cold restores

The results above reiterate the need to restore cold backups in reverse order. Restoring backups in any other order yielded unpredictable results, and in some cases, an unstable cluster. Hot backups were much more effective at restoring the original cluster state a lot quicker (less time to start up services as memory was preserved). Hot snapshots were taken within a much closer time frame, which means etcd had a very high chance of recovering from minor inconsistencies between instances.

In most cases, hot snapshots are not used as the sole backup method and should be used appropriately as documented by the platform provider. However, based on the above results they can be reliable to quickly recover from fatal changes to the cluster.

3.4.3 Platform backup process

As described in section 3.2, the recommended approach is to regularly backup the individual components of an IBM Cloud Private cluster. More frequent backups mean that a cluster administrator always has a recent backup of all the core components to successfully restore a cluster in the event of a failure, without reverting to a previous infrastructure backup. This section will describe the steps required to back up the various components to external storage using the tools available in IBM Cloud Private.

It's advised that the core components are backed up in the following order:

1. etcd
2. MongoDB
3. MariaDB (Optional)

This is not a *strict* backup order, just an *advised* order. MongoDB persists Team and User data, and etcd persists the relevant Role Base Access Control (RBAC) Kubernetes resources associated to those Teams and users. If you were to back up etcd first, add a new Team with Users, then back up MongoDB, you would end up with out of sync data as etcd would not contain the data to restore the RBAC Kubernetes resources for the new Team.

Alternatively, if you backup MongoDB first, add a new Team and Users, then back up etcd, data would still be out of sync as MongoDB would not contain the Team and User data upon restore. Based on this, the backup order is entirely the choice of the Cluster Administrator, but following the approach in the infrastructure backup sections this chapter takes a backup of etcd first and everything else after. Ideally, both etcd and MongoDB should be backed up within a close time frame, at a period of time where cluster activity is low.

The non-core components can be backed up in any order. When restoring the Vulnerability Advisor, it's recommended to restore full functionality to the Elasticsearch stack first, as Vulnerability Advisor and Mutation Advisor both rely on a healthy Elasticsearch to store and retrieve data.

The recommended approach is to regularly back up these core components by using a CronJob. This creates an instance of a Kubernetes job on a time-based schedule, allowing users to execute jobs at any time using the Unix crontab format. In IBM Cloud Private, CronJobs can be used to run etcd and MongoDB backups periodically, using a PersistentVolume so the data can be stored on an external storage provider. This has the major benefit of keeping the latest backups, with no cluster down time and no heavy storage implications, which would normally be the case for full infrastructure backups.

For all of the components that require a PersistentVolume (PV) and PersistentVolumeClaim (PVC), the same PV for an NFS server is used. If the backups should be segregated on different volumes, create additional PVs and PVCs for each component.

Creating the core-backup PersistentVolume and PersistentVolumeClaim

Create a new PersistentVolume (PV) and PersistentVolumeClaim (PVC) to use to store the backups. Use Example 3-1 as a template for a new NFS PV, and Example 3-2 as a template for an NFS PVC.

Example 3-1 YAML definition for core-backup PV

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: core-backup
```

```
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 200Gi
  nfs:
    path: /backup
    server: 198.0.0.1
  persistentVolumeReclaimPolicy: Retain
```

Example 3-2 YAML definition for core-backup PV

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: core-backup
  namespace: kube-system
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 200Gi
  storageClassName: ""
```

The use of `accessModes: ReadWriteMany` here allows multiple containers to write to the volume simultaneously. Some storage types do not provide this capability and in such scenarios, one PV and PVC should be created per component. The storage capacity of 200Gi is used as an example and the real-world value depends on the size of the backups, and the duration the backup data is kept (which typically depends on internal customer requirements/policies). As this is reused among all components that require a PV and PVC, the directory structure is the following:

```
backup
... etcd
... logging
... mariadb
... mongodb
```

The volumes will be mounted to the appropriate directories on the NFS server.

etcd and MongoDB CronJob

This CronJob creates a job every day at 23:30pm to deploy two containers: one to snapshot etcd and one to export the current MongoDB database content. Each container has the same shared storage mounted (NFS in this example).

The easiest way to determine how much storage etcd and MongoDB require is to manually run a backup of etcd and MongoDB using `kubectl` and check the backup size. From this, the storage capacity required for the PVs can be calculated from the CronJob schedule. “Manual etcd and MongoDB backups” on page 89 provides more information on manual backups.

Complete the following steps to run a manual backup:

1. Copy the CronJob YAML definition in Example 3-3 on page 86 to your local machine and save to a file named `core-backup_cronjob.yaml`. For security and isolation from the worker nodes, the pods created in this job are scheduled to a master node.

Example 3-3 core-backup_cronjob.yaml

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: core-backup
  namespace: kube-system
spec:
  schedule: "30 23 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          affinity:
            nodeAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
                nodeSelectorTerms:
                  - matchExpressions:
                      - key: beta.kubernetes.io/arch
                        operator: In
                        values:
                          - amd64
                          - ppc64le
                          - s390x
                  - key: master
                        operator: In
                        values:
                          - "true"
          containers:
            - name: etcdbackup
              image: ibmcom/etcd:3.2.24
              command: ["bin/sh", "-c", "cat /certs/etcd-cert > client.pem; cat
/certs/etcd-key > client-key.pem; cat /certs/etcd-ca > ca.pem; etcdctl
--cert=client.pem --key=client-key.pem --cacert=ca.pem --endpoints
https://$(ENDPOINT):4001 snapshot save /backup/etcd/etcd.$(date
+%Y-%m-%d_%H:%M:%S).db"]
              env:
                - name: ENDPOINT
                  value: "#ETCD-ENDPOINT"
                - name: ETCDCTL_API
                  value: "3"
              volumeMounts:
                - mountPath: "/backup"
                  name: backup
                - mountPath: "/certs"
                  name: etcd-certs
            - name: mongodbdump
              image: ibmcom/icp-mongodb:4.0.5
              command: ["bin/bash", "-c", "cat /certs/tls.crt certs/tls.key >
mongo.pem; export PRIMARY=$(mongo --host rs0/mongodb:27017 --username admin
--password $ADMIN_PASSWORD --authenticationDatabase admin --ssl --sslCAFile
/ca/tls.crt --sslPEMKeyFile mongo.pem --eval=\"db.isMaster()['primary']\" |
grep ^icp-mongodb-); mongodump --host $PRIMARY --username admin --password
$ADMIN_PASSWORD --authenticationDatabase admin --ssl --sslCAFile /ca/tls.crt
--sslPEMKeyFile mongo.pem --oplog --verbose --gzip
--archive=/backup/mongodb/mongodb-backup-$(date +%Y-%m-%d_%H:%M:%S).gz"]
```

```

env:
  - name: ADMIN_PASSWORD
    valueFrom:
      secretKeyRef:
        key: password
        name: icp-mongodb-admin
volumeMounts:
  - mountPath: "/backup"
    name: backup
  - mountPath: "/ca"
    name: cluster-ca
  - mountPath: "/certs"
    name: mongodb-certs
tolerations:
  - effect: NoSchedule
    key: dedicated
    operator: Exists
volumes:
  - name: backup
    persistentVolumeClaim:
      claimName: core-backup
  - name: cluster-ca
    secret:
      secretName: cluster-ca-cert
  - name: mongodb-certs
    secret:
      secretName: icp-mongodb-client-cert
  - name: etcd-certs
    secret:
      secretName: etcd-secret
restartPolicy: Never

```

This file can also be downloaded from the following location:

https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch3-Backup-and-Restore/Backup/core-backup_cronjob.yaml

2. The ETCD-ENDPOINT environment variable for the etcd container should be changed to reflect an actual etcd IP address. Modify the containers section in Example 3-4 and replace #ETCD-ENDPOINT with an appropriate etcd IP address.

Example 3-4 etcd container definition

```

containers:
  - name: etcdbackup
    image: ibmcom/etcd:3.2.24
    command: ["/bin/sh", "-c", "cat /certs/etcd-cert > client.pem; cat /certs/etcd-key > client-key.pem; cat /certs/etcd-ca > ca.pem; etcdctl --cert=client.pem --key=client-key.pem --cacert=ca.pem --endpoints https://$(ENDPOINT):4001 snapshot save /data/etcd.$(date +%Y-%m-%d_%H:%M:%S).db"]
    env:
      - name: ENDPOINT
        value: "#ETCD-ENDPOINT"

```

3. Alternatively, sed can be used to easily do this, by executing `sed -i -e s/#ETCD-ENDPOINT/<etcd-endpoint>/g core-backup_cronjob.yaml` replacing <etcd-endpoint> with your own.

CronJob schedules use the Unix crontab format, so adjust the `spec.schedule` value as required. For more information and guidance in generating a valid crontab schedule, see <https://crontab.guru/>. As an example, to run a job every 10 minutes, the schedule value would be `*/10 * * * *`, for every hour `0 */1 * * *`, and every Friday at 9pm `0 21 * * FRI`.

4. The `ibmcom/etcd:3.2.24` image is the default etcd image for IBM Cloud Private 3.1.2. If the environment does not have internet access, replace `ibmcom` with the private image registry local to the cluster, for example, `mycluster.icp:8500/ibmcom/etcd:3.2.24`.
5. Create the CronJob using `kubectl create -f core-backup_cronjob.yaml`. The CronJob should be visible in the **Workloads** → **Jobs** → **CronJobs** tab in IBM Cloud Private (Figure 3-5).

Name	Namespace	Schedule	Suspend	Active	Created
core-backup	kube-system	*/10 * * * *	False	0	4 hours ago
logging-elk-elasticsearch-curator	kube-system	30 23 * * *	False	0	4 days ago

Figure 3-5 Running CronJob

6. All generated jobs (at the defined schedule) should be visible in the **Workloads** → **Jobs** → **BatchJobs** tab (Figure 3-6).

Name	Namespace	Completions	Parallelism	Successful	Created
core-backup-1551137400	kube-system	1	1	1	2 minutes ago
logging-elk-elasticsearch-curator-1551137400	kube-system	1	1	1	2 minutes ago
core-backup-1551136800	kube-system	1	1	1	12 minutes ago
core-backup-1551136200	kube-system	1	1	1	22 minutes ago

Figure 3-6 Running jobs

7. If a job is completed successfully, it will show a 1 in the **Successful** column. If the job was not successful, troubleshoot it using the guidance provided in the troubleshooting chapter. Verify that the backups exist on the storage provider (external NFS server in this example) as demonstrated in Example 3-5.

Example 3-5 Tree output of NFS backup directory

```
[root@icp-nfs backup]# tree --du -h
.
... [87M]  etcd
.   ... [ 29M]  etcd.2019-02-26_02:50:08.db
.   ... [ 29M]  etcd.2019-02-26_03:00:02.db
.   ... [ 29M]  etcd.2019-02-26_03:10:05.db
... [2.8M]  mongodb
.   ... [966K]  mongodb-backup-2019-02-26_02:50:09.gz
.   ... [967K]  mongodb-backup-2019-02-26_03:00:03.gz
.   ... [967K]  mongodb-backup-2019-02-26_03:10:06.gz
```

89M used in 2 directories, 6 files

This output also provides a baseline to calculate the necessary storage requirements. For example, if etcd data should be kept for 30 days and each backup is about ~30 Mb, the total needed for etcd backups is 900 Mb. Similarly for MongoDB, ~30 Mb is required for 30 days worth of backup data. Naturally, the backup size is likely to grow as the cluster is used over time, so ensure the backup sizes are monitored periodically and adjust the PV size as required.

Manual etcd and MongoDB backups

Manual backups of etcd and MongoDB can be taken in a variety of ways. This section will cover how to do this using Docker, kubectl, and Kubernetes jobs.

Backup etcd using Docker

Perform the following steps on an IBM Cloud Private master node to backup etcd using Docker. If etcd is running on dedicated nodes, perform the steps on that node.

1. Find the k8s-etcd container (Linux grep and awk tools are used to simplify the output):

```
[root@icp-ha-master-1 ~]# docker ps | grep k8s_etcd_k8s-etcd | awk
'FNR==1{print $1}'
af39b9392d85
```

2. Execute the **snapshot** command:

```
[root@icp-ha-master-1 ~]# docker run --entrypoint=etcdctl -e ETCDCTL_API=3 -v
/tmp/backup:/backup -v /etc/cfc/conf/etcd:/certs -v /var/lib/etcd:/var/lib/etcd
mycluster.icp:8500/ibmcom/etcd:3.2.24 --cert /certs/client.pem --key
/certs/client-key.pem --cacert /certs/ca.pem --endpoints
https://172.24.19.201:4001 snapshot save /backup/snapshot.db
Snapshot saved at /backup/snapshot.db
```

3. Verify the backup exists:

```
[root@icp-ha-master-1 ~]# ls -sh /tmp/backup/
total 30M
30M snapshot.db
```

4. Copy the snapshot to a backup storage location outside of the cluster.

Backup etcd using kubectl

Perform the following steps on any machine that has kubectl configured to access to the cluster:

1. Get the etcd pod name:

```
[root@icp-ha-boot ~]# kubectl -n kube-system get pods | grep "k8s-etcd" | awk
'NR==1{print $1}'
k8s-etcd-172.24.19.201
```

2. Run the **snapshot** command:

```
[root@icp-ha-boot ~]# kubectl -n kube-system exec k8s-etcd-172.24.19.201 -- sh
-c "export ETCDCTL_API=3; etcdctl --cert /etc/cfc/conf/etcd/client.pem --key
/etc/cfc/conf/etcd/client-key.pem --cacert /etc/cfc/conf/etcd/ca.pem
--endpoints https://172.24.19.201:4001 snapshot save snapshot.db"
Snapshot saved at snapshot.db
```

3. Copy the **snapshot.db** file from the **etcd** container to the local machine:

```
[root@icp-ha-boot ~]# kubectl cp kube-system/k8s-etcd-172.24.19.201:snapshot.db .
```

4. Verify the backup exists:

```
[root@icp-ha-boot ~]# ls -sh .
total 30M
30M snapshot.db
```

Backup etcd using a Kubernetes job

etcd can be snapshot to a PersistentVolume using a Kubernetes job. For security and isolation from the worker nodes, the pods created in this job are scheduled to a master node. Perform the following steps on any machine that has **kubectl** configured to access to the cluster:

1. Create a PersistentVolume (PV) and PersistentVolumeClaim (PVC), similar to the examples in Example 3-1 on page 84 and Example 3-2 on page 85 to store the etcd snapshot. This job definition uses the same PV and PVC created in those examples.
2. Create the **etcd-backup_job.yaml** file shown in Example 3-6. Alternatively, download the file from the following location:

https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch3-Backup-and-Restore/Backup/etcd-backup_job.yaml

Example 3-6 The etcd-backup_job.yaml file

```
apiVersion: batch/v1
kind: Job
metadata:
  name: etcdbackup
  namespace: kube-system
spec:
  template:
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: beta.kubernetes.io/arch
                    operator: In
                    values:
                      - amd64
                      - ppc64le
                      - s390x
                  - key: master
                    operator: In
```



```

        values:
        - "true"
    containers:
    - name: etcdbackup
      image: ibmcom/etcd:3.2.24
      command: ["/bin/sh","-c","cat /certs/etcd-cert > client.pem; cat
/certs/etcd-key > client-key.pem; cat /certs/etcd-ca > ca.pem; etcdctl
--cert=client.pem --key=client-key.pem --cacert=ca.pem --endpoints
https://$(ENDPOINT):4001 snapshot save /backup/etcd/etcdbackup.$(date
+%Y-%m-%d_%H:%M:%S).db"]
      env:
      - name: ENDPOINT
        value: "#ETCD-ENDPOINT"
      - name: ETCDCCTL_API
        value: "3"
      volumeMounts:
      - mountPath: "/backup"
        name: backup
      - mountPath: "/certs"
        name: etcd-certs
    tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Exists
    volumes:
    - name: backup
      persistentVolumeClaim:
        claimName: core-backup
    - name: etcd-certs
      secret:
        secretName: etcd-secret
      restartPolicy: Never
    backoffLimit: 1

```

The ETCD-ENDPOINT environment variable for the etcd container should be changed to reflect an actual etcd IP address. Modify the containers section in Example 3-7 and replace #ETCD-ENDPOINT with an appropriate etcd IP address.

Example 3-7 etcd container definition

```

containers:
- name: etcdbackup
  image: ibmcom/etcd:3.2.24
  command: ["/bin/sh","-c","cat /certs/etcd-cert > client.pem; cat /certs/etcd-key
> client-key.pem; cat /certs/etcd-ca > ca.pem; etcdctl --cert=client.pem
--key=client-key.pem --cacert=ca.pem --endpoints https://$(ENDPOINT):4001 snapshot
save /data/etcd.$(date +%Y-%m-%d_%H:%M:%S).db"]
  env:
  - name: ENDPOINT
    value: "#ETCD-ENDPOINT"

```

Alternatively, sed can be used to easily do this, by executing `sed -i -e s/#ETCD-ENDPOINT/<etcd-endpoint>/g core-backup_cronjob.yaml` replacing <etcd-endpoint> with your own.

The `ibmcom/etcd:3.2.24` image is the default etcd image for IBM Cloud Private 3.1.2. If the environment does not have internet access, replace `ibmcom` with the private image registry local to the cluster, for example, `mycluster.icp:8500/ibmcom/etcd:3.2.24`.

3. Create the job:

```
[root@icp-ha-boot ~]# kubectl create -f etcd-backup_job.yaml
job.batch "etcdump" created
```

4. Verify that the job was successful:

```
[root@icp-ha-boot ~]# kubectl -n kube-system get jobs etcdump
NAME          DESIRED    SUCCESSFUL  AGE
etcdump       1          1           1m
```

5. Verify that the etcd snapshot exists:

```
[root@icp-nfs etcd]# ls -sh etcdump*
30M etcdump.2019-02-26_04:49:06.db
```

Backup MongoDB using kubectl

Perform the following steps on any machine that has `kubectl` configured to access to the cluster:

1. Get the MongoDB pod name:

```
[root@icp-ha-boot ~]# kubectl -n kube-system get pods -l app=icp-mongodb
-o=jsonpath='{.items[0].metadata.name}'
icp-mongodb-0
```

2. Retrieve the primary replica:

```
[root@icp-ha-boot ~]# kubectl -n kube-system exec icp-mongodb-0 -n kube-system
-- sh -c 'mongo --host rs0/mongodb:27017 --username admin --password
$ADMIN_PASSWORD --authenticationDatabase admin --ssl --sslCAFile
/data/configdb/tls.crt --sslPEMKeyFile /work-dir/mongo.pem
--eval="db.isMaster()[\"primary\"]" | grep ^icp-mongodb-'
icp-mongodb-1.icp-mongodb.kube-system.svc.cluster.local:27017
```

3. Run the `mongodump` command using the primary replica retrieved from step 2:

```
[root@icp-ha-boot ~]# kubectl -n kube-system exec icp-mongodb-0 -- sh -c 'mkdir
-p /work-dir/backup/; mongodump --host
icp-mongodb-1.icp-mongodb.kube-system.svc.cluster.local:27017 --username admin
--password $ADMIN_PASSWORD --authenticationDatabase admin --ssl --sslCAFile
/data/configdb/tls.crt --sslPEMKeyFile /work-dir/mongo.pem --oplog --quiet
--gzip --archive="/work-dir/backup/backup.gz"'
```

4. Copy the `backup.gz` file from the `icp-mongodb-0` container to the local machine:

```
[root@icp-ha-boot ~]# kubectl cp
kube-system/icp-mongodb-0:/work-dir/backup/backup.gz .
```

5. Verify that the backup exists:

```
[root@icp-ha-boot ~]# ls -sh .
total 972K
972K backup.gz
```

Backup MongoDB using a Kubernetes job

MongoDB can be backed up to a PersistentVolume using a Kubernetes job. For security and isolation from the worker nodes, the pods created in this job are scheduled to a master node. Perform the following steps on any machine that has `kubectl` configured to access to the cluster:

1. Create a PersistentVolume (PV) and PersistentVolumeClaim (PVC), similar to the examples in Example 3-1 and Example 3-2 to store the backup file. This job definition uses the same PV and PVC created in those examples.
2. Create the `mongodb-backup_job.yaml` file from Example 3-8 below.

Example 3-8 mongodb-backup_job.yaml

```

apiVersion: batch/v1
kind: Job
metadata:
  name: mongodbdump
  namespace: kube-system
spec:
  template:
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: beta.kubernetes.io/arch
                    operator: In
                    values:
                      - amd64
                      - ppc64le
                      - s390x
                  - key: master
                    operator: In
                    values:
                      - "true"
      containers:
        - name: mongodbdump
          image: ibmcom/icp-mongodb:4.0.5
          command: ["/bin/bash", "-c", "cat /certs/tls.crt certs/tls.key > mongo.pem;
export PRIMARY=$(mongo --host rs0/mongodb:27017 --username admin --password
$ADMIN_PASSWORD --authenticationDatabase admin --ssl --sslCAFile /ca/tls.crt
--sslPEMKeyFile mongo.pem --eval=\"db.isMaster()['primary']\" | grep
^icp-mongodb-); mongodump --host $PRIMARY --username admin --password
$ADMIN_PASSWORD --authenticationDatabase admin --ssl --sslCAFile /ca/tls.crt
--sslPEMKeyFile mongo.pem --oplog --verbose --gzip
--archive=/backup/mongodb/mongodbdump-$(date +%Y-%m-%d_%H:%M:%S).gz"]
          env:
            - name: ADMIN_PASSWORD
              valueFrom:
                secretKeyRef:
                  key: password
                  name: icp-mongodb-admin
          volumeMounts:
            - mountPath: "/backup"
              name: backup
            - mountPath: "/ca"
              name: cluster-ca
            - mountPath: "/certs"
              name: mongodb-certs
      tolerations:
        - effect: NoSchedule

```

```

    key: dedicated
    operator: Exists
  volumes:
  - name: backup
    persistentVolumeClaim:
      claimName: core-backup
  - name: cluster-ca
    secret:
      secretName: cluster-ca-cert
  - name: mongodb-certs
    secret:
      secretName: icp-mongodb-client-cert
  restartPolicy: Never
  backoffLimit: 1

```

Alternatively, download the file from the following link:

https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch3-Backup-and-Restore/Backup/mongodb-backup_job.yaml

The `ibmcom/icp-mongodb:4.0.5` image is the default MongoDB image for IBM Cloud Private 3.1.2. If the environment does not have internet access, replace `ibmcom` with the private image registry local to the cluster, for example: `mycluster.icp:8500/ibmcom/icp-mongodb:4.0.5`.

3. Create the job:

```

[root@icp-ha-boot ~]# kubectl create -f mongodb-backup_job.yaml
job.batch "mongodbdump" created

```

4. Verify that the job was successful:

```

[root@icp-ha-boot ~]# kubectl get jobs mongodbdump
NAME          DESIRED  SUCCESSFUL  AGE
mongodbdump   1        1           3m

```

5. Verify that the etcd snapshot exists:

```

[root@1f-icp-nfs mongodb]# ls -sh mongodbdump*
976K mongodbdump-2019-02-26_05:58:22.gz

```

Backing up MariaDB

MariaDB in IBM Cloud Private is only used for storing the transient OpenIDConnect data. The default refresh period for user tokens is 12 hours. If this data is lost between restores, the data can simply be regenerated by logging in to the platform. If there is a use case to retain the MariaDB data, you can back up the database using `kubectl` or a Kubernetes job.

Backing up MariaDB using kubectl

Perform the following steps on any machine that has `kubectl` configured to access to the cluster:

1. Get the MariaDB pod name:

```

[root@icp-ha-boot ~]# kubectl -n kube-system get pods -l release=mariadb
-o=jsonpath='{.items[0].metadata.name}'
mariadb-0

```

2. Run the `mysql` command:

```

[root@icp-ha-boot ~]# kubectl -n kube-system exec -it mariadb-0 -c mariadb --
sh -c 'mysqldump --host=$MARIADB_SERVICE_HOST --user=root
--password=$MYSQL_ROOT_PASSWORD --all-databases > /backup.sql'

```

3. Copy the backup file from the mariadb-0 container to the local machine:

```
[root@icp-ha-boot ~]# kubectl cp kube-system/mariadb-0:backup.sql -c mariadb .
```

4. Move the backup.sql to a location outside of the cluster.

Backing up MariaDB using a Kubernetes job

The mariadb-backup_job.yaml and mariadb-backup_cronjob.yaml can be retrieved here:

https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch3-Backup-and-Restore/Backup/mariadb-backup_job.yaml

Run the job in the same way as the etcd and MongoDB jobs.

Backing up the private image registry

The private image registry stores all system and user deployed images in the /var/lib/registry directory on the master nodes. When backing up this data, it is sufficient to create an archive of the data using the OS tools available, such as tar. The loss of this data means losing all users images which will result in failed deployments when the cluster is restored. You can copy the whole contents of /var/lib/registry and move to a location outside of the cluster.

When later restoring a cluster, the system images will already exist after installation, so it does not always make sense to back up the whole /var/lib/registry directory unless it is easier to do so. It's recommended to also store the application images in a location external to the cluster, so that each individual image can be pushed to the image repository if needed.

To back up an individual image, you can use commands such as **docker save -o <backup-name>.tar <repo>/<namespace>/<image-name>:<tag>**

For example, to back up a user deployed python image:

```
docker save -o python-image.tar mycluster.icp:8500/development/python:v1.0.0
```

Backing up the Helm repository

All Helm charts pushed to the IBM Cloud Private Helm repository are stored in MongoDB, and the deployable chart data can be located on the master nodes. Due to this method it's not mandatory to back up the chart archives from the master nodes, but it's recommended to do so in the event of a MongoDB failure, resulting in data loss.

Only one helm-repo pod is running at one time and is scheduled to a master node, using a LocalVolume PersistentVolume. This means that the most up to date chart information from MongoDB is stored on the master node hosting the helm-repo pod, so chart backups should be retrieved from that node.

Backing up the contents of the /var/lib/icp/helmrepo directory on the master node hosting the helm-repo pod is sufficient. Example 3-9 shows how to retrieve the host node name or IP address using **kubectl**.

Example 3-9 Retrieve helm-repo host IP

```
[root@icp-ha-boot cluster]# kubectl -n kube-system get pods -l app=helm-repo  
-o=custom-columns=NAME:.metadata.name,NODE:.spec.nodeName  
NAME                                NODE  
helm-repo-866cd5f9bd-c7cxt          172.24.19.203
```

Use your preferred methods to retrieve the contents of `/var/lib/icp/helmrepo` from the master node and store the data outside of the cluster.

Backing up the Elasticsearch cluster

As the Elasticsearch, Logstash and Kibana (ELK) stack provides users with the ability to retrieve historical log data it is an important component to backup so the same data can be restored in the event of cluster failure. The current logging release does not provide the capability to backup the logging data without suspending the logging service (only partially for installations with multiple management node).

General considerations

By default, IBM Cloud Private configures the ELK stack to retain log data using the logstash index for one day, and in this configuration, it is worth considering whether or not the platform log data is actually meaningful enough to keep. In most cases, the platform log data is transient, especially if the default curation is kept and logs are removed after 24 hours.

Backing up log data is only really valuable if the default 24 hours is extended to a longer duration and where the platform ELK stack is the primary source of application log data. The Logging and Monitoring Chapter provides information on the various use cases for platform and application logging and the alternative approaches to consider, such as deploying a dedicated ELK for applications.

Backing up the logging filesystem

This method requires downtime of management nodes to be able to accurately backup the logging data. This is because Elasticsearch is constantly reading/writing data to the filesystem at `/var/lib/icp/logging` on the management nodes especially when the cluster is active. Attempting to copy the logging data from the filesystem whilst the Elasticsearch cluster is online has a high chance of creating a faulty backup and some shards will be corrupted upon restoration, resulting in data loss.

The recommended way to backup the logging file system is to ensure Elasticsearch is not running on the management node by stopping all containers running on it. Use the method described in “Stopping an IBM Cloud Private node” on page 78 to stop kubelet and docker, then the `/var/lib/icp/logging/elk-data/nodes/0/indices` directory can be safely copied. During this time, no log data generated by the platform or application will be persisted.

In environments with more than one management node, multiple Elasticsearch data pods are deployed (one on each management node), so it is possible to keep the logging services running by repeating the above approach on each management node one by one. During this time, Elasticsearch will persist the data only to the available data pods, which means that there will be an imbalance of spread of replicas per shard on each management node.

Elasticsearch will attempt to correct this as data pods are brought back online so higher CPU utilization and disk I/O is normal in this situation. More information about how replica shards are stored and promoted when data pods are taken offline can be found here:

<https://www.elastic.co/guide/en/elasticsearch/guide/current/replica-shards.html>

Backing up the monitoring data

The monitoring stack comprises of the Alert Manager, Prometheus and Grafana. By default, IBM Cloud Private does not deploy these components with a PersistentVolume (PV) or PersistentVolumeClaim (PVC), so any data stored by these components will be deleted if the container gets restarted. The platform does, however, store any user-configured AlertRules and MonitoringDashboard resources within Kubernetes itself, so these will persist during container restarts.

Alerts and Dashboards

If you have created custom alert rules or Grafana dashboards and modified them in Prometheus or Grafana directly, you should export the YAML definitions to a file and store these resources outside of the cluster, as part of your backup routine.

To export an AlertRule to a file, use the command `kubectl -n <namespace> get alertrule <name> > <name>.yaml` replacing `<namespace>` with the hosting namespace and `<name>` with the AlertRule name.

To export a Dashboard to a file, use the command `kubectl -n <namespace> get monitoringdashboard <name> > <name>.yaml` replacing `<namespace>` with the hosting namespace and `<name>` with the Dashboard name.

Adding persistence to the monitoring stack

The recommended approach to persisting data in the monitoring stack is by adding a PV and PVC to the deployments using `helm upgrade`. The following steps will create a new PV and PVC and mount them to the Alert Manager, Prometheus and Grafana deployments.

Important: These steps will force the containers to be restarted in order to mount the new volume, so any existing data stored in the containers will no longer be available. AlertRules and Dashboards are stored as Kubernetes resources and can be exported, but current monitoring data in Prometheus or additional configurations from modifying ConfigMaps directly may be lost. Be sure to export your additional configurations prior to executing these steps.

1. If you do not have dynamic storage provisioning, create a separate PV and PVC for the Alert Manager (1 Gb), Prometheus (10 Gb) and Grafana (10 Gb). Examples of PVs and PVCs can be found throughout this chapter and the Kubernetes documentation at <https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistent-volumes>.
2. Retrieve the current monitoring Helm chart and the release values, replacing `mycluster.icp` with a value suitable for your environment:

```
wget --no-check-certificate  
https://mycluster.icp:8443/mgmt-repo/requiredAssets/ibm-icpmonitoring-1.4.0.tgz
```

3. Create a new YAML file called `monitoring-override.yaml` containing the PV definition in Example 3-10.

Example 3-10 monitoring-override.yaml

```
prometheus:
  persistentVolume:
    enabled: true
    useDynamicProvisioning: false
    size: 10Gi
    storageClass: ""
    existingClaimName: "monitoring-prometheus"
    selector:
      label: ""
      value: ""

alertmanager:
  persistentVolume:
    enabled: true
    useDynamicProvisioning: false
    size: 1Gi
    storageClass: ""
    existingClaimName: "monitoring-alertmanager"
    selector:
      label: ""
      value: ""

grafana:
  persistentVolume:
    enabled: true
    useDynamicProvisioning: false
    size: 1Gi
    storageClass: ""
    existingClaimName: "monitoring-grafana"
    selector:
      label: ""
      value: ""
```

Replace the values for `useDynamicProvisioning` and `storageClass` to suit your environment.

4. Use `helm upgrade` to apply the changes to the monitoring release:

```
helm upgrade monitoring ibm-icpmonitoring-1.4.0.tgz -f monitoring-override.yaml  
--reuse-values --recreate-pods --tls
```

The data stored by the monitoring stack is portable and can be reused in different instances, so use the storage backup tools available to backup the PV data. If the monitoring data may be restored on a different cluster, you need to ensure that the configuration is the same (for example the Grafana ConfigMap is the same) for the data to be reused in a new deployment of Alert Manager, Prometheus or Grafana.

Using your preferred methods, copy the PV data to a location outside of the IBM Cloud Private cluster.

Backing up Vulnerability Advisor

As discussed in the section “Vulnerability Advisor” on page 76, consider whether it is worth backing up Vulnerability Advisor (VA) and Mutation Advisor (MA) at all. Imported data in to a new cluster will mean that the restored VA and MA data is no longer relevant to the current environment and would not offer any benefit other than analysis of the previous environment. If backing up the VA and MA data is required then this section is relevant.

VA and MA store all its data in two places; on the hosts filesystem and in Elasticsearch. The data stored by VA and MA is not configured to a specific host so data can be transported between installations of IBM Cloud Private. The VA and MA filesystems that require backup are as follows:

- ▶ `/var/lib/icp/va/minio`
- ▶ `/var/lib/icp/va/zookeeper`
- ▶ `/var/lib/icp/va/kafka`

In high availability (HA) deployments, the `/var/lib/icp/va/minio` is on shared storage so back this up using the preferred methods.

Copy this data to a location outside of the IBM Cloud Private cluster.

Backing up the cluster installation data

Depending on how an IBM Cloud Private cluster is restored, it may be necessary to copy the cluster installation data. If IBM Cloud Private needs to be reinstalled on the existing nodes with the same IP addresses, copy the following data in the `<installation-directory>/cluster` directory on the boot node.

- ▶ `cfc-certs` (directory)
- ▶ `config.yaml` (optional)
- ▶ `hosts` (optional)
- ▶ `ssh_key` (optional)

The most important data to be backed up is the `cfc-certs` directory, as it contains the certificates and keys generated for the existing cluster that are used for the whole platform. The `config.yaml`, `hosts` and `ssh_key` files can be easily replaced.

Copy this data to a location outside of the IBM Cloud Private cluster.

Backing up PersistentVolume data

Each storage provider used in IBM Cloud Private will typically have it's own method for backing up data or replicating it to another system. For example, GlusterFS can snapshot volumes, Ceph can export images, vSphere volumes can make use of VMWare native backup functions and hostpath/LocalVolumes benefit from any standard Linux file copy methods.

The platform components mostly store data on the host node, simplifying the backup procedures by allowing the use of common tools to backup the data. For applications that rely on PersistentVolumes, it is the cluster administrators responsibility to ensure that they are familiar with the storage technologies they offer to application developers.

The use of a storage provider running as containers on IBM Cloud Private, such as GlusterFS and Ceph, can complicate the backup process as these technologies typically require the write operations to be suspended while the volume/block is backed up using the native snapshot tools. If the cluster fails and you can no longer interact with the storage provider, there is a higher risk of data loss unless the application can handle such scenarios gracefully.

The recommended way to reduce the risk of application data loss is to use storage providers such as GlusterFS or Ceph on Virtual Machines external to the cluster. This way the dependency on the IBM Cloud Private platform itself is removed and the use of dynamic volume provisioning is still enabled. Regardless of the storage provider, ensure backups of the application data are taken regularly.

3.4.4 Platform restore process

No backup is good without first testing it can be restored, and this process should be thoroughly tested to ensure that each backup can be restored multiple times for any given environment. The information in this section assumes that a new IBM Cloud Private cluster needs to be installed on new or existing infrastructure and that the guidelines in “Platform backup process” on page 84 has been followed to for each of the necessary components.

In the example commands provided, the same backup files created in the “Platform backup process” section will be used. It’s important to take a full infrastructure backup of the new environment before attempting to restore the cluster. It’s also recommended to use the steps in “Backup etcd using kubect” on page 89 to create an etcd backup of the current environment, in case the restore process fails and you end up in a situation where some nodes are restored and others are not. There are several steps that involve replacing core data files and there is a high margin for error.

The restore process will address the following use cases:

- ▶ Restore an IBM Cloud Private configuration and data to an existing cluster
- ▶ Restore an IBM Cloud Private configuration and data to a remote cluster

The local cluster restore steps in this section were performed in a lab environment with LDAP configured and light workloads, tested on the following cluster configurations:

- ▶ Single master, management, proxy, Vulnerability Advisor and 3 x worker nodes
- ▶ 3 x masters, 2 x management, 2 x proxies, 3 x Vulnerability Advisor and 3 x worker nodes
- ▶ 3 x etcd, 3 x masters, 2 x management, 2 x proxies, 3 x Vulnerability Advisor and 3 x worker nodes

The remote cluster restore steps in this section were performed in a lab environment with LDAP configured and light workloads, tested on the following cluster configurations: Single master, management, proxy, Vulnerability Advisor, and 3 x worker nodes.

The core components should be restored in the following order:

1. MongoDB
2. Private image registry
3. etcd
4. Persistent Volume data

The other components can be restored in any order, although it is recommended that the logging data is restored *before* the Vulnerability Advisor. The Persistent Volume Data is restored *last* so that when a cluster is restored, the backed up data is not overwritten with new cluster data, which would later corrupt the restored applications.

Local cluster restore prerequisites

If IBM Cloud Private is reinstalled on the existing infrastructure, the content in the `cfc-certs` directory must be copied to the `<installation_directory>/cluster` directory before installation. This ensures the same certificates from the previous installation are used in the new installation and that any restored components are able to interact with the cluster.

Remote cluster restore prerequisites

If IBM Cloud Private is reinstalled on new infrastructure (for example different nodes with different IP addresses) then you need to ensure that the cluster configuration is as close as possible to the old configuration, in particular the cluster name. The cluster name is used for the image repository (for example, `mycluster.icp:8500`), so all deployments restored from etcd will use the old cluster image repository and if this is different between clusters, you'll need to manually edit each deployed resource to use a different image.

Before restoring the cluster, you'll need to extract all of the platform ConfigMaps and Secrets immediately post-installation, as these will contain the certificates and keys specific to this cluster. When etcd is restored, it will restore the same resources that existed in the backed up installation, which means the new installation will not function correctly if these are not replaced with the current ones. To export all of the current ConfigMaps and Secrets to your local machine, run the script in Example 3-11.

Example 3-11 The save-yamls.sh script

```
#!/bin/bash
namespaces=(kube-system kube-public services istio-system ibmcom cert-manager)
echo "Saving all ConfigMaps, Secrets, Deployments and Daemonsets..."
for ns in ${namespaces[@]}
do
    mkdir $ns.configmap $ns.secret $ns.deployment.extensions
    $ns.daemonset.extensions
    # save configmap, secret, deployment, daemonset
    for n in $(kubectl get -o=name configmap,secret,ds,deployment -n $ns)
    do
        kubectl get -o yaml -n $ns $n > $ns.$n.yaml
    done
done

kubectl get APIService -n kube-system -o yaml > kube-system.servicecatalog.k8s.io >
kube-system.servicecatalog.yaml

echo "Done."
```

It's important to note that unlike the local cluster installation, you do not need to use the same certificates from the backed up `cfc-certs` directory.

Restoring MongoDB

The recommended way to restore MongoDB from a backup is by using the `mongorestore` utility. This section will cover two methods; using `kubectl` and using a Kubernetes job. The use of each method entirely depends on the method used to backup MongoDB. For example, if you used `kubectl` to backup the database, it makes sense to use `kubectl` to restore it. If a CronJob or Job was used, then the `core-backup` PersistentVolume created in the backup steps can be used to remove the manual effort of copying the file to the local machine.

It's worth checking the current database statistics as a reference point, to know whether the `mongorestore` command has successfully repopulated the database.

1. Use `kubectl exec` to create a shell session in the `icp-mongodb-0` pod.

```
[root@icp-ha-boot cluster]# kubectl -n kube-system exec -it icp-mongodb-0 -c
icp-mongodb sh
$
```

2. Log in to the primary replica:

```
$ mongo --host rs0/mongodb:27017 --username admin --password $ADMIN_PASSWORD
--authenticationDatabase admin --ssl --sslCAFile /data/configdb/tls.crt
--sslPEMKeyFile /work-dir/mongo.pem
```

This will output some connectivity information and eventually show the `rs0:PRIMARY>` prompt.

3. Use the `show dbs` command to view the current list of databases and each databases current size.

```
rs0:PRIMARY> show dbs
HELM_API_DB          0.000GB
admin                0.000GB
config               0.000GB
helm-repo_DB         0.001GB
key_protect_metadata 0.000GB
key_protect_secrets  0.000GB
local                0.018GB
metering_DB          0.001GB
platform-db          0.000GB
```

Restoring using kubectl

Perform the following steps to restore MongoDB. Ensure the backup exists on the local machine, and that you can access the cluster using `kubectl`.

1. Find the primary MongoDB replica. During testing this process, some observations were made where the database did not restore correctly when explicitly using the `--host rs0/mongodb:27017 replicaset` instead of the hostname of the primary replica. This step, shown in Example 3-12, ensures that the correct primary instance is used.

Example 3-12 Find the primary MongoDB replica

```
[root@icp-ha-boot cluster]# kubectl exec icp-mongodb-0 -n kube-system -- sh -c
'mongo --host rs0/mongodb:27017 --username admin --password $ADMIN_PASSWORD
--authenticationDatabase admin --ssl --sslCAFile /data/configdb/tls.crt
--sslPEMKeyFile /work-dir/mongo.pem --eval="db.isMaster()["primary\"]" | grep
^icp-mongodb-'
icp-mongodb-0.icp-mongodb.kube-system.svc.cluster.local:27017
```

2. Copy the backup file to the pod from the previous step.

```
[root@icp-ha-boot ~]# kubectl cp backup.gz
"kube-system/icp-mongodb-0:/work-dir/backup.gz" -c icp-mongodb
```

3. Verify the file was copied successfully.

```
[root@icp-ha-boot ~]# kubectl -n kube-system exec icp-mongodb-0 -c icp-mongodb
ls /work-dir
backup.gz
credentials.txt
log.txt
mongo.pem
openssl.cnf
peer-finder
```

4. Execute the restore.

```
[root@icp-ha-boot ~]# kubectl exec icp-mongodb-0 -n kube-system -- sh -c
'mongorestore --host
icp-mongodb-0.icp-mongodb.kube-system.svc.cluster.local:27017 --username admin
```

```
--password $ADMIN_PASSWORD --authenticationDatabase admin --ssl --sslCAFile
/data/configdb/tls.crt --sslPEMKeyFile mongo.pem --oplogReplay --gzip
--archive=/work-dir/backup.gz'
```

```
2019-03-05T13:07:57.309+0000 preparing collections to restore from
2019-03-05T13:07:57.373+0000 reading metadata for metering_DB.usage from
/work-dir/backup/metering_DB/usage.metadata.json
2019-03-05T13:07:57.373+0000 restoring metering_DB.usage from
/work-dir/backup/metering_DB/usage.bson
2019-03-05T13:07:57.406+0000 reading metadata for helm-repo_DB.mgmtrepo-assets
from /work-dir/backup/helm-repo_DB/mgmtrepo-assets.metadata.json
2019-03-05T13:07:57.406+0000 restoring helm-repo_DB.mgmtrepo-assets from
/work-dir/backup/helm-repo_DB/mgmtrepo-assets.bson
...
2019-03-05T13:08:41.235+0000 done
```

5. Repeat step 1 on page 101 to verify that the mongorestore has been successful and added data to the database. The output might look similar to the following:

```
rs0:PRIMARY> show dbs
HELM_API_DB          0.000GB
admin                0.000GB
config              0.000GB
helm-repo_DB        0.003GB
key_protect_metadata 0.000GB
key_protect_secrets  0.000GB
local               2.174GB
metering_DB         0.177GB
platform-db         0.001GB
```

Restoring using a Kubernetes job

Execute the following steps to restore the MongoDB using a job.

1. Create a PersistentVolume (PV) and PersistentVolumeClaim (PVC), similar to the examples in Example 3-1 and Example 3-2 on page 85. This job definition uses the same PV and PVC created in those examples.
2. Create the mongodb-restore_job.yaml as defined in Example 3-13.

Example 3-13 YAML definition for mongodb-restore_job.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: mongodbrestore
  namespace: kube-system
spec:
  template:
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: beta.kubernetes.io/arch
                    operator: In
                    values:
                      - amd64
                      - ppc64le
```

```

      - s390x
    - key: master
      operator: In
      values:
      - "true"
  containers:
    - name: mongodbrestore
      image: ibmcom/icp-mongodb:4.0.5
      command: ["/bin/bash","-c","cat /certs/tls.crt certs/tls.key > mongo.pem;
export PRIMARY=$(mongo --host rs0/mongodb:27017 --username admin --password
$ADMIN_PASSWORD --authenticationDatabase admin --ssl --sslCAFile /ca/tls.crt
--sslPEMKeyFile mongo.pem --eval=\"db.isMaster()['primary']\" | grep
^icp-mongodb-); mongorestore --host $PRIMARY --username admin --password
$ADMIN_PASSWORD --authenticationDatabase admin --ssl --sslCAFile /ca/tls.crt
--sslPEMKeyFile mongo.pem --oplogReplay --gzip
--archive=/backup/mongodb/$BACKUP_NAME"]
      env:
      - name: BACKUP_NAME
        value: "#BACKUP"
      - name: ADMIN_PASSWORD
        valueFrom:
          secretKeyRef:
            key: password
            name: icp-mongodb-admin
      volumeMounts:
      - mountPath: "/backup"
        name: backup
      - mountPath: "/ca"
        name: cluster-ca
      - mountPath: "/certs"
        name: mongodb-certs
  tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Exists
  volumes:
    - name: backup
      persistentVolumeClaim:
        claimName: core-backup
    - name: cluster-ca
      secret:
        secretName: cluster-ca-cert
    - name: mongodb-certs
      secret:
        secretName: icp-mongodb-client-cert
  restartPolicy: Never
  backoffLimit: 1

```

This file can also be downloaded from the following link:

https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch3-Backup-and-Restore/Backup/mongodb-backup_job.yaml

The BACKUP_NAME environment variable should be changed to reflect the name of an actual MongoDB backup file that exists on the PV. If the MongoDB backup job or CronJob in this chapter was used to create the backup, the file name format is

mongodb-backup-<year-month-day_hour:minutes:seconds>.gz. Alternatively, sed can be used to easily do this, by executing `sed -i -e s/#BACKUP/<name.gz>/g mongodb-restore_job.yaml` replacing <name.gz> with your own.

The `ibmcom/icp-mongodb:4.0.5` image is the default MongoDB image for IBM Cloud Private 3.1.2. If the environment does not have internet access, replace `ibmcom` with the private image registry local to the cluster, for example, `mycluster.icp:8500/ibmcom/icp-mongodb:4.0.5`.

Create the job using `kubectl create -f mongodb-restore_job.yaml`. The job should be visible in the **Workloads** → **Jobs** → **BatchJobs** tab in IBM Cloud Private. If the job did not complete successfully, troubleshoot the failure using the guidance provided in the troubleshooting chapter. If the job ran successfully, use the same method of verification using `show dbs` as described earlier. At this point, MongoDB should be restored.

Restoring the private image registry

Copy the contents of the private image registry backup to the `/var/lib/registry` directory on the master node. In high availability environments, the data should be restored to the shared filesystem and all master nodes should be able to access this directory. The system images will already exist after installing IBM Cloud Private, so it may be preferable to restore the user images from other image backups by pushing these images to the image repository before restoring etcd.

After filesystem data is restored, use `kubectl` to delete the image manager pods running on all master nodes:

```
[root@icp-ha-boot ~]# kubectl delete pods -l app=image-manager
pod "image-manager-0" deleted
pod "image-manager-1" deleted
pod "image-manager-2" deleted
```

When the `image-manager` pods are running again, the restored images should now be available. If you backed up images using `docker save`, you can reload them using `dockerload`. For example, use the following command:

```
docker load -i backup-image.tar
```

PersistentVolumes

Due to the large number of storage providers available in IBM Cloud Private, restoring the data to the `PersistentVolumes` is not covered in this chapter. When restoring etcd, the original `PersistentVolume` and `PersistentVolumeClaim` names will be restored so at this point, the original data should be restored as per the storage providers recommendations.

The restored volumes will use the same configuration from the backed up cluster, so restoring this data is a good idea to do before restoring etcd, so that there is minimal disruption to the applications that start up and expecting data to be present. For `hostPath` or `LocalVolume` type volumes and any external storage volumes where the data points on the host/remote storage have not moved, the data restoration should be straight forward.

If you're restoring a cluster that previously used a containerized storage provider such as GlusterFS or Ceph, you will need to ensure that any LVM groups/volumes and disk paths used in the storage configuration are identical to the backed up cluster. Restoring a cluster with containerized storage providers was not tested in this chapter, and may yield unpredicted results.

Restoring etcd on local cluster

Restoring etcd is not a trivial task and there are several steps required, including modifying core files. Ensure the following pre-requisites are met before restoring etcd:

- ▶ You have backed up the cluster at the infrastructure level
- ▶ You have access to the master and management nodes
- ▶ The etcd backup file exists on your local machine

Most steps are run on the etcd/master nodes but the management nodes also require some actions.

Restoring etcd on single and High-Availability installations

The restore procedure for etcd on both single and multi-master installations (where etcd is running on the master nodes) use the same set of steps. Ensure that any steps directed at master nodes are repeated on all master nodes. It's also important to perform each step on all specified nodes before moving on to the next step, to ensure all nodes are at the same stage. This helps with reversing the process if a step should fail. Perform the following steps:

1. On *all master* nodes, create a backup directory and move the etcd pods to it. This will stop the etcd container, so verify that it is stopped:
 - a. Create the backup directories:

```
mkdir -p /etc/cfc/podbackup
mkdir -p /var/lib/etcdbackup
mkdir -p /var/lib/etcdwalbackup
```
 - b. Move the pod data to the backup directory:

```
mv /etc/cfc/pods/* /etc/cfc/podbackup/
```
 - c. Move the current etcd data to the backup directory:

```
mv /var/lib/etcd/* /var/lib/etcdbackup/
mv /var/lib/etcd-wal/* /var/lib/etcdwalbackup/
```
 - d. Verify the etcd pod has stopped.

```
docker ps | grep etcd
```
2. On the *master and management* nodes, stop kubelet and restart docker to stop all running pods:

```
systemctl stop kubelet
systemctl restart docker
```

3. Copy the etcd snapshot file to *all masters*.
4. Copy the script in Example 3-14 and run it on *all masters*. This script will retrieve the current environment information from the `/etc/cfc/podbackup/etcd.json` file on the master nodes, then construct and execute a docker command to generate the new etcd data from the snapshot provided.

When running the script ensure the etcd snapshot file name is passed as a parameter and it is located in the current directory (where the script is run), as the docker container will mount the current directory and use the snapshot file.

Example 3-14 The etcd-restore.sh script

```
#!/bin/bash
[ -z "$1" ] && { echo "Please provide the etcd snapshot file name"; exit 1; }
data_dir="/var/lib/etcd"
restore_dir="/var/lib/etcd/restored"
```



```
# Get etcd docker image details
[ -z $etcd_image ] && etcd_image=$(grep '"image"' /etc/cfc/podbackup/etcd.json |
sed -e 's/^\s*//' -e 's/\",//g' -e 's/\\"//g' -e 's/--//g' -e 's/image://g')

# Set volume mounts
volume_mounts="-v $(pwd):/data -v /etc/cfc/conf/etcd:/certs -v
/var/lib/etcd:/var/lib/etcd"
self=$(grep 'advertise-client-urls=' /etc/cfc/podbackup/etcd.json | sed -e
's/^\s*//' -e 's/\",//g' -e 's/\\"//g')

# Get etcd cluster settings
node_name=$(grep 'name=' /etc/cfc/podbackup/etcd.json | sed -e 's/^\s*//' -e
's/\",//g' -e 's/\\"//g')
initial_advertise_peer_urls=$(grep 'initial-advertise-peer-urls='
/etc/cfc/podbackup/etcd.json | sed -e 's/^\s*//' -e 's/\",//g' -e 's/\\"//g')
initial_cluster=$(grep 'initial-cluster=' /etc/cfc/podbackup/etcd.json | sed -e
's/^\s*//' -e 's/\",//g' -e 's/\\"//g')
initial_cluster_token=$(grep 'initial-cluster-token=' /etc/cfc/podbackup/etcd.json
| sed -e 's/^\s*//' -e 's/\",//g' -e 's/\\"//g')

# Delete the etcd data
rm -rf /var/lib/etcd

# Run the restore on the node
docker run --entrypoint=etcdctl -e ETCDCTL_API=3 ${volume_mounts} ${etcd_image}
--cert /certs/client.pem --key /certs/client-key.pem --cacert /certs/ca.pem
--endpoints ${self} snapshot restore /data/$1 --data-dir=$restore_dir $node_name
$initial_advertise_peer_urls $initial_cluster_token $initial_cluster

# Print result
[ $? -eq 0 ] && echo "etcd restore successful" || echo "etcd restore failed"
```

Alternatively, download and run the etcd-restore.sh script from the following link:

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch3-Backup-and-Restore/Restore/etcd-restore.sh>

If successful, the output should be etcd restore successful.

Tip: There are other ways to read the correct variable values from the etcd.json file by using tools such as jq, but the method used here is designed to be as generic as possible. If this script fails with an error similar to Unable to find image, the etcd.json file may not have been parsed correctly, potentially due to double hyphens or similar characters in the cluster name. Set the etcd_image environment variable manually by getting the docker image name using `docker images | grep etcd` and then setting etcd_image using `export etcd_image=<docker-image>`.

```
[root@icp-master ~]# ./etcd-restore.sh snapshot.db
etcd restore successful
```

If etcd failed to restore, review the error message and take appropriate action. A common error is providing the snapshot name but failing to copy/move it to the current director, so etcd will report that the snapshot file was not found.

5. Move the data in /var/lib/etcd/restored to /var/lib/etcd and remove the empty /var/lib/etcd/restored directory:

```
mv /var/lib/etcd/restored/* /var/lib/etcd/  
rmdir /var/lib/etcd/restored
```

6. The current kubelet pods need to be deleted so that the cache will be consistent between the kubelet and etcd data. Do this on the *master and management* nodes:

```
mount | grep kubelet | awk '{ system("umount \"$3\") }'  
rm -rf /var/lib/kubelet/pods
```

7. Start kubelet on the *master and management* nodes:

```
systemctl start kubelet
```

8. Enable the etcd pods on *all master* nodes:

```
mv /etc/cfc/podbackup/* /etc/cfc/pods/
```

9. Allow around 30 seconds for etcd to rebuild its cluster state and verify it is healthy, replacing 172.24.19.201 with the IP address of any master node. See Example 3-15.

Example 3-15 Verify that etcd is healthy,

```
[root@icp-master ~]# docker run --entrypoint=etcdctl -v  
/etc/cfc/conf/etcd:/certs -v /var/lib/etcd:/var/lib/etcd ibmcom/etcd:3.2.24  
--cert-file=/certs/client.pem --key-file=/certs/client-key.pem  
--ca-file=/certs/ca.pem --endpoints https://172.24.19.201:4001 cluster-health
```

```
member 8a2d3ec6df19666f is healthy: got healthy result from  
https://172.24.19.201:4001  
cluster is healthy
```

Note that the `ibmcom/etcd:3.2.24` image may need to be replaced with the private image registry name for the environment. For example, `mycluster.icp:8500/ibmcom/etcd:3.2.24`.

10. Clear your web browser cache.
11. The IBM Cloud Private platform may take some time to start up all the pods on the master node and become available for use. Once the dashboard is accessible, verify that the cluster has started recreating the previous state by redeploying workloads from the backed up cluster. You may experience a few pod failures for some time until the system becomes stable again. Logging may be particularly slow as it deals with the sudden influx of log data from newly recreated pods all at once.

Restoring etcd on dedicated etcd installations

To restore etcd on an installation with dedicated etcd nodes, you can use the same steps for single and multi-master installations, but using the dedicated etcd nodes in place of the masters. All etcd data resides on the dedicated etcd nodes so the only steps that apply to the masters in this case is the same steps that apply to the management nodes.

Failed etcd restores

If the etcd snapshot was not restored, it may be possible to recover the initial state of the cluster. A common error on RHEL systems is that `systemd` becomes unresponsive when Docker is stopped and started. A usual symptom of this is when the `docker restart` command takes a long time to run, and simple commands that depend on `systemd` (such as `reboot`) just hang. In this situation, only a reboot from the hypervisor will fix a hung `systemd`.

If the etcd snapshot restore was successful on some nodes and failed on others, you will need to either address the issue on the failed nodes and try again, or revert the successful nodes to the original state by restoring the etcd snapshot taken from the new installation.

To revert successful nodes back to the previous state, try the following steps. If these commands do not restore a healthy state, you may need to reinstall the cluster.

On all master nodes (or dedicated etcd nodes if they are separate)

1. Stop Kubelet:

```
systemctl stop kubelet
```

2. Restart Docker:

```
systemctl restart docker
```

3. Remove existing etcd data:

```
rm -rf /var/lib/etcd
```

4. Recreate Etcd directory:

```
mkdir -p /var/lib/etcd
```

5. Copy data from backup:

```
cp -r /var/lib/etcdbackup/* /var/lib/etcd/  
cp -r /var/lib/etcdwalbackup/* /var/lib/etcd-wal/
```

6. Recreate pods directory:

```
mkdir -p /etc/cfc/pods
```

7. Restore pod data:

```
mv /etc/cfc/podbackup/* /etc/cfc/pods/
```

8. Remove kubernetes pods:

```
rm -rf /var/lib/kubelet/pods
```

9. Start kubelet:

```
systemctl start kubelet
```

10. Check the Etcd health:

```
docker run --entrypoint=etcdctl -v /etc/cfc/conf/etcd:/certs -v  
/var/lib/etcd:/var/lib/etcd ibmcom/etcd:3.2.24 --cert-file=/certs/client.pem  
--key-file=/certs/client-key.pem --ca-file=/certs/ca.pem --endpoints  
https://172.24.19.201:4001 cluster-health
```

```
member 8a2d3ec6df19666f is healthy: got healthy result from  
https://172.24.19.201:4001  
cluster is healthy
```

Replace 172.24.19.201 with any etcd/master node IP address.

10. On *all management* nodes (and master nodes too if etcd is running on dedicated nodes)

- a. Move the pod files back to the original directory:

```
mv /etc/cfc/podbackup/* /etc/cfc/pods/
```

- b. Start kubelet:

```
systemctl start kubelet
```

Allow some time for all pods to start up. The cluster should be returned to a normal working state.

Restoring etcd on remote cluster

To restore etcd in a remote cluster, repeat steps 1 to 9 in the section “Restoring etcd on local cluster” on page 106. Some further steps need to be performed to clean up the old nodes and data from the restored etcd snapshot and apply the working configuration. Perform the following steps:

1. After restoring etcd, all the cluster nodes from the initial installation are not present in the new Kubernetes cluster, so for each node kubelet and docker should be restarted. You can check which nodes currently exist using **kubect1 get nodes**. Restart kubelet and docker on *all* cluster nodes using **systemctl stop kubelet && systemctl restart docker && systemctl start kubelet**. After this, the new cluster nodes should be visible using **kubect1 get nodes**.
2. Wait for about 90 seconds after restoring etcd for the restored nodes to change from **Ready** to **NotReady** state, then remove the **NotReady** nodes. Be sure to *only* remove the nodes that contain the old cluster IP addresses. If nodes with the current cluster IP addresses are in **NotReady** state, you should investigate whether the node is online first.
 - a. Get the current list of nodes in the NotReady state.

```
kubect1 get nodes | grep NotReady | awk '{print $1}'
```
 - b. Remove the old nodes.

```
kubect1 delete node $(kubect1 get nodes | grep NotReady | awk '{print $1}')
```
3. Replace the ConfigMap and Secrets restored by etcd with the data exported in the section “Restoring MongoDB” on page 101. Run the script in Example 3-16.

Example 3-16 The restore-yamls.sh script

```
#!/bin/bash
echo "Restoring..."
# re-generate default secret
kubect1 get secret -n kube-system -o wide | grep "\-token-" | awk
'{system("kubect1 delete secret \"$1 \" -n kube-system")}'
kubect1 get secret -n services -o wide | grep "\-token-" | awk '{system("kubect1
delete secret \"$1 \" -n services")}'
kubect1 get secret -n istio-system -o wide | grep "\-token-" | awk
'{system("kubect1 delete secret \"$1 \" -n istio-system")}'
kubect1 get secret -n kube-public -o wide | grep "\-token-" | awk '{system("kubect1
delete secret \"$1 \" -n kube-public")}'
kubect1 get secret -n ibmcom -o wide | grep "\-token-" | awk '{system("kubect1
delete secret \"$1 \" -n ibmcom")}'
kubect1 get secret -n cert-manager -o wide | grep "\-token-" | awk
'{system("kubect1 delete secret \"$1 \" -n cert-manager")}'

namespaces=(kube-system services istio-system kube-public ibmcom cert-manager)

for ns in ${namespaces[@]}
do
    #secret
    for s in $(ls $ns.secret/ | grep -v "\-token-")
    do
        kubect1 delete -f $ns.secret/$s && kubect1 create -f $ns.secret/$s
    done

    #configmap
    for s in $(ls $ns.configmap/)
    do
```

```

        kubectl delete -f $ns.configmap/$s && kubectl create -f $ns.configmap/$s
    done
done

kubectl --force --grace-period=0 delete pv $(kubectl get pv | grep -e
"image-manager-" -e "icp-mongodb-" -e "mariadb-" -e "logging-datanode-" -e
"kafka-" -e "zookeeper-" -e "minio-" | awk '{print $1}')

kubectl patch pv $(kubectl get pv | grep Terminating | awk '{print $1}') -p
'{"metadata":{"finalizers":null}}'

kubectl delete pvc -n kube-system $(kubectl get pvc -n kube-system | grep -e
"image-manager-image-manager-" -e "mongodbdircp-mongodb-" -e
"mysqldata-mariadb-" -e "data-logging-elk-data-" -e "-vulnerability-advisor-" -e
"datadir-vulnerability-advisor-kafka-" -e
"datadir-vulnerability-advisor-zookeeper-" -e
"datadir-vulnerability-advisor-minio-" | awk '{print $1}')

echo "Done."

```

Alternatively download and run the script from the following link:

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch3-Backup-and-Restore/Restore/restore-yamls.sh>

If Vulnerability Advisor is not installed, ignore the errors.

4. Several files created during the initial cluster installation need to be reapplied:
 - a. Change the directory to <installation_directory>/cluster/cfc-components/:


```
cd <installation_directory>/cluster/cfc-components/
```
 - b. Apply the bootstrap-secret.yaml file:


```
kubectl apply -f bootstrap-secret.yaml
```
 - c. Apply the tiller.yaml file:


```
kubectl apply -f tiller.yaml
```
 - d. Apply the image-manager.yaml file:


```
kubectl apply -f image-manager/image-manager.yaml
```
 - e. Apply the whole storage directory:


```
kubectl apply -f storage/
```
5. Restore the resources that use IP addresses in the YAML configuration. Run the script in Example 3-17.

Example 3-17 The restore-ip-yamls.sh script

```

#!/bin/bash

# Set files
files=(
servicecatalog.yaml
daemonset.extensions/service-catalog-apiserver.yaml
daemonset.extensions/auth-idp.yaml
daemonset.extensions/calico-node.yaml

```

```

daemonset.extensions/nginx-ingress-controller.yaml
daemonset.extensions/icp-management-ingress.yaml
deployment.extensions/helm-api.yaml
deployment.extensions/helm-repo.yaml
deployment.extensions/metering-ui.yaml
deployment.extensions/metering-dm.yaml
deployment.extensions/monitoring-prometheus-alertmanager.yaml
deployment.extensions/monitoring-prometheus.yaml
)

# Delete and recreate resources
for f in ${files[@]}
do
    kubectl delete -f kube-system.$f && kubectl create -f kube-system.$f
done

```

Alternatively, download and run the script from the following link:

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch3-Backup-and-Restore/Restore/restore-ip-yamls.sh>

6. Restart kubelet and docker on *all* nodes again. When restarted, the pod should start scheduling across all the cluster nodes.
7. After all nodes are online, all kube-system pods need to be deleted so they inherit the Secrets and ConfigMaps restored earlier.

```
kubectl -n kube-system delete pods $(kubectl -n kube-system get pods | awk '{print $1}')
```

Check that all the pods start being deleted, using `kubectl -n kube-system get pods`. Some pods may be stuck in **Terminating** state, which can prevent the new pods from starting up properly.

To delete all stuck pods, using the following code:

```
kubectl delete pod --force --grace-period=0 -n kube-system $(kubectl get pods -n kube-system | grep Terminating | awk '{print $1}')
```

You may also need to delete all pods running in the istio-system and cert-manager namespaces, so use the same approach as above.

```
kubectl delete pod -n istio-system $(kubectl get pods -n istio-system | awk '{print $1}')
```

```
kubectl delete pod -n cert-manager $(kubectl get pods -n cert-manager | awk '{print $1}')
```

Allow some time for all pods to start up, which may take 10 - 40 minutes, depending on the size of the cluster. The logging pods might take even longer than this to cater for the influx of logs generated from the restore. If individual pods are still in error state, review the errors and attempt to resolve the problems. Issues will vary with each environment, so there is no common resolution. If the restore was successful, you should be able to access and operate the cluster again.

Restoring MariaDB

As discussed in “Backing up MariaDB” on page 94, it’s not usually necessary to restore MariaDB data, however for completeness, the information is provided here. The recommended way to restore MariaDB from a backup is by using the `mysql` utility. This section will cover two methods; using `kubectl` and using a Kubernetes job. The use of each method entirely depends on the method used to backup MariaDB.

For example, if you used `kubect1` to back up the database, it makes sense to use `kubect1` to restore it. If a job was used, then the core-backup PersistentVolume created in the backup steps can be used to remove the manual effort of copying the file to the local machine.

Restore MariaDB using kubect1

Perform the following steps to restore MariaDB. Ensure that the backup exists on the local machine, and that you can access the cluster using `kubect1`.

1. Get the MariaDB pod name:

```
kubect1 -n kube-system get pods -l release=mariadb
-o=jsonpath='{.items[0].metadata.name}'
mariadb-0
```

2. Copy the backup file to the mariadb-0 container:

```
kubect1 cp backup.sql kube-system/mariadb-0:/backup.sql
```

3. Execute the `mysql` command:

```
kubect1 -n kube-system exec -it mariadb-0 -c mariadb -- sh -c 'mysql
--host=$MARIADB_SERVICE_HOST --user=root --password=$MYSQL_ROOT_PASSWORD <
/backup.sql'
```

Restore MariaDB using a Kubernetes job

If MariaDB was backed up using a job, recreate the PersistentVolume and PersistentVolumeClaim definition that was used for the backup job. Download the `mariadb-restore_job.yaml` from https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch3-Backup-and-Restore/Restore/mariadb-restore_job.yaml, replacing the `volumeClaimName` with your own, and create it using `kubect1 create -f mariadb-restore_job.yaml`.

Restoring the logging data

You can restore the logging data by copying the backed up data back to the management node filesystem.

Use the preferred tools to restore the backed up data to `/var/lib/icp/logging/elk-data/nodes/0/indices` on each management node. The Elasticsearch data node(s) need to be restarted to reload the new indices. You can do this by using `kubect1 -n kube-system delete pods logging-elk-data-0`, and allowing some time for the pod to start up again. Once it is started, all the backed up indices should be restored and available in Elasticsearch.

Restoring the monitoring data

Restore the monitoring data by recreating the PersistentVolume (PV) and PersistentVolumeClaim used during the backup steps. Assuming that the appropriate restore methods for the storage technology used for this PV, the monitoring pods should continue from the backed up data.

As AlertRules and Dashboards are stored in CustomResourceDefinitions, these should be restored when etcd is restored. Alternatively, you can recreate the resources from the YAML definitions used to create the AlertRule or Dashboard, or from the exported YAML definitions as described in “Backing up the monitoring data” on page 96. See the Logging and Monitoring chapter for more information about creating monitoring resources.

Restoring Vulnerability Advisor data

The complete restoration of Vulnerability Advisor (VA) and Mutation Advisor (MA) in this step depends on whether or not the logging data (specifically the indices related to VA and MA) was restored successfully, as VA stores data in Elasticsearch. To restore the VA and MA data, extract the backed up filesystem data to `/var/lib/icp/va` on the VA node(s) using the preferred OS tools.

Restoring Helm charts

All Helm chart data is stored in MongoDB and *should* be recreated on the master nodes when the `helm-repo` pod is restored. However, in the event this does not happen, perform the following steps:

1. Make sure that `helm-repo` is running and that the `/var/lib/icp/helmrepo` directory exists all master nodes.
2. Move the charts that you backed up to the `/var/lib/icp/helmrepo` directory on all master nodes.
3. Delete the `helm-repo` deployment pod, and then the `index.yaml` repopulates with the restored charts.



Managing persistence in IBM Cloud Private

While initially Kubernetes environments were used primarily to run stateless applications, benefits of the platform attracted also workloads which require data persistence. Additionally, by design IBM Cloud Private requires persistent storage for running multiple platform services added on top of the regular Kubernetes cluster like: logging, monitoring, identity and access management, and so forth.

IBM Cloud Private cluster needs to be prepared for the data persistence and in this chapter we discuss options available to the IBM Cloud Private administrator regarding data persistence for running containerized applications.

We assume that the reader is familiar with persistent volume and persistent volume claim terminology and in this chapter we are not going to discuss basic storage concepts which you can find in the Kubernetes documentation (<https://kubernetes.io/docs/concepts/storage/persistent-volumes/>).

This chapter has the following sections:

- ▶ 4.1, “Designing the cluster for data persistence” on page 116
- ▶ 4.2, “Persistent storage for platform services” on page 118
- ▶ 4.3, “Configuring persistent storage for application containers” on page 118
- ▶ 4.4, “Managing the storage hosted on IBM Cloud Private” on page 147
- ▶ 4.5, “Performance considerations” on page 151

4.1 Designing the cluster for data persistence

Starting with version 1.9 Kubernetes implements Container Storage Interface (CSI) specification, which was designed to provide a standard and consistent way of attaching persistent volumes to running containers. As other container-related specifications, CSI is pretty open and allows for wide variety of implementations. Some of the storage provisioners are provided with the Kubernetes, while others need to be installed additionally. See <https://kubernetes.io/docs/concepts/storage/storage-classes/#provisioner> for a list of Kubernetes supported provisioners. Thanks to the pluggable architecture users can define multiple storage providers and multiple storage classes, but it is important to understand that there are significant differences related to the performance and stability of available options.

The following sections discuss the key persistent storage aspects that need to be considered.

4.1.1 Workload specific requirements

The following paragraphs list the workload specific requirements when designing the cluster for data persistence.

Size and performance

The fundamental question related to running workloads that require persistent volumes is related to the size of the data volumes and performance in terms of the required *I/O operations per second (IOPS)*. Obviously the storage available to the containers will not be faster than the underlying disks, so the equipment used for building the cluster is a consideration factor as well.

Data sensitivity and isolation

The next question that needs to be considered is the sensitivity of the data that will be stored and the isolation level required for different workloads. If the data is classified as sensitive (in any sense), you should choose the storage technology that provides encryption at rest and in transit. Most of the dynamic Kubernetes provisioners do not provide the possibility to specify the per-volume encryption, *which means that the encryption must be managed at the storage provider level*.

For *GlusterFS* and *Ceph* you can use the disk volume encryption using *dm-crypt*. For *FlexVolume* based drivers that use external storage providers, the encryption level can often be part of the storage class specification (If supported by the back-end hardware). If data isolation is required you can define multiple dedicated hostgroups and create a separate data cluster that uses one of the supported distributed file systems (in such case storageclass used will determine the data placement).

Access mode

Another question is related to concurrency while accessing the data. Kubernetes *PersistentVolumeClaim* can specify one of the three access modes: *ReadWriteOnce (RWO)*, *ReadOnlyMany (ROX)* and *ReadWriteMany (RWX)*. If any of the latter two are required, then the underlying storage technology must support concurrent access to the storage volume from multiple worker nodes.

At the time of writing multi-attach was not supported by storage providers implementing *FlexVolumes* based on the *iSCSI* protocol as well as the *VMware vSphere* provider. If your application requires *ReadOnlyMany* and *ReadWriteMany* modes, then you should select either any of supported distributed file systems (such as *GlusterFS* or *IBM Spectrum Scale*), container native storage provider like *Portworx* or *NFS* as your storage technology.

Dynamic versus static provisioning

Storage volumes can be provisioned either manually by cluster administrator or dynamically when the PersistentVolumeClaim object is created. At the time of writing IBM Cloud Private did not support dynamic provisioning for NFS volumes, however there is an open source `nfs-client-provisioner` implementing this functionality, which can be installed in an IBM Cloud Private cluster. The detailed procedure is described in section “Configuring the dynamic NFS provisioner” on page 123.

For production clusters it is highly recommended to pick the storage option that supports dynamic provisioning.

Snapshots and Data cloning

Finally, do you need ability to easily snapshot and clone persistent volumes? This functionality can prove very useful for testing the applications as well as for performing backup of the data on persistent volumes. At the time of writing such functionality was available from specialized commercial storage solutions like Trident by NetApp, Portworx (with Stork), or IBM Storage Solutions for Containers.

System architecture heterogeneity

IBM Cloud Private is one of the Kubernetes distributions that supports a variety of processor architectures (x86, ppc64le and s390). Some of the distributed file systems may be not available on the specific architecture that you want to use for your work. In case you plan to have heterogeneous worker nodes, we recommend that you check the latest support statement

(https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/supported_system_configuration/supported_os.html).

4.1.2 Maintainability requirements

Another consideration area is about ease and cost of maintaining the storage solution. Is your organization familiar with the selected technology? Do you have a skilled personnel available to monitor, tune and troubleshoot the selected storage provider? Do you have a backup solution in place for the selected provider type?

The answers to the above mentioned questions affect also another choice: whether to use storage provider external to the cluster or the internal one, installed directly on the IBM Cloud Private. When using the distributed file systems installed on IBM Cloud Private nodes the appropriate configuration of the provider is done automatically during the installation and will be automatically upgraded with the future releases of IBM Cloud Private. The drawback of this option is related to the additional CPU load that storage containers will introduce to your worker nodes.

For additional backup considerations see “Backing up PersistentVolume data” on page 99.

4.1.3 Windows worker node support

With IBM Cloud Private version 3.1.2 the initial support for Windows worker nodes was introduced (*as a Technology Preview*). If any of the worker nodes in your cluster is Windows based, this affects the storage technologies that you may use for your persistent volumes. At the time of writing the Windows worker nodes provided no support for dynamic storage provisioning and the only supported volume types were: local, emptyDir and hostPath.

So in case you want to use persistent volumes on Windows worker nodes, you must provide the shared storage statically with one of the Windows supported technologies like CIFS or NFS and mount the volumes as hostPath.

4.2 Persistent storage for platform services

IBM Cloud Private uses persistent volumes for multiple platform services like MariaDB and MongoDB that are required by Identity and Access Management service or ElasticSearch Data Node used by the Logging service. Those volumes are created at the time of installation as Hostpath and LocalVolume types on the designated master and management nodes as shown in Figure 4-1.

IBM Cloud Private

[Create resource](#)
[Catalog](#)
[Docs](#)
[Support](#)

Storage

[PersistentVolume](#)
[PersistentVolumeClaim](#)

Q Search

Create PersistentVolume +

Name	Type	Capacity	Access mode	Reclaim policy	Status	Claim	Created
image-manager-10.10.27.116	LocalVolume	20Gi	RWO	Retain	Bound	kube-system/image-manager-image-manager-0	14 days ago
mongodb-10.10.27.116	LocalVolume	20Gi	RWO	Retain	Bound	kube-system/mongodbdicp-mongodb-0	14 days ago
mariadb-10.10.27.116	LocalVolume	10Gi	RWO	Retain	Bound	kube-system/mysqldata-mariadb-0	14 days ago
logging-datanode-10.10.27.117	LocalVolume	20Gi	RWO	Retain	Bound	kube-system/data-logging-elk-data-0	14 days ago
helm-repo-pv	Hostpath	5Gi	RWO	Delete	Bound	kube-system/helm-repo-pvc	14 days ago
mgmt-repo-pv	Hostpath	5Gi	RWO	Delete	Bound	kube-system/mgmt-repo-pvc	14 days ago

Figure 4-1 Persistent volumes created by IBM Cloud Private during default installation

If you are installing IBM Cloud Private in a high availability topology with multiple master nodes, certain directories on the master nodes must use a shared storage. Those directories are used for storing the platform internal image registry as well as some platform logs.

You can either provide an existing shared storage that uses the NFS protocol or use any other distributed file system like GlusterFS. The section “Configuring persistent storage for application containers” discusses mounting a volume from an external GlusterFS storage cluster.

4.3 Configuring persistent storage for application containers

Apart from using persistent storage for platform services, another hot topic is the selection of the storage option for running containerized applications. Which one should you choose? There is only one right answer to this question: It depends on your specific circumstance.

Luckily Kubernetes provides an open architecture which allows for multiple storage options to coexist within the same cluster. Using the storageclass definition users can pick the storage option that is best for their specific workload.

However, it should be noted that while multiple storage technologies provide more flexibility, it comes at the cost of increased operational complexity. In an enterprise environment, the assumption that users will perform backups on their own is a risky one. The same is true for troubleshooting the performance or access problems, so for the operations team careful selection of the storage technology is crucial.

In this section, we describe how to configure some of the popular storage technologies that are supported in IBM Cloud Private environments:

- ▶ In “Configuring vSphere storage provider for IBM Cloud Private” on page 119 we discuss how to set up and use the VMware storage provider.
- ▶ In “Configuring NFS Storage for IBM Cloud Private” on page 120 we present a short cookbook on setting up the NFS storage provider for IBM Cloud Private.
- ▶ In “Configuring GlusterFS for IBM Cloud Private” on page 125 we present how to configure a GlusterFS distributed file system.
- ▶ In “Configuring Ceph and Rook for IBM Cloud Private” on page 131 we show you how to configure a Ceph RBD cluster.
- ▶ In “Configuring Portworx in IBM Cloud Private” on page 140 we present how to configure a Portworx cluster.
- ▶ Finally in “Configuring Minio in IBM Cloud Private” on page 147 we present how to configure Minio, a lightweight S3-compatible object storage in your IBM Cloud Private cluster.

4.3.1 Configuring vSphere storage provider for IBM Cloud Private

Since IBM Cloud Private is often installed on machines running in VMware vSphere environments, using the VMware cloud provider for dynamic storage provisioning comes as a natural choice. The product documentation on configuring the VMware cloud provider is quite extensive (see

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_cluster/vsphere_1and.html for more information) so we are not going to replicate it in this book.

We mention a few caveats below that are often omitted.

- ▶ *Pay attention to the prerequisites of the VMware cloud provider.* See https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_cluster/vsphere_prereq.html. All of these requirements must be met and the provisioning will not work if any of them are omitted. In vSphere 6.7 or newer, the `disk.EnableUUID` property must be added to each virtual machine configuration (for all cluster nodes) and set to `true`.

The best option is to set this property on a template used to create virtual machines for the cluster nodes. This way when adding additional nodes to the cluster later, you will not forget to perform this step.

- ▶ *Be aware of limitations.* Even though the documentation states that `ReadWriteMany` access mode can be used with the VMware storage when pods are collocated on a single worker node, persistent volumes with `RWX` and `ROX` access modes will not be provisioned dynamically.

If you create a persistent volume claim with access mode `ReadWriteMany` or `ReadOnlyMany` specifying the storageclass that uses the VMware cloud provider, the provisioner just ignores such a request and your claim will be unbound until you create the appropriate persistent volume manually. This may also happen when provisioning services from IBM Cloud Private catalog, as some of the Helm charts available there may use the `RWX` access mode for persistent volumes.

4.3.2 Configuring NFS Storage for IBM Cloud Private

Below we present a short cookbook on setting up an NFS server on RedHat Linux server that can be used for test installations of IBM Cloud Private. It is not recommended to use such an NFS server for performing the HA installation, *because the NFS server will be a single point of failure in such a setup*. If you plan to use NFS as a shared storage for master nodes, consider using a proper NFS server with replication.

Prerequisites

In order to setup an NFS server you must meet the following prerequisites:

- ▶ Linux machine (in this case, we will use a Red Hat Enterprise Linux system) with a sufficient disk space available.
- ▶ Yum repository configured to install NFS packages.
- ▶ The designated NFS server and its clients (in our case the cluster nodes) should be able to reach each other over a network.

Setting up the NFS Server

On the NFS server, run the following command to install the required NFS packages:

```
sudo yum install -y nfs-utils nfs-utils-lib
```

Start the NFS service and enable it to persist after reboots:

```
sudo systemctl start nfs
sudo systemctl enable nfs
```

In case your Linux machine has a firewall turned on, open the ports required for NFS.

```
sudo firewall-cmd --permanent --zone=public --add-service=nfs
sudo firewall-cmd --permanent --zone=public --add-service=mountd
sudo firewall-cmd --permanent --zone=public --add-service=rpc-bind
sudo firewall-cmd --reload
```

Create the directories on the local file system

On the NFS server create a directory that will be used by the clients.

```
mkdir -p <path_to_share>
```

Run the following commands to add the entry to /etc/exports.

```
echo '<path_to_share> <subnet_address>/24(no_root_squash,rw,sync)' >> /etc/exports
```

Replace the <path_to_share> with the path on the file system that you have sufficient disk space and <subnet_address> with the subnet of your worker nodes. In case you are using a different mask than 255.255.255.0 for your worker nodes, replace the '24' with the correct value. Optionally, instead of a subnet you may specify multiple entries with IP addresses of worker nodes and /32 mask as shown below:

```
echo '<path_to_share> <node1_IP>/32(no_root_squash,rw,sync)' >> /etc/exports
echo '<path_to_share> <node2_IP>/32(no_root_squash,rw,sync)' >> /etc/exports
...
echo '<path_to_share> <nodeX_IP>/32(no_root_squash,rw,sync)' >> /etc/exports
```

Tip: Do not put any spaces between the netmask value and the opening bracket '(' -, otherwise you will allow the NFS share to be mounted by any IP address.

Refresh the NFS server with the following command.

```
exportfs -r
```

You can verify if the directory was successfully exported by running the **exportfs** command without parameters as shown in Example 4-1.

Example 4-1 Verification of the exported directories on NFS server

exportfs

```
/storage/vol001 10.10.99.0/24
/storage/vol002 10.10.99.0/24
/storage/vol003 10.10.99.0/24
/storage/vol004 10.10.99.0/24
/storage/vol005 10.10.99.0/24
```

Configuring NFS clients

On all of the worker nodes install the NFS packages.

```
sudo yum install -y nfs-utils nfs-utils-lib
```

You can verify that the shared directories are visible from the NFS client by running the **showmount** command on the worker node as shown in Example 4-2.

Example 4-2 Verification of shared directories from NFS client

showmount -e 10.10.99.30

```
Export list for 10.10.99.30:
/storage/vol005      10.10.99.0/24
/storage/vol004      10.10.99.0/24
/storage/vol003      10.10.99.0/24
/storage/vol002      10.10.99.0/24
/storage/vol001      10.10.99.0/24
```

Testing the NFS configuration

On any worker node try mounting the exported directory.

```
mount -t nfs <nfs_server_IP>:<path_to_share> /mnt
```

If the mount works, you are ready to use NFS in your IBM Cloud Private cluster.

Create a persistent volume

Create a new persistent volume in IBM Cloud Private with the path to the shared directory (Example 4-3). Using the **Create Resource** menu option, paste the code into the dialog box.

Example 4-3 YAML file for creating an NFS PersistentVolume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: demo-pv
  labels:
    app: demo-app
spec:
  accessModes:
```

```
- ReadWriteMany
capacity:
  storage: 1Gi
nfs:
  path: <path_to_share>
  server: <nfs_server_IP>
persistentVolumeReclaimPolicy: Retain
```

Replace the <path_to_share> and <nfs_server_IP> with your own values.

Create a persistent volume claim for the new volume

Using the **Create Resource** menu option, paste the following into the dialog box as shown in Example 4-4.

Example 4-4 YAML file for creating a PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: demo-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  selector:
    matchLabels:
      app: demo-app
```

Testing the volume

Create a new deployment to use the new volume. See Example 4-5.

Example 4-5 YAML file for creating a Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-nfs
  namespace: default
labels:
  app: nginx-nfs
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-nfs
  template:
    metadata:
      labels:
        app: nginx-nfs
    spec:
      volumes:
```



```

- name: demo-nfs-storage
  persistentVolumeClaim:
    claimName: demo-pvc
  containers:
  - name: nginx-nfs-container
    image: nginx
    ports:
    - containerPort: 80
      name: "http-server"
    volumeMounts:
    - mountPath: "/usr/share/nginx/html"
      name: demo-nfs-storage

```

Once the deployment is ready, get the pod name. Execute a shell session on the pod and create a test file as shown in Example 4-6.

Example 4-6 Create a test file

```
kubectl get pods -l app=nginx-nfs
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-nfs-5b4d97cb48-2bdt6	1/1	Running	0	4m

```
kubectl exec -it nginx-nfs-5b4d97cb48-2bdt6 /bin/bash
```

```
root@nginx-nfs-5b4d97cb48-2bdt6:/# cd /usr/share/nginx/html/
```

```
root@nginx-nfs-5b4d97cb48-2bdt6:/usr/share/nginx/html# touch pod-file-check
```

A file pod-file-check should be created on your NFS server.

Configuring the dynamic NFS provisioner

In Example 4-6 we have created a static NFS share and then manually created the PersistentVolume and PersistentVolumeClaim objects. This approach is good for testing, but does not scale well and is not useful in real-life environments. In order to allow for dynamic creation of NFS based PersistentVolume based on the claims, you can install the NFS dynamic provisioner using the Helm chart from the public Kubernetes Helm repo:

<https://github.com/helm/charts/tree/master/stable/nfs-client-provisioner>

Note: Dynamic provisioning of NFS persistent volumes is not supported by IBM, so use it at your own risk.

The nfs-client-provisioner requires existing NFS share to be precreated, so we will reuse the one defined in the section “Create the directories on the local file system” on page 120.

Whitelisting nfs-client-provisioner image

Before installing the Helm chart with the provisioner, we need to allow the provisioner image to be run in our cluster. Create the YAML file with the ClusterImagePolicy as shown in Example 4-7.

Example 4-7 ClusterImagePolicy yaml definition

```

apiVersion: securityenforcement.admission.cloud.ibm.com/v1beta1
kind: ClusterImagePolicy
metadata:

```

```

name: nfs-client-provisioner-whitelist
spec:
  repositories:
    - name: quay.io/external_storage/nfs-client-provisioner:*

```

Use the following command:

```
kubectl create -f nfs-provisioner-imagepolicy.yaml
```

Installing a nfs-client-provisioner Helm chart

In order to follow the procedure below you need **cloudctl** and Helm CLIs installed. Login to your IBM Cloud Private cluster using **cloudctl**.

```
cloudctl login -a https://<cluster_address>:8443 --skip-ssl-validation
```

Install the Helm chart with the following command:

```

helm install --name nfs-client-provisioner --tls --set \
nfs.server=<nfs_server_address> --set nfs.path=<path_to_share> --set \
podSecurityPolicy.enabled=true --set rbac.pspEnabled=true \
stable/nfs-client-provisioner

```

Setting the values `podSecurityPolicy.enabled` and `rbac.pspEnabled` to 'true' is required in default IBM Cloud Private 3.1.x installations that have the `podSecurityPolicies` enabled by default. See the output in Example 4-8.

Example 4-8 Sample output of nfs-client-provisioner Helm chart installation

```

NAMESPACE: kube-public
STATUS: DEPLOYED

```

RESOURCES:

```

==> v1beta1/PodSecurityPolicy
NAME                                DATA    CAPS      SELINUX   RUNASUSER  FSGROUP  SUPGROUP
READONLYROOTFS  VOLUMES
nfs-client-provisioner  false  RunAsAny  RunAsAny  RunAsAny  RunAsAny  false
secret,nfs

```

==> v1/Pod(related)

```

NAME                                READY  STATUS              RESTARTS  AGE
nfs-client-provisioner-5f7bf7f77d-7x7cn  0/1    ContainerCreating    0          0s

```

==> v1/StorageClass

```

NAME            PROVISIONER                                AGE
nfs-client  cluster.local/nfs-client-provisioner  0s

```

==> v1/ServiceAccount

```

NAME                                SECRETS  AGE
nfs-client-provisioner  1          0s

```

==> v1/ClusterRole

```

NAME                                AGE
nfs-client-provisioner-runner  0s

```

==> v1/ClusterRoleBinding

```

NAME                                AGE
run-nfs-client-provisioner  0s

```

```
==> v1/Deployment
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
nfs-client-provisioner	1	1	1	0	0s

As shown in Example 4-8 on page 124, the Helm chart creates a new storage class named “nfs-client”. You can test the dynamic provisioning of the NFS persistent volumes by creating a new persistent volume claim, as shown in Example 4-9.

Example 4-9 *YAML file for creating new PersistentVolumeClaim using nfs-client storageclass*

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-nfs-client
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: nfs-client
```

As a result you should get the new persistent volume claim bound to the dynamically provisioned persistent volume using a subdirectory from your NFS server as shown in Example 4-10.

Example 4-10 *Verification of PersistentVolumeClaim status*

```
kubectl get pvc test-nfs-client -n default
```

NAME	STATUS	VOLUME	CAPACITY
ACCESS MODES	STORAGECLASS	AGE	
test-nfs-client	Bound	pvc-9dfa0156-349a-11e9-9666-0687b75cc59f	1Gi
RWO	nfs-client	9m52s	

When you delete the persistent volume claim the nfs-client-provisioner will automatically remove the associated persistent volume, however the subdirectories created on the NFS share will not be deleted, but renamed with the “archived.” prefix. This is the default behavior of the nfs-client-provisioner and can be changed with the `storageClass.archiveOnDelete` parameter.

4.3.3 Configuring GlusterFS for IBM Cloud Private

In this section. we demonstrate the steps that are required to enable the GlusterFS dynamic storage capabilities in an existing IBM Cloud Private cluster. IBM Cloud Private supports configuring GlusterFS both during and after installation and the configuration steps are the same, except that when configuring GlusterFS on existing worker nodes they must be manually labeled.

What is GlusterFS?

GlusterFS is a scalable, distributed file system that aggregates disk storage resources from multiple servers into a single global namespace. For more information on GlusterFS see <https://docs.gluster.org/en/latest/>

What is Heketi?

Heketi is a dynamic provisioner for GlusterFS that exposes REST API and is capable of creating storage volumes on request. More information on Heketi can be obtained from the project page: <https://github.com/heketi/heketi>

GlusterFS prerequisites

The following prerequisites have to be met when configuring GlusterFS within an IBM Cloud Private cluster.

- ▶ There must be a minimum of 3 worker nodes.
- ▶ Each worker node should have at least one spare disk volume of at least 25GB.
- ▶ Each worker node must be connected to a yum repository or have the glusterfs-client package already installed.

Preparing the GlusterFS storage nodes

GlusterFS requires at least 3 nodes in the cluster to be designated as GlusterFS peers. While it is possible to install GlusterFS on master nodes, it is not recommended *because GlusterFS will introduce additional workload on the nodes*. The following steps describe how to prepare worker nodes for installing GlusterFS. These steps should be repeated for each spare volume that will be used by GlusterFS on each of the worker nodes.

Enabling the GlusterFS Client on the worker nodes

Each cluster node that will use a persistent volume created on GlusterFS needs to have a glusterFS client installed. To install and configure a glusterFS client on Red Hat run the following commands:

```
sudo yum install glusterfs-client
sudo modprobe dm_thin_pool
echo dm_thin_pool | sudo tee -a /etc/modules-load.d/dm_thin_pool.conf
```

For other operating systems you can find the appropriate commands here:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_cluster/prepare_nodes.html.

Identifying the device to be used by GlusterFS

Access the terminal on worker nodes and run the following command:

```
fdisk -l
```

The output will be a list of attached disk volumes, along with their details.

The sample output is shown in Example 4-11.

Example 4-11 Sample output of fdisk command

```
fdisk -l
```

```
Disk /dev/loop0: 870 MiB, 912261120 bytes, 1781760 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

Disklabel type: dos
Disk identifier: 0x13982071

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/loop0p1	*	0	1781759	1781760	870M	0	Empty
/dev/loop0p2		448612	453347	4736	2.3M	ef	EFI (FAT-12/16/32)

Disk /dev/sda: 200 GiB, 214748364800 bytes, 419430400 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x9cc95f36

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sda1	*	2048	419430366	419428319	200G	83	Linux

Disk /dev/sdb: 200 GiB, 214748364800 bytes, 419430400 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

In this example, the disk name we are looking for is /dev/sdb. You may see other names such as /dev/vdb, /dev/sdc as this depends on the hypervisor type and number of disks that are attached to the virtual machine.

Wiping the disk

Run the following command to wipe the disk ready for use.

```
sudo wipefs --all --force /dev/sdb
```

Getting the SymLink

The *config.yaml* file requires a symlink for the hard disk device to use on the virtual machine. Run the following command to retrieve the symlink path.

```
ls -ltr /dev/disk/* | grep 'sdb'
```

This should return a line similar to the following:

```
lrwxrwxrwx 1 root root 9 Mar 5 21:34 pci-0000:00:10.0-scsi-0:0:1:0 -> ../../sdb
```

This gives the /dev/disk/by-path symlink, which we will use in this example. Note that there are other methods available such as /dev/disk/by-id, /dev/disk/by-uuid, or /dev/disk/by-label, but only by-path has been used in this example.

Note: In some environments, such as IBM Cloud Virtual Servers or SUSE Linux Enterprise Server (SLES), no symlinks are automatically generated for the devices. In such case, you have to manually create the symlinks. See https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_cluster/prepare_disks.html#manual for the detailed procedure.

Make a note of the symlink and its link path. For the example device sdb, /dev/disk/by-path is the link path and pci-0000:00:10.0-scsi-0:0:1:0 is the symlink. For each device that you are using for the GlusterFS configuration, you need to add the line in the config.yaml file. For the example device sdb, you would add /dev/disk/by-path/pci-0000:00:10.0-scsi-0:0:1:0 in the config.yaml file.

Important: While symlinks may be identical on different nodes, the actual value will depend on the number and type of the disks in the particular node. So it is better to double-check the symlink path on each node individually.

Configuring the GlusterFS hostgroup in the hosts file

When the nodes are prepared, the next step is to modify the hosts file in the installation directory on the boot node. Create a new stanza named `[hostgroup-glusterfs]` and add the IP addresses of the worker nodes that you prepared before, as shown in Example 4-12.

Example 4-12 Adding hostgroup stanza for glusterfs nodes

```
[master]
...
[management]
..

[worker]
...

[proxy]
...

[hostgroup-glusterfs]
<worker_1_ip>
<worker_2_ip>
<worker_3_ip>
```

Labeling worker nodes for GlusterFS

When installing GlusterFS on an existing cluster you must manually label the worker nodes which are listed in the `[hostgroup-glusterfs]` stanza in the previous step. Run the following command:

```
kubectl label nodes <worker_1_ip> <worker_2_ip> <worker_3_ip> \
hostgroup=glusterfs --overwrite=true
```

Note: In some environments where the nodes names use hostnames instead of the IP addresses, replace the worker node IPs with the appropriate worker node names, as listed in the output of `kubectl get nodes` command.

Configuring the GlusterFS volume information in config.yaml

The GlusterFS configuration needs to be added to the `config.yaml`. This is applicable to both new installations, or when adding GlusterFS to an existing IBM Cloud Private cluster.

In the default configuration, `glusterfs` is disabled in the `config.yaml` supplied by IBM in the `icp-inception` image. To enable GlusterFS in the `config.yaml` file located in the installation directory on the boot node find the `management_services` section and change the value `storage-glusterfs` to `enabled`.

Example 4-13 Enabling the glusterfs management service

```
management_services:
...
  storage-glusterfs: enabled
...
```

Next, edit the GlusterFS Storage Settings section (which is commented out by default) by providing the following properties as shown in Example 4-14.

Example 4-14 Sample GlusterFS configuration section in the config.yaml

```
## GlusterFS Storage Settings
storage-glusterfs:
  nodes:
    - ip: <worker_1_ip>
      devices:
        - /dev/disk/by-path/pci-0000:03:00.0-scsi-0:0:2:0
    - ip: <worker_2_ip>
      devices:
        - /dev/disk/by-path/pci-0000:03:00.0-scsi-0:0:1:0
    - ip: <worker_3_ip>
      devices:
        - /dev/disk/by-path/pci-0000:03:00.0-scsi-0:0:2:0
  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
    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
```

Tip: The entries shown in Example 4-14 are already present in the config.yaml, but are commented out. Uncomment and change them to suit your needs. The resources specified for gluster pods are minimal.

You will find below the description of the important elements of the configuration that were marked as **bold** in Example 4-14 on page 129:

ip	is the IP address of the worker node where you want to deploy GlusterFS (you must add at least three worker nodes).
device	is the full path to the symlink of the storage device.
storageClass	this section defines if the storageclass should automatically be created (created: true) and the properties of a storage class for GlusterFS, such as the name, reclamationPolicy, replicacount, and so forth.
nodeSelector	this section refers to the labels that are added to the worker nodes. Make sure that the key and value are identical to those used in the step “Labeling worker nodes for GlusterFS” on page 128.
prometheus	this section enables automatic collection of the performance metrics related to glusterFS in the Prometheus database. It is recommended that you turn it on by providing the enabled: true value.

The GlusterFS cluster volumes, as well as the hosts and the config.yaml on the boot node, are now ready. You can run the installation with the following command:

```
docker run --rm -t -e LICENSE=accept --net=host -v $(pwd):/installer/cluster \
<icp_inception_image_used_for_installation> addon
```

The GlusterFS cluster nodes are now ready. You can test the storage provisioning with the following yaml file:

Example 4-15 Testing glusterfs storageclass

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: glusterfs-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
  storageClassName: glusterfs
---
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx-glusterfs
  labels:
    app: nginx-glusterfs
spec:
  replicas: 2
  selector:
    matchLabels:
```



```

    app: nginx-glusterfs
  template:
    metadata:
      labels:
        app: nginx-glusterfs
    spec:
      volumes:
      - name: demo-glusterfs-storage
        persistentVolumeClaim:
          claimName: glusterfs-pvc
      containers:
      - name: nginx-glusterfs-container
        image: nginx
        ports:
        - containerPort: 80
          name: "http-server"
        volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: demo-glusterfs-storage

```

To test the volume, follow the steps described in “Testing the volume” on page 122 by using the selector `app=nginx-glusterfs` and creating a file on one pod and verifying its existence on the other one.

4.3.4 Configuring Ceph and Rook for IBM Cloud Private

In this section we demonstrate how to create a Ceph RBD cluster and a Rook management agent in IBM Cloud Private.

What is Ceph?

Ceph is open source software designed to provide highly scalable object, block and file-based storage under a unified system.

Ceph storage clusters are designed to run on commodity hardware, using an algorithm called CRUSH (Controlled Replication Under Scalable Hashing) to ensure data is evenly distributed across the cluster and that all cluster nodes can retrieve data quickly without any centralized bottlenecks. See <https://ceph.com/ceph-storage/> for more information.

What is Rook

Rook is an open source orchestrator for distributed storage systems running in cloud native environments.

Rook turns distributed storage software into self-managing, self-scaling, and self-healing storage services. It does this by automating deployment, bootstrapping, configuration, provisioning, scaling, upgrading, migration, disaster recovery, monitoring, and resource management. Rook uses the facilities that are provided by the underlying cloud-native container management, scheduling and orchestration platform to perform its duties.

Rook integrates deeply into cloud native environments leveraging extension points and providing a seamless experience for scheduling, lifecycle management, resource management, security, monitoring, and user experience.

See <https://rook.io> for more information.

Prerequisites

In order to configure a Ceph storage with Rook on an IBM Cloud Private cluster, the following prerequisites have to be met:

- ▶ IBM Cloud Private cluster must be up and running.
- ▶ Cluster nodes that will be part of the Ceph storage must have additional raw disks available or enough space in existing filesystems.
- ▶ You must have at least a Cluster Administrator role.

Attention: While Ceph provides file based and object storage interfaces, the content provided by IBM creates a distributed storage cluster for block devices (RADOS Block Devices or RBD in short). At the time of writing Rook did not support mounting RBD volumes to multiple nodes at the same time, so this class of storage cannot be used for RWX and ROX access modes.

This guide assumes you have a cluster with internet access to pull the required Helm packages and images.

At the time of writing this book, the installation of a Rook Ceph cluster was a three-step process:

1. Configure role-based access control (RBAC).
2. Install the Rook Ceph Operator Helm chart.
3. Install the Rook Ceph cluster (ibm-rook-rbd-cluster) chart.

Configure role-based access control (RBAC)

A Rook Ceph Operator chart is provided by the Rook project. It does not include definitions of the appropriate RBAC roles required in the IBM Cloud Private environment. Thus, as an initial step you must configure some of the security objects. Authenticate to your cluster with the admin user using the `cloudctl` command and run the following steps.

Create a PodSecurityPolicy

PodSecurityPolicy, as shown in Example 4-16, is required for the Rook Operator and Rook Ceph chart to install properly.

Example 4-16 Sample YAML file defining PodSecurityPolicy for Rook

```
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: rook-privileged
spec:
  fsGroup:
    rule: RunAsAny
  privileged: true
  runAsUser:
    rule: RunAsAny
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: RunAsAny
  volumes:
  - *
  allowedCapabilities:
```

```
- {}
hostPID: true
hostIPC: true
hostNetwork: true
hostPorts:
  # CEPH ports
  - min: 6789
    max: 7300
  # rook-api port
  - min: 8124
    max: 8124
```

Tip: Copying the content from the book PDF may mess up indentation, which results in parsing errors. Access the examples source code at GitHub:
<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide.git>.

Create the file `rook-privileged-psp.yaml` with the content shown in Example 4-16 and run the following command:

```
kubectl create -f rook-privileged-psp.yaml
```

As the result you should see the following output:

```
podsecuritypolicy.extensions/rook-privileged created
```

Create a ClusterRole

Next, you need to create a ClusterRole that uses the PodSecurityPolicy which was defined above, as shown in Example 4-17.

Example 4-17 Sample YAML file with ClusterRole definition for Rook

```
# privilegedPSP grants access to use the privileged PSP.
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: privileged-psp-user
rules:
- apiGroups:
  - extensions
  resources:
  - podsecuritypolicies
  resourceNames:
  - rook-privileged
  verbs:
  - use
```

Create a file `rook-privileged-psp-user.yaml` with the content of Example 4-17 and run the following command:

```
kubectl create -f rook-privileged-psp-user.yaml
```

As the result you should see the following output:

```
clusterrole.rbac.authorization.k8s.io/privileged-psp-user created
```

Create a namespace for Rook operator

The ClusterRole defined above must be bound to a namespace in which you install the Rook operator chart. In our example we will create a new namespace for that purpose with the following command:

```
kubectl create namespace rook
```

As the result you should see the following output:

```
namespace/rook created
```

Create a ClusterRoleBinding for Rook operator

To bind the privileged-psp-user role to the namespace created above, you need to create a ClusterRoleBinding object. See Example 4-18.

Example 4-18 Sample YAML file for ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: rook-agent-psp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: privileged-psp-user
subjects:
- kind: ServiceAccount
  name: rook-agent
  namespace: rook
```

Create a file rook-clusterrolebinding.yaml with the content of Example 4-18 and run the following command:

```
kubectl create -f rook-clusterrolebinding.yaml
```

As the result you should see the following output:

```
clusterrolebinding.rbac.authorization.k8s.io/rook-agent-psp created
```

Create RBAC for Pre-validation checks (Optional)

Rook Ceph Cluster (ibm-rook-rbd-cluster V 0.8.3) Helm chart includes a pre-validation check that requires additional RBAC permissions. You can either create the resources listed in Example 4-19 or set preValidation.enabled=false during the installation.

Example 4-19 Sample YAML file for creating RBAC for pre-validation checks

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "list"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
```

```
  name: pod-reader-binding
subjects:
- kind: ServiceAccount
  name: default
  namespace: rook
roleRef:
  kind: ClusterRole
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

You need a ClusterRoleBinding for each namespace in which you install the Rook Ceph Cluster chart. In our example, we use the same namespace rook we have created in step “Create a namespace for Rook operator” on page 134. Create a rook-pre-validation.yaml file with content of the Example 4-19 on page 134 and run the command:

```
kubectl create -f rook-pre-validation.yaml
```

As the result you should see the following output:

```
clusterrole.rbac.authorization.k8s.io/pod-reader created
clusterrolebinding.rbac.authorization.k8s.io/pod-reader-binding created
```

Install the Rook Ceph Operator Helm chart

At the time of writing this book IBM Cloud Private supported the Rook Ceph Operator in version 0.8.3. Follow the steps below to install Rook Ceph Operator chart using the Helm CLI.

Add the rook Helm repo with the following command:

```
helm repo add rook-beta https://charts.rook.io/beta
```

Create the rook-values.yaml file with the content shown in Example 4-20.

Example 4-20 The values.yaml file used for Rook Ceph Operator chart installation

```
image:
  prefix: rook
  repository: rook/ceph
  tag: v0.8.3
  pullPolicy: IfNotPresent

resources:
  limits:
    cpu: 100m
    memory: 128Mi
  requests:
    cpu: 100m
    memory: 128Mi

rbacEnable: true
pspEnable: true
```

Then, run the following command to install the Rook Ceph Operator chart:

```
helm install --tls --namespace rook --name rook-ceph rook-beta/rook-ceph \
--version v0.8.3 -f rook-values.yaml
```

Example 4-21 shows the sample output.

Example 4-21 Sample output

```
helm install --tls --namespace rook --name rook-ceph rook-beta/rook-ceph -f
rook-values.yaml
```

```
NAME: rook-ceph
LAST DEPLOYED: Fri Feb 22 00:29:10 2019
NAMESPACE: rook
STATUS: DEPLOYED
```

RESOURCES:

```
==> v1beta1/PodSecurityPolicy
```

NAME	DATA	CAPS	SELINUX	RUNASUSER	FSGROUP	SUPGROUP
READONLYROOTFS	VOLUMES					
00-rook-ceph-operator	true	*	RunAsAny	RunAsAny	RunAsAny	RunAsAny false
*						

```
==> v1beta1/CustomResourceDefinition
```

NAME	AGE
clusters.ceph.rook.io	0s
volumes.rook.io	0s
pools.ceph.rook.io	0s
objectstores.ceph.rook.io	0s
filesystems.ceph.rook.io	0s

```
==> v1beta1/ClusterRole
```

	AGE
rook-ceph-system-psp-user	0s
rook-ceph-global	0s
rook-ceph-cluster-mgmt	0s

```
==> v1beta1/Role
```

	AGE
rook-ceph-system	0s

```
==> v1beta1/RoleBinding
```

NAME	AGE
rook-ceph-system	0s

```
==> v1/ServiceAccount
```

NAME	SECRETS	AGE
rook-ceph-system	1	0s

```
==> v1beta1/ClusterRoleBinding
```

NAME	AGE
rook-ceph-global	0s
rook-ceph-system-psp-users	0s

```
==> v1beta1/Deployment
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
rook-ceph-operator	1	1	1	0	0s

```
==> v1/Pod(related)
```

NAME	READY	STATUS	RESTARTS	AGE
rook-ceph-operator-f4cd7f8d5-ks5n5	0/1	ContainerCreating	0	0s

NOTES:

The Rook Operator has been installed. Check its status by running:
kubect1 --namespace rook get pods -l "app=rook-ceph-operator"

Visit <https://rook.io/docs/rook/master> for instructions on how to create & configure Rook clusters

You can verify the successful installation, as shown in Example 4-22.

Example 4-22 Verification of Rook Ceph Operator pods status

kubect1 get pods -n rook

NAME	READY	STATUS	RESTARTS	AGE
rook-ceph-agent-g2r9g	1/1	Running	0	2m49s
rook-ceph-agent-nscs5	1/1	Running	0	2m49s
rook-ceph-agent-qdwz7	1/1	Running	0	2m49s
rook-ceph-operator-6d7f98d49d-shtp9	1/1	Running	0	3m1s
rook-discover-976cj	1/1	Running	0	2m49s
rook-discover-gmkj5	1/1	Running	0	2m49s
rook-discover-jgd7q	1/1	Running	0	2m49s

Install the Rook Ceph cluster (ibm-rook-rbd-cluster) chart

As the last step we install the Rook Ceph cluster chart. You can do this from the IBM Cloud Private catalog. However, below we show the command to install it with the Helm CLI.

Prepare the values.yaml file for Rook RBD Cluster chart

Create a rbd-values.yaml file that contains the nodenames and disk devices as shown in Example 4-23 or node names and directory paths as shown in Example 4-24.

Example 4-23 Sample rbd-values.yaml file specifying target disk devices

```
rookOperatorNamespace: "rook"
cluster:
  storage:
    nodes:
      - name: "10.73.147.205"
        devices:
          - name: "xvde"
      - name: "10.73.147.237"
        devices:
          - name: "xvde"
      - name: "10.73.147.243"
        devices:
          - name: "xvde"
```

Example 4-24 Sample values.yaml file specifying target paths

```
rookOperatorNamespace: "rook"
cluster:
  storage:
    nodes:
      - name: "1.2.3.4"
        directories:
```

```

    - path: "/rook/storage-dir"
  - name: "1.2.3.5"
    directories:
      - path: "/rook/storage-dir"
  - name: "1.2.3.6"
    directories:
      - path: "/rook/storage-dir"

```

Important: You should use *only one* of the above examples *either* raw devices *or* directory paths.

Install the Rook RBD Cluster using the command line

To install the charts from a remote repository using the Helm CLI, you must add this repo on the workstation from which you run the Helm commands. Add the ibm-charts repo with the following commands:

```

export HELM_HOME=~/.helm
helm init --client-only
helm repo add ibm-charts \
https://raw.githubusercontent.com/IBM/charts/master/repo/stable

```

You can verify that this step has been completed successfully as shown in Example 4-25.

Example 4-25 Sample output of helm repo list command

helm repo list

NAME	URL
stable	https://kubernetes-charts.storage.googleapis.com
local	http://127.0.0.1:8879/charts
rook-beta	https://charts.rook.io/beta
ibm-charts	https://raw.githubusercontent.com/IBM/charts/master/repo/stable

Install the ibm-rook-rbd-cluster chart as shown in Example 4-26. It will be deployed to the namespace selected in kubectl context. To target different namespace add **--namespace <namespace>** to the command.

Example 4-26 Installation of ibm-rook-rbd-cluster Helm chart

```

helm install --name rook-rbd-cluster -f rbd-values.yaml \
ibm-charts/ibm-rook-rbd-cluster --tls

```

```

NAME:      rook-rbd-cluster
LAST DEPLOYED: Fri Feb 22 00:33:16 2019
NAMESPACE: kube-system
STATUS:    DEPLOYED

RESOURCES:
==> v1beta1/Cluster
NAME                                                    AGE
rook-rbd-cluster-ibm-rook-rbd-cluster-rook-ceph-cluster 0s

==> v1beta1/Pool
rook-rbd-cluster-ibm-rook-rbd-cluster-rook-ceph-pool  0s

==> v1/StorageClass

```


NAME	PROVISIONER	AGE
rook-storage-class	rook.io/block	0s

==> v1/ServiceAccount

NAME	SECRETS	AGE
rook-ceph-cluster	1	0s

==> v1beta1/Role

NAME	AGE
rook-ceph-cluster	0s

==> v1beta1/RoleBinding

NAME	AGE
rook-ceph-cluster	0s
rook-ceph-cluster-mgmt	0s

==> v1/RoleBinding

rook-ceph-osd-psp	0s
rook-default-psp	0s

NOTES:

1. Installation of Rook RBD Cluster

rook-rbd-cluster-ibm-rook-rbd-cluster-rook-ceph-cluster successful.

```
kubectl get cluster rook-rbd-cluster-ibm-rook-rbd-cluster-rook-ceph-cluster
--namespace kube-system
```

2. A RBD pool kube-system-pool is also created.

```
kubectl get pool --namespace kube-system
```

3. Storage class rook-storage-class can be used to create RBD volumes.

```
kubectl get storageclasses rook-storage-class
```

You can verify that the resources were created as shown in Example 4-27. Target namespace may be different depending on to which namespace you have deployed the chart.

Example 4-27 Verify that the resources were created

```
kubectl get cluster rook-rbd-cluster-ibm-rook-rbd-cluster-rook-ceph-cluster \
--namespace kube-system
```

NAME	AGE
rook-rbd-cluster-ibm-rook-rbd-cluster-rook-ceph-cluster	3m

```
kubectl get pool --namespace kube-system
```

NAME	AGE
rook-rbd-cluster-ibm-rook-rbd-cluster-rook-ceph-pool	4m

```
kubectl get storageclasses rook-storage-class
```

NAME	PROVISIONER	AGE
rook-storage-class	rook.io/block	4m8s

Verification of the ceph storage cluster

To verify if everything works fine, create a new persistent volume claim with the storage class `rbd-storage-class`. Create the `ceph-test-pv.yaml` file that contains the lines shown in Example 4-28.

Example 4-28 ceph-test-pv.yaml file

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ceph-test
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: rbd-storage-class
```

Run the following command to create a persistent volume claim:

```
kubectl create -f ceph-test-pv.yaml
```

The output of the command should be:

```
persistentvolumeclaim/ceph-test created
```

Finally, you can verify that the persistent volume was dynamically created and bound as shown in Example 4-29.

Example 4-29 Verification of the PersistentVolumeClaim status

```
kubectl get pvc -n default
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS
MODES	STORAGECLASS	AGE		
ceph-test	Bound	pvc-e6a6d1e8-3635-11e9-a33b-06d591293f01	1Gi	RWO
rbd-storage-class		13s		

4.3.5 Configuring Portworx in IBM Cloud Private

Portworx is a commercial, cloud native storage and data management solution for Kubernetes. There is a Portworx community helm chart provided in IBM Cloud Private catalog. When installing the helm chart you automatically get a 30 day Trial license for PX-Enterprise product. Limited free PX-Developer version is also available from Portworx. More info on Portworx here: <https://portworx.com/products/introduction/>.

Portworx Helm chart has multiple options regarding what drives/filesystems to use as well as set of optional components (for example, a dedicated UI to manage Portworx cluster). The described procedure uses default settings and installs the PX-Enterprise Trial version with Storage Operator Runtime for Kubernetes (Stork), as described on github.com:

Stork can be used to co-locate pods with where their data is located. This is achieved by using a kubernetes scheduler extender. The scheduler is configured to use stork as an extender. Therefore, every time a pod is being scheduled, the scheduler will send filter and prioritize requests to stork.

Stork will then check with the storage driver. You can either configure the default Kubernetes scheduler to communicate with stork or launch another instance of kube-scheduler.¹

More information on Stork can be found here: <https://github.com/libopenstorage/stork>.

Prerequisites

To install the Portworx Helm chart you need at least 1 node with 4 CPU cores, 4 GB RAM and available free unmounted volumes or filesystems of at least 8 GB size. (See the full list of prerequisites here:

<https://docs.portworx.com/start-here-installation/#installation-prerequisites>).

Portworx requires an existing key-value database to be available at the installation time (for example etcd or consul). *When installing on IBM Cloud Private it is not recommended to reuse cluster etcd database.*

Installing etcd database for portworx

The easiest way to provide etcd database for Portworx is to install one on IBM Cloud Private. In our environment we install single node instance (not recommended for production use) using the Bitnami Helm chart. To install the etcd Helm chart, follow the steps below:

1. Authenticate to IBM Cloud Private cluster using **cloudctl login** command.
2. Add the Bitnami chart repo:

```
helm repo add bitnami https://charts.bitnami.com
```
3. Install the etcd Helm chart to a <namespace> providing existing <storage_class> name. If you don't yet have the storage class with dynamic provisioning available, you can create a static persistent volume with any name as the storage class and then use it in the command.

```
helm install bitnami/etcd --version 2.0.0 --name px --namespace <namespace> \
--set persistence.storageClass=<storage_class> --tls
```
4. Verify that the etcd service has been created. Note down the clusterIP address.

```
kubectl get svc px-etcd
```

Installing the Portworx Helm chart

To install the Portworx Helm chart version 1.0.1 (was available in the community catalog at the time of writing the book) you need the following steps:

1. Add ImagePolicy or ClusterImagePolicy to allow the docker images required by the Portworx chart to be run in your cluster. As the Portworx chart has to be installed into the kube-system we used ImagePolicy. Create a px-imagepolicy.yaml file with the content shown in Example 4-30.

Example 4-30 ImagePolicy for portworx images

```
apiVersion: securityenforcement.admission.cloud.ibm.com/v1beta1
kind: ImagePolicy
metadata:
  name: "portworx-image-policy"
  namespace: "kube-system"
spec:
  repositories:
    - name: "docker.io/portworx/*"
    policy:
```

¹ <https://github.com/libopenstorage/stork#hyper-convergence>

```

      va:
        enabled: false
- name: "docker.io/openstorage/*"
  policy:
    va:
      enabled: false
- name: "gcr.io/google-containers/*"
  policy:
    va:
      enabled: false
- name: "docker.io/lachlanevenson/*"
  policy:
    va:
      enabled: false
- name: "docker.io/hrishi/*"
  policy:
    va:
      enabled: false
- name: "quay.io/k8scsi/*"
  policy:
    va:
      enabled: false

```

Add the Image policy with the command:

```
kubectl create -f portworx-imagepolicy.yaml
```

2. By default IBM Cloud Private in version 3.1.x uses the restricted mode, which means that the service account used to run pods needs a binding to a cluster role with the proper cluster RBAC permissions. Portworx chart creates a set of service accounts and roles, but these are not sufficient in an IBM Cloud Private environment. Create an additional set of clusterrolebindings as shown Example 4-31.

Example 4-31 Additional ClusterRoleBindings for portworx chart

```

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: px-account-anyuid-binding
roleRef:
  kind: ClusterRole
  name: ibm-anyuid-hostaccess-clusterrole
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: px-account
  namespace: kube-system

```

```

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: px-account-privileged-binding
roleRef:
  kind: ClusterRole
  name: ibm-privileged-clusterrole

```

```

    apiGroup: rbac.authorization.k8s.io
  subjects:
  - kind: ServiceAccount
    name: px-account
    namespace: kube-system
---

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: stork-sa-anyuid-binding
roleRef:
  kind: ClusterRole
  name: ibm-anyuid-clusterrole
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: stork-account
  namespace: kube-system
---

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: stork-sched-sa-anyuid-binding
roleRef:
  kind: ClusterRole
  name: ibm-anyuid-clusterrole
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: stork-scheduler-account
  namespace: kube-system
---

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: portworkx-hook-anyuid-binding
roleRef:
  kind: ClusterRole
  name: ibm-anyuid-clusterrole
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: portworx-hook
  namespace: kube-system

```

Create a portworx-clusterrolebindings. yaml file with the content of Example 4-31 and install it with the following command:

```
kubectl apply -f portworx-clusterrolebindings.yaml
```

If the `ibm-anyuid-clusterrole` is not bound to service accounts used by Portworx, you will notice that the pods will stall in `ContainerCreatingError` state with the following message in `kubectl describe pod` output:

Error: container has `runAsNonRoot` and image will run as root

You can now install a Portworx chart either from the IBM Cloud Private catalog page or using the Helm client. Below we show how to install Portworx with the Helm CLI.

3. Add the community chart repo to your local Helm client using:

```
helm repo add ibm-community-charts \
https://raw.githubusercontent.com/IBM/charts/master/repo/community
```

4. Install the Portworx chart as shown in Example 4-32. Use `<ClusterIP_of_etcd_service>` from the `px-etcd` service (See step 4 of “Installing etcd database for portworx” section.) In our installation when we used the `px-etcd.<namespace>.svc.cluster.local` service name, the Portworx pods were not able to resolve the name.

Attention: By default, the Portworx chart creates a daemonset that means that the Portworx pods will spread on all of the worker nodes in the cluster, scanning for available disk drives. To prevent this behaviour, label the nodes where you *don't* want Portworx to be installed with the following command:

```
kubectl label nodes <nodes list> px/enabled=false --overwrite
```

Example 4-32 Installation of the Portworx Helm chart

```
helm install --name portworx ibm-community-charts/portworx --namespace \
kube-system --set etcdEndPoint=etcd:http://<ClusterIP_of_etcd_service>:2379 --tls
```

```
NAME: portworx
LAST DEPLOYED: Sun Mar 10 17:56:36 2019
NAMESPACE: kube-system
STATUS: DEPLOYED
```

RESOURCES:

```
==> v1/ConfigMap
```

NAME	DATA	AGE
stork-config	1	0s

```
==> v1/ClusterRole
```

NAME	AGE
node-get-put-list-role	0s
stork-scheduler-role	0s
stork-role	0s

```
==> v1/Service
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
portworx-service	ClusterIP	10.0.0.246	<none>	9001/TCP	0s
stork-service	ClusterIP	10.0.118.214	<none>	8099/TCP	0s

```
==> v1beta1/Deployment
```

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
stork-scheduler	3	3	3	0	0s
stork	3	3	3	0	0s

```
==> v1beta1/StorageClass
```

NAME	PROVISIONER	AGE
------	-------------	-----

```

portworx-null-sc    kubernetes.io/portworx-volume 0s
portworx-db2-sc     kubernetes.io/portworx-volume 0s
portworx-db-sc      kubernetes.io/portworx-volume 0s
portworx-shared-sc  kubernetes.io/portworx-volume 0s

```

```

==> v1/StorageClass
stork-snapshot-sc  stork-snapshot 0s

```

```

==> v1/ServiceAccount
NAME                SECRETS  AGE
px-account          1        0s
stork-scheduler-account 1        0s
stork-account        1        0s

```

```

==> v1/ClusterRoleBinding
NAME                AGE
node-role-binding   0s
stork-scheduler-role-binding 0s
stork-role-binding   0s

```

```

==> v1beta1/DaemonSet
NAME      DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE
portworx  3        3        0      3          0          <none>         0s

```

```

==> v1/Pod(related)
NAME                READY  STATUS             RESTARTS  AGE
portworx-6qlvp      0/1    ContainerCreating  0         0s
portworx-c4h4t      0/1    ContainerCreating  0         0s
portworx-p86qn      0/1    ContainerCreating  0         0s
stork-scheduler-679f999679-5gbt7 0/1    ContainerCreating  0         0s
stork-scheduler-679f999679-jq6gp 0/1    ContainerCreating  0         0s
stork-scheduler-679f999679-w5mv5 0/1    ContainerCreating  0         0s
stork-86bb9cb55d-467wr 0/1    ContainerCreating  0         0s
stork-86bb9cb55d-969vf 0/1    ContainerCreating  0         0s
stork-86bb9cb55d-khwrv 0/1    ContainerCreating  0         0s

```

NOTES:

Your Release is named "portworx"
 Portworx Pods should be running on each node in your cluster.

Portworx would create a unified pool of the disks attached to your Kubernetes nodes.

No further action should be required and you are ready to consume Portworx Volumes as part of your application data requirements.

For further information on usage of the Portworx in creating Volumes please refer <https://docs.portworx.com/scheduler/kubernetes/preprovisioned-volumes.html>

For dynamically provisioning volumes for your Stateful applications as they run on Kubernetes please refer <https://docs.portworx.com/scheduler/kubernetes/dynamic-provisioning.html>

Want to use Storage Orchestration for hyperconvergence, Please look at STork here.
 (NOTE: This isnt currently deployed as part of the Helm chart)

<https://docs.portworx.com/scheduler/kubernetes/stork.html>

Refer application solutions such as Cassandra, Kafka etcetera.

<https://docs.portworx.com/scheduler/kubernetes/cassandra-k8s.html>

<https://docs.portworx.com/scheduler/kubernetes/kafka-k8s.html>

We tested this procedure on an IBM Cloud Private 3.1.2 cluster running Centos 7 nodes. Additional steps might be required while running it on IBM Cloud Virtual Servers and SLES.

If the installation succeeds you can verify the Portworx cluster status as shown in Example 4-33.

Example 4-33 Verification of portworx cluster status

```
PX_POD=$(kubectl get pods -l name=portworx -n kube-system -o \
jsonpath='{.items[0].metadata.name}')
kubectl exec $PX_POD -n kube-system -- /opt/pwx/bin/pxctl status
```

```
Status: PX is operational
License: Trial (expires in 31 days)
Node ID: dc9ddffb-5fc3-4f5d-b6df-39aee43325df
      IP: 10.10.27.120
      Local Storage Pool: 1 pool
      POOL      IO_PRIORITY      RAID_LEVEL      USABLE  USED      STATUS  ZONE
REGION
default 0      HIGH      raid0      50 GiB  6.0 GiB Online  default
      Local Storage Devices: 1 device
      Device Path      Media Type      Size      Last-Scan
      0:1      /dev/sdb      STORAGE_MEDIUM_MAGNETIC 50 GiB      10 Mar 19
10:14 UTC
      total      -      50 GiB
Cluster Summary
      Cluster ID: px-cluster-1a2f8f7b-f4e8-4963-8011-3926f00ac9bc
      Cluster UUID: 91f47566-cac4-46f0-8b10-641671a32afd
      Scheduler: kubernetes
      Nodes: 3 node(s) with storage (3 online)
      IP      ID      SchedulerNodeName
StorageNode  Used  Capacity  Status  StorageStatus  Version
Kernel      OS
      10.10.27.120  dc9ddffb-5fc3-4f5d-b6df-39aee43325df  icp-worker3
Yes      6.0 GiB 50 GiB      Online Up (This node) 2.0.2.3-c186a87
3.10.0-957.5.1.el7.x86_64      CentOS Linux 7 (Core)
      10.10.27.118  85019d27-9b33-4631-84a5-7a7b6a5ed1d5  icp-worker1
Yes      6.0 GiB 50 GiB      Online Up      2.0.2.3-c186a87
3.10.0-957.5.1.el7.x86_64      CentOS Linux 7 (Core)
      10.10.27.119  35ab6d0c-3e3a-4e50-ab87-ba2e678384a9  icp-worker2
Yes      6.0 GiB 50 GiB      Online Up      2.0.2.3-c186a87
3.10.0-957.5.1.el7.x86_64      CentOS Linux 7 (Core)
Global Storage Pool
      Total Used      : 18 GiB
      Total Capacity : 150 GiB
```


5. The Portworx chart does not automatically create a storage class. To enable dynamic storage provisioning, create a `portworx-sc.yaml` file containing the lines in Example 4-34.

Example 4-34 Portworx storage class definition

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: portworx-sc
provisioner: kubernetes.io/portworx-volume
parameters:
  repl: "1"
```

Apply the definition with command:

```
kubectl create -f portworx-sc.yaml
```

There are many other parameters that you can define in the Portworx storage class:

<https://docs.portworx.com/portworx-install-with-kubernetes/storage-operations/create-pvcs/dynamic-provisioning/#using-dynamic-provisioning>

Attention: The procedure described in this section will not work in the air-gapped environments as Portworx pods download the content and activate Trial license on the Portworx site. For air-gapped installation see Portworx manuals at <https://docs.portworx.com/portworx-install-with-kubernetes/on-premise/airgapped>.

Your portworx storage cluster should be ready with active Trial license for 30 days. In case you want to purchase a production license visit Portworx website.

4.3.6 Configuring Minio in IBM Cloud Private

Minio is the special kind of persistent storage available in IBM Cloud Private. It does not provide Kubernetes persistent volumes as NFS, GlusterFS or Rook Ceph as described above, but it exposes existing persistent storage for the applications using a S3-compatible object storage interface. Consider using Minio if any of the applications that will run in your IBM Cloud Private cluster require object storage.

Because the IBM Knowledge Center provides detailed step-by-step instructions on deploying Minio Helm chart we will not replicate this in this book. See https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_cluster/configure_minio.html.

4.4 Managing the storage hosted on IBM Cloud Private

In this section, we briefly describe the features provided by IBM Cloud Private to help users with managing the second day operations on the storage used for persistent volumes.

4.4.1 Monitoring storage status and performance

Monitoring of the storage performance for storage like vSphere datastores, NFS or any external storage clusters is outside of the scope of this book. Usually these storage technologies have dedicated monitoring tools available to the storage teams. In this section we focus on the monitoring of the storage solution hosted within IBM Cloud Private.

In order to help users manage their storage hosted on an IBM Cloud Private cluster, both GlusterFS as well as Rook Ceph solutions provided by IBM have the Prometheus endpoints enabled. This means that they automatically collect performance metrics related to storage in the platform monitoring service.

1. To see the dashboards that are provided out-of-the-box open the IBM Cloud Private Console and from the lefthand side menu. Select **Platform** and **Monitoring** as shown in Figure 4-2. This will open the default Grafana interface.

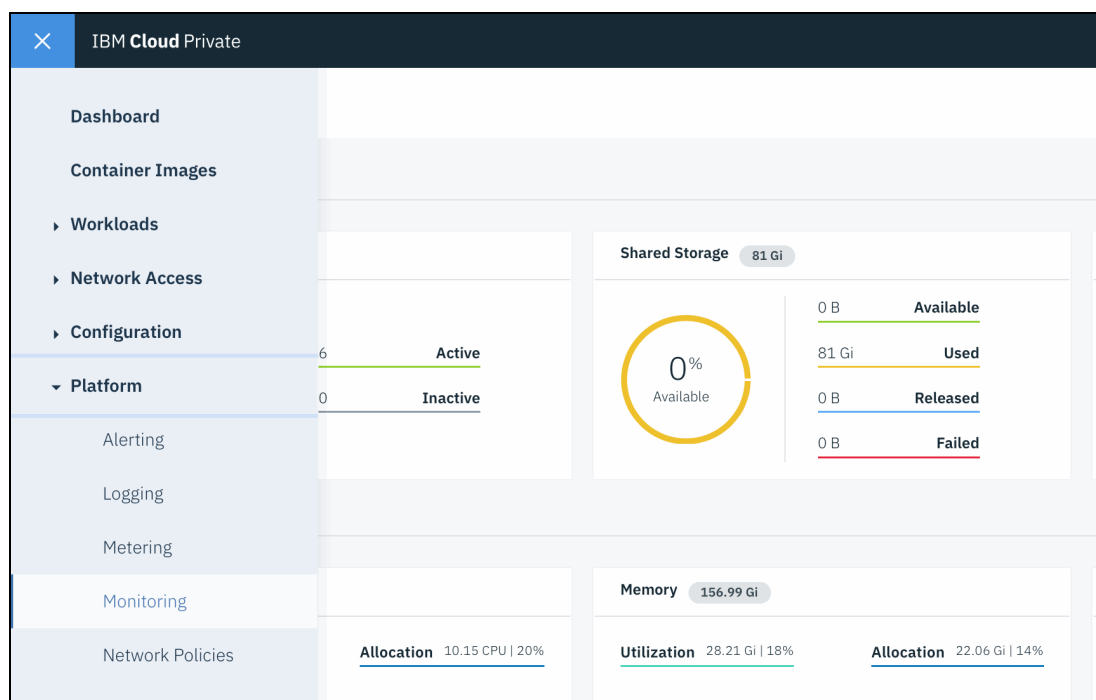


Figure 4-2 Opening the IBM Cloud Private monitoring UI

2. On the default Grafana dashboard click the **Home dropdown** in the upper left corner, as shown in Figure 4-3.

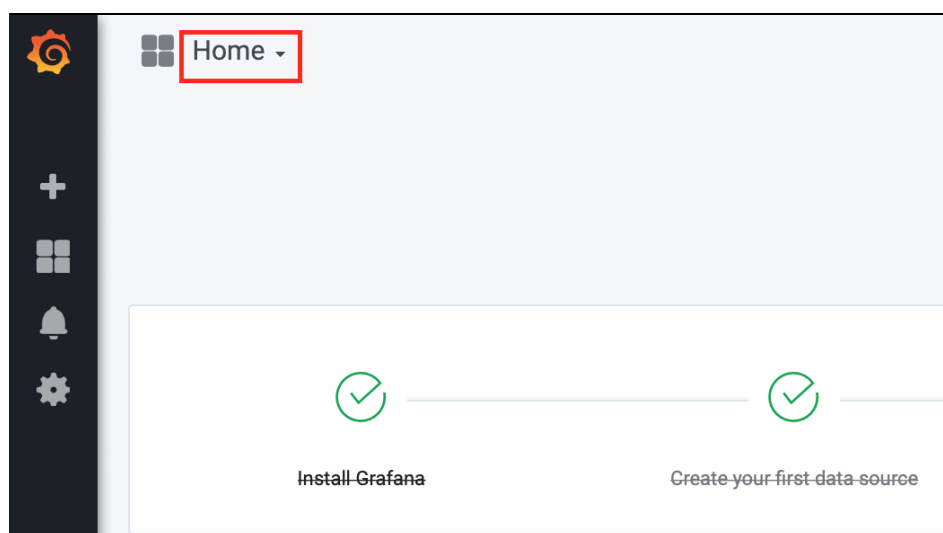


Figure 4-3 Opening the list of available dashboards

3. This opens the list of dashboards that are loaded in your environment. IBM Cloud Private V 3.1.2 provides out-of-the-box dashboards for storage solutions as shown in Figure 4-4.

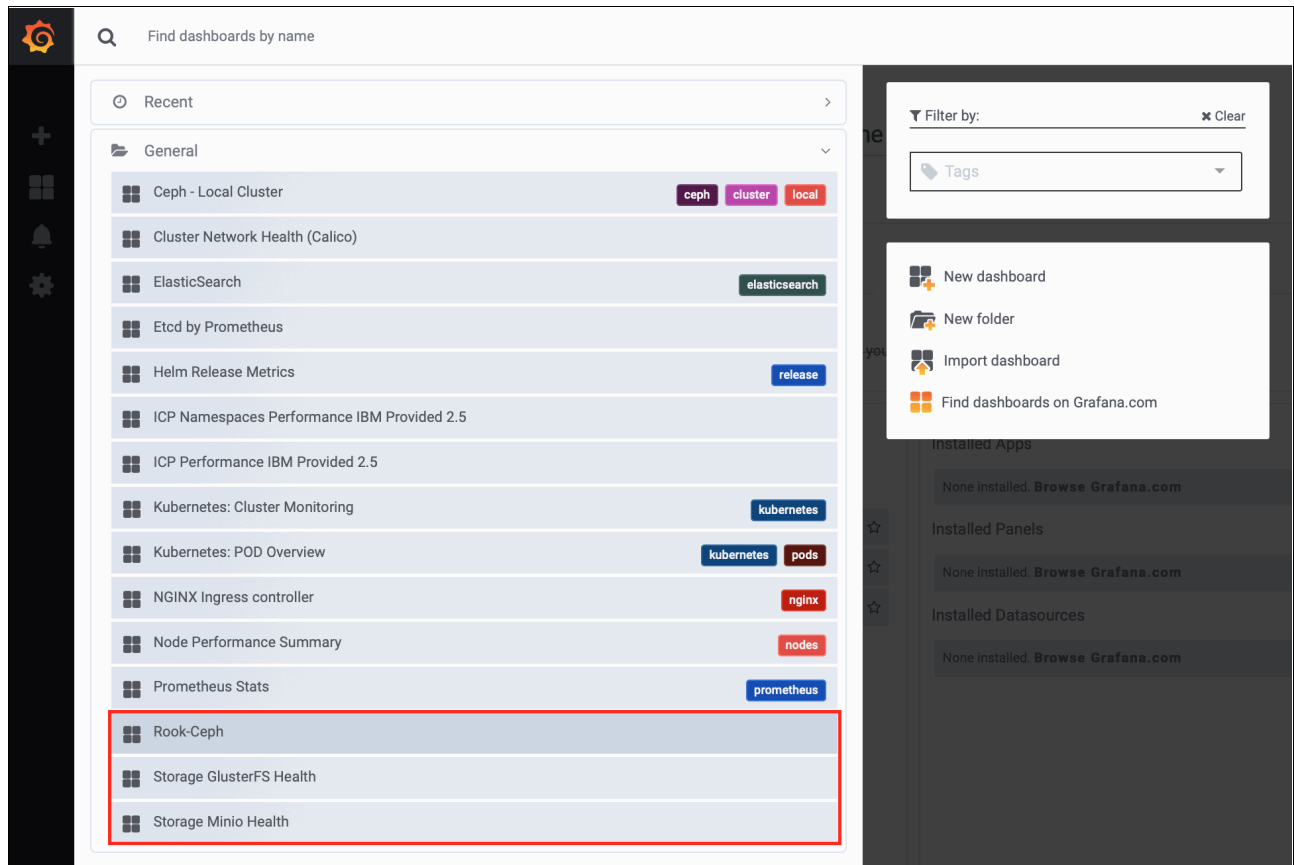


Figure 4-4 List of predefined Grafana dashboards available in IBM Cloud Private

4. Selecting any item on the list will open the dashboard. In Figure 4-5 we show a sample Rook Ceph dashboard.

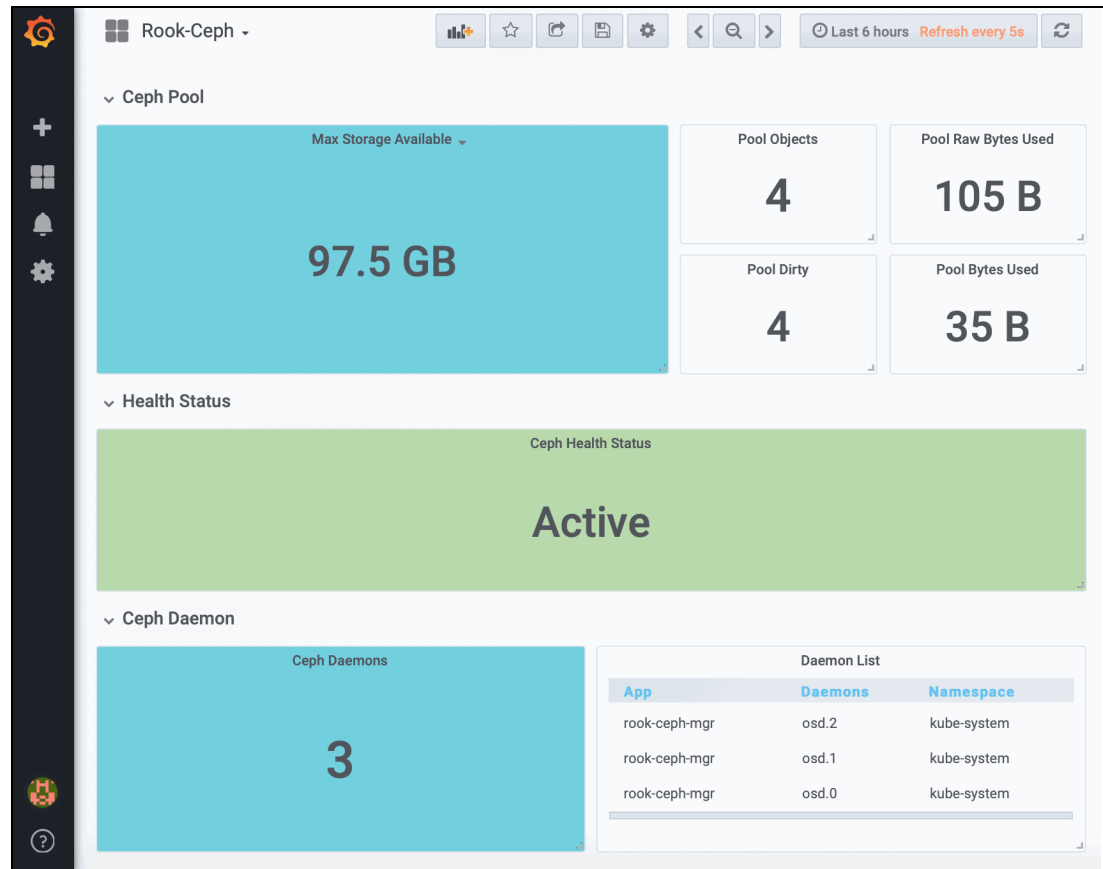


Figure 4-5 Sample Rook Ceph dashboard

If you do not see any data, make sure that the user account that you have used for authenticating to the IBM Cloud Private Console has the access rights to the namespace hosting the Rook Ceph Helm chart.

4.4.2 Extending the available storage

One of the common tasks related to the storage is extending available capacity. For solutions hosted on the IBM Cloud Private cluster described in this book, it can be achieved by adding more raw disks to the distributed filesystem clusters. New devices can be mounted either on the worker nodes that are already used as storage providers or on new nodes - extending not only the storage space, but also the I/O performance.

IBM Knowledge Center provides detailed step-by-step instructions on configuring additional storage space for GlusterFS. See https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_cluster/increase_vol.html.

4.5 Performance considerations

During writing this chapter we set up an IBM Cloud Private environment with different types of persistent storages:

- ▶ NFS
- ▶ GlusterFS
- ▶ Ceph
- ▶ VMware vSphere
- ▶ Portworx

All of the persistent volumes used for tests were hosted on datastore residing on local drives inside the ESXi server.

We have run series of performance benchmarks using dbench and pgbench tools. All the volumes used were hosted on SSD drives inside a ESXi server used for the test.

Below we present the results of these tests.

Note: All of the storage providers were used with the default settings as of IBM Cloud Private V 3.1.2. It is very likely that some tuning could improve the results, especially for the distributed filesystems.

4.5.1 Performance test using dbench

The first test we performed using a dbench image is available here:

<https://github.com/logdna/dbench> which makes use of the Flexible I/O tester (FIO) tool (<https://github.com/axboe/fio>).

Table 4-1 shows the results averaged over several runs of the test jobs.

Attention: *The results are provided on a as-is basis* using the output that was produced by the FIO tool. Note that NFS uses the *client caching feature*, which allows the data to be cached on the NFS client and read out of local memory instead of remote disk. This might affect the results for NFS. Also, Portworx applies some I/O optimization techniques which might affect the results.

Table 4-1 Performance test using dbench

Provider	Random Read [IOPS]	Random Write [IOPS]	Random Read [MB/s]	Random Write [MB/s]
Vmware	2 184	2 021	121	135
Gluster	1 023	465	55	23
Ceph RBD	855	438	47	77
NFS	16 300	1 664	660	73
Portworx	126 000	31 500	4 277	279

4.5.2 PostgreSQL database performance

The raw IOPS (Input/Output Operations Per Second) test may not be relevant for a real application workload, so in our econd test we used a standard PostgreSQL performance test `pgbench`. For each of the storage classes in our test environment we installed one instance of the `ibm-postgres-dev` Helm chart (version 1.1.3) using default settings. Each PostgreSQL deployment was exposed using the NodePort. From an external system we ran the following commands:

```
pgbench -i -d -s 64 -h $PGHOST -p $PGPORT -U admin postgres
```

This command creates the sample database of 1 GB size. After that we ran the test command:

```
pgbench -c 10 -j 2 -t 10000 -h $PGHOST -p $PGPORT -U admin postgres
```

The command runs 10 clients in parallel with 2 threads each, where each client executes 10000 transactions.

Neither the database nor the storage were not optimized in any way as the goal of this test was to just show the relative performance of different storage options using the same environment.

The results returned by the benchmark are shown in Table 4-2.

Table 4-2 Results of pgbench benchmark

Provider	Transactions per Second
Vmware	1898
Gluster	103
Ceph RBD	225
NFS	1283
Portworx	676

The results are in general consistent with IOPS test, showing significant performance advantage of Vmware over the other providers. This is not very surprising in our test setup. However, it is worth noticing that Ceph RBD performed over 2 times better than GlusterFS (while both storage providers reported similar IOPS performances). From the distributed storage providers, Portworx showed almost 6 times better performance than GlusterFS and 3 times better than Ceph RBD.

Additionally, on GlusterFS the benchmark returned few errors such as:

```
ERROR: unexpected data beyond EOF in block 105200 of relation base/16384/16503
HINT: This has been seen to occur with buggy kernels; consider updating your system.
```

A brief search on the Internet returns the hint from GlusterFS documentation: “*Gluster does not support so called ‘structured data’, meaning live, SQL databases.*”

This might change with the newer versions of GlusterFS.



Logging and monitoring

The logging and monitoring tools used in IBM Cloud Private are at the core of how users interact with their application log data and metrics. The Elasticsearch, Logstash and Kibana (also called ELK) stack is a suite of open source tools designed to provide extensive log capture, retention, visualization and query support for application log data, and is the primary way for users to interact with their application log data. Alert Manager, Prometheus and Grafana is another suite of open source tools that provides the user with powerful capabilities to query metrics for their application containers, and raise alerts when something isn't quite right.

This chapter explores each of these components in depth, describing their function in an IBM Cloud Private cluster and how the logging and monitoring systems can be leveraged to cover a range of common use cases when used with IBM Cloud Private.

This chapter has the following sections:

- ▶ 5.1, “Introduction” on page 154
- ▶ 5.2, “IBM Cloud Private Logging” on page 155
- ▶ 5.3, “IBM Cloud Private Monitoring” on page 222

5.1 Introduction

This section will provide an overview and describe the main functions of each of the components within the logging and monitoring tools used in IBM Cloud Private. It will discuss the importance of each role and how each technology plays a key part in providing the user with all the tools necessary to store, view, query and analyze log data and performance metrics for their deployed application containers.

5.1.1 Elasticsearch, Logstash and Kibana

Elasticsearch, Logstash and Kibana are the three components that make up the ELK stack. Each component has a different role, but is heavily integrated with each other to allow application log analysis, visualization and RESTful access to the data generated by the whole IBM Cloud Private platform. The ELK stack is coupled with a Filebeat component that deals with collecting the raw log data from each node in the cluster.

Elasticsearch

Elasticsearch is a NoSQL database that is based on the Lucene search engine. Elasticsearch in IBM Cloud Private has three main services that process, store and retrieve data; the client, master and data nodes.

The client (also known as a ‘smart load-balancer’) is responsible for handling all requests to Elasticsearch. It is the result of a separation of duty from the master node and the use of a separate client enables stability by reducing the workload on the master.

The master node is responsible for lightweight cluster-wide actions such as creating or deleting an index, tracking which nodes are part of the cluster and deciding which shards to allocate to which nodes.

Data nodes hold the shards that contain the documents you have indexed. Data nodes handle data related operations like CRUD, search and aggregations. These operations are I/O and memory intensive. It is important to monitor these resources and to add more data nodes if they are overloaded. The main benefit of having dedicated data nodes is the separation of the master and data roles to help stabilise the cluster when under load.

Logstash

Logstash is a log pipeline tool that accepts inputs from various sources, executes different transformations, and exports the data to various targets. In IBM Cloud Private, it acts as a central input for different log collectors, such as Filebeat, to rapidly buffer and process data before sending it to Elasticsearch. Logstash can be configured to output data not just to Elasticsearch, but a whole suite of other products to suit most other external log analysis software.

Kibana

Kibana is an open source analytics and visualization layer that works on top of Elasticsearch that allows end users to perform advanced data analysis and visualize your data in a variety of charts, tables and maps. The lucene search syntax allows users to construct complex search queries for advanced analysis and, feeding in to a visualization engine to create dynamic dashboards for a real time view of log data.

Filebeat

Filebeat is a lightweight shipper for forwarding and centralizing log data. Filebeat monitors the specified log locations to collect log events and data from containers running on a host and forwards them to Logstash.

5.2 IBM Cloud Private Logging

The ELK stack plays a key role in an IBM Cloud Private cluster, as it acts as a central repository for all logging data generated by the platform and the only method to access log data without accessing the Kubernetes API server directly. This section will explore how the whole logging system works and how it can be used effectively to satisfy several common use cases seen by Cluster Administrators when faced with configuring or customizing IBM Cloud Private to suit their requirements for viewing and storing application log data.

5.2.1 ELK architecture

Figure 5-1 shows the architecture overview for the IBM Cloud Private platform ELK stack and the logical flow between the components.

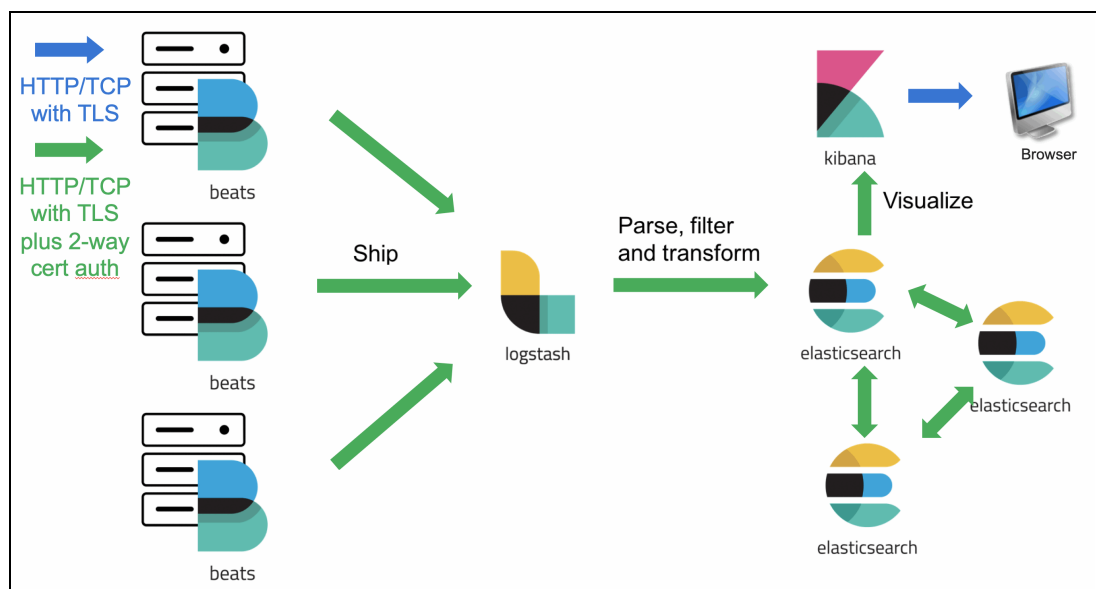


Figure 5-1 ELK high level overview

In IBM Cloud Private, the platform logging components are hosted on the management nodes, with the exception of Filebeat that runs on all nodes, collecting log data generated by Docker. Depending on the cluster configuration, there are multiples of each component. For example, in a High Availability (HA) configuration with multiple management nodes, multiple instances of the Elasticsearch components will be spread out across these nodes. The overview in Figure 5-2 shows how the Elasticsearch pods are spread across management nodes

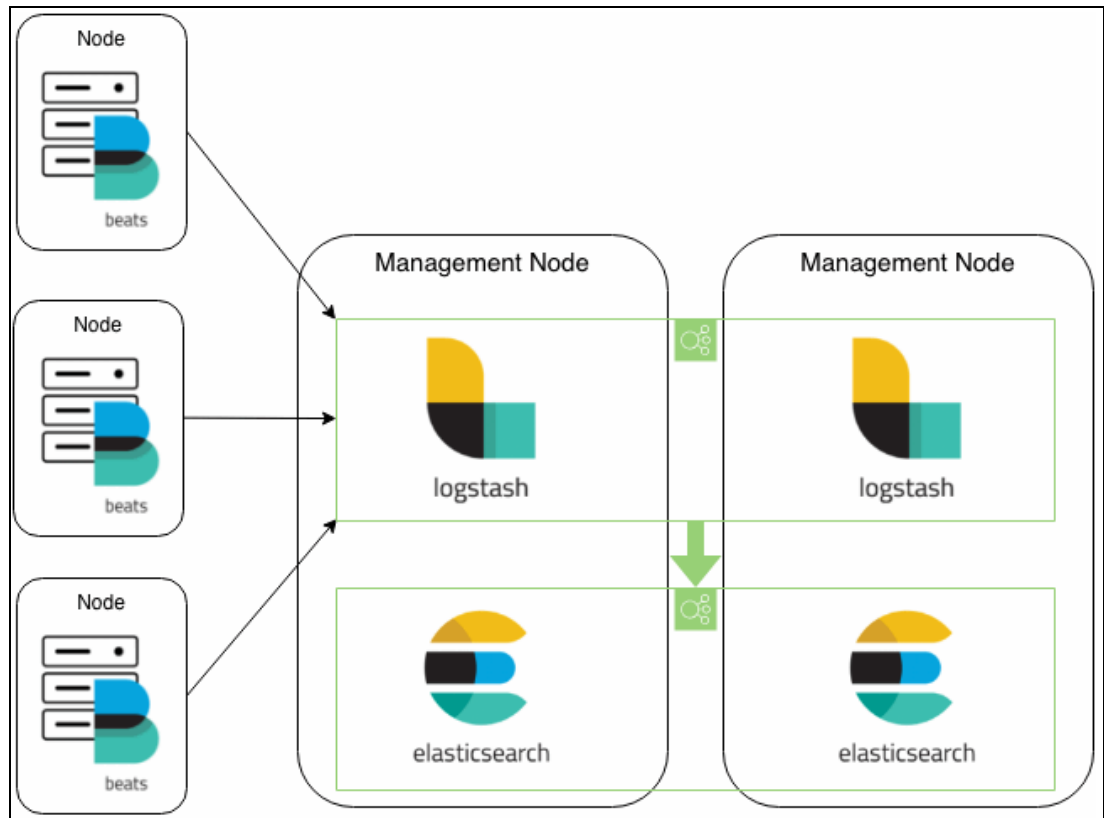


Figure 5-2 Logging on IBM Cloud Private with multiple management nodes

Each of the components are configured to run only on management nodes and, where possible, spread evenly across them to ensure that the logging service remains available in the event a management node goes offline.

5.2.2 How Elasticsearch works

This section will explore how raw log data is collected by Filebeat and transformed into an Elasticsearch document ready for analysis and querying. Figure 5-3 shows an overview of the process from Filebeat collecting the data, to Elasticsearch storing it in an IBM Cloud Private environment with multiple management nodes.

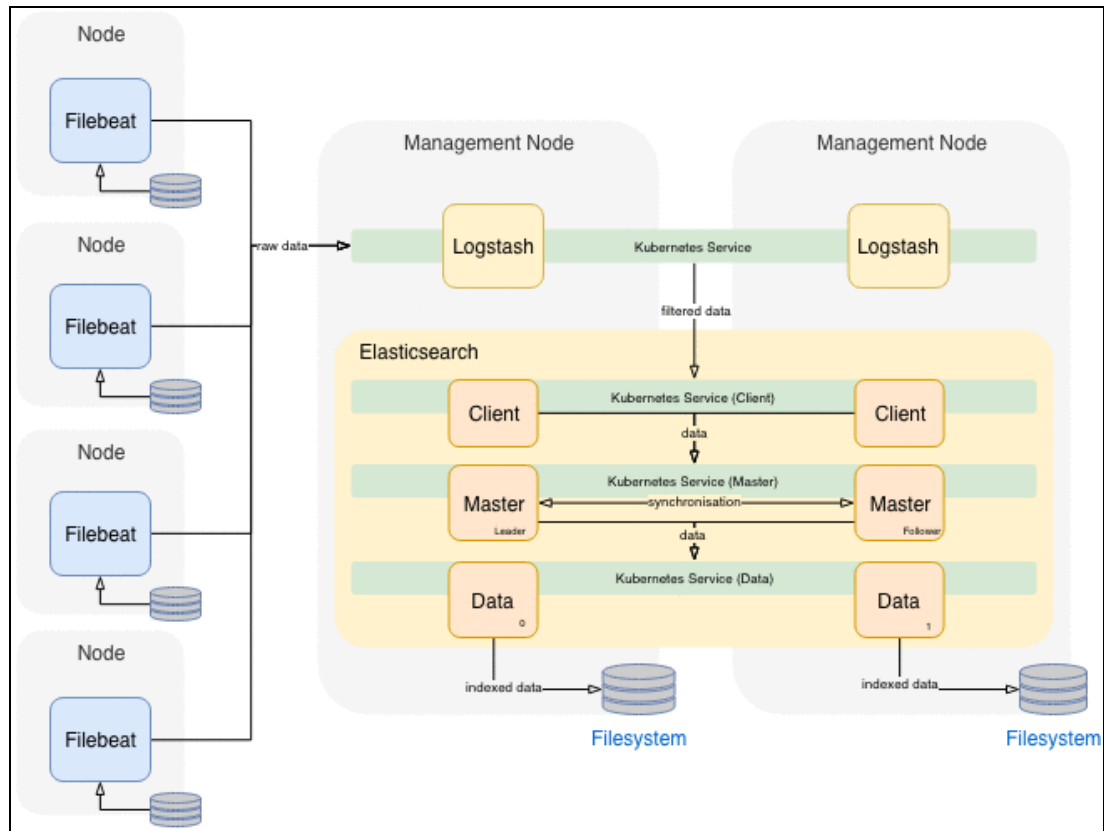


Figure 5-3 Elasticsearch data flow overview with multiple management nodes

Collecting the data

On every IBM Cloud Private cluster node, an instance of Filebeat is running. The Filebeat pods are controlled by a Daemonset that is actively keeping at least one Filebeat pod running on each cluster node to ensure all node are collecting log data across the whole cluster. By default, ELK is configured to be cluster-wide, monitoring all namespaces on all nodes.

All containers running on a host write out data to stdout and stderr, which is captured by Docker and stored on the host filesystem. In IBM Cloud Private, Docker is configured to use the json-file logging driver, which means that Docker captures the standard output (and standard error) of all the containers and writes them to the filesystem in files using the JSON format. The JSON format annotates each line with its origin (stdout or stderr) and its timestamp and each log file contains information about only one container. For each container, Docker stores the JSON file in a unique directory using the container ID. A typical format is `/var/lib/docker/containers/<container-id>/<container-id>-json.log`.

The `/var/lib/docker/containers/` directory has a symlink for each file to another location at `/var/log/pods/<uid>/<container-name>/<number>.log`.

Filebeat then continuously monitors the JSON data for every container, but it does not know anything about the container IDs. It does not query Kubernetes for every container ID, so the kubelet service creates a series of symlinks pointing to the correct location (useful for centralized log collection in Kubernetes) and it retrieves the container logs from the host filesystem at `/var/log/containers/<container-name>_<namespace>_<uid>.log`. In the Filebeat configuration, this filepath is used to retrieve log data from all namespaces using a wildcard filepath `/var/log/containers/*.log` to retrieve everything, but it's also possible to configure more accurate filepaths for specific namespaces.

As a reference, it's worth noting that this filepath is not the same as the one generated by Docker. Using a the `ls -l` command shows that the `/var/log/containers/<pod_name>_<namespace>_<container_name>-<uid>.log` contains a symlink to `/var/log/pods/<id>/<container-name>/<index>.log`. Following the trail, the `<index>.log` file contains yet another symlink to finally arrive at `/var/lib/docker/containers/<uid>/<uid>-json.log`. Example 5-1 shows the symlinks for the `icp-mongodb-0` container in a real environment.

Example 5-1 Container log symlink trail

```
[root@icp-boot ~]# ls -l /var/log/containers/
...
lrwxrwxrwx 1 root root 66 Feb 11 09:44
icp-mongodb-0_kube-system_bootstrap-c39c7b572db78c957d027f809ff095666678146f8d04dc102617003
f465085f2.log -> /var/log/pods/96b4abef-2e24-11e9-9f38-00163e01ef7a/bootstrap/0.log
...
[root@icp-boot ~]# ls -l /var/log/pods/96b4abef-2e24-11e9-9f38-00163e01ef7a/bootstrap/
lrwxrwxrwx 1 root root 165 Feb 11 09:44 0.log ->
/var/lib/docker/containers/c39c7b572db78c957d027f809ff095666678146f8d04dc102617003f465085f2
/c39c7b572db78c957d027f809ff095666678146f8d04dc102617003f465085f2-json.log
```

Filebeat consists of two components; inputs and harvesters. These components work together to tail files and send event data to a specific output. A harvester is responsible for reading the content of a single file. It reads each file, line by line, and sends the content to the output. An input is responsible for managing the harvesters and finding all sources to read from. If the input type is `log`, the input finds all files on the drive that match the defined glob paths and starts a harvester for each file.

Filebeat keeps the state of each file and, if the output (such as Logstash) is not reachable, keeps track of the last lines sent so it can continue reading the files as soon as the output becomes available again, which improves the overall reliability of the system. In IBM Cloud Private, Logstash is pre-configured as an output in the Filebeat configuration, so Filebeat is actively collecting logs from all cluster nodes and sending the data to Logstash.

Filtering and sending the data

Logstash has 3 main stages; inputs, filters, and outputs. The input stage is the means in which Logstash receives data. It can be configured to receive data from a number of sources, such as the file system, Redis, Kafka, or Filebeat. The filters stage is where the inbound data from Filebeat is transformed to extract certain attributes, such as the pod name and namespace, remove sensitive data such as the host and drop empty lines from the log file in an effort to remove unnecessary processing in Elasticsearch. Logstash has many different outputs and available plug-ins. A full list of outputs can be found at <https://www.elastic.co/guide/en/logstash/5.5/output-plugins.html>. In IBM Cloud Private, the default is Elasticsearch. Logstash send the data, along with the index name (defaults to `logstash-<year.month.day>`) to Elasticsearch to process and store.

Indexing and storing the data

When Elasticsearch receives a new request, it gets to work processing and storing the data, ready for searching at a later stage. When discussing how it stores and searches the data it holds, it's important to understand a few key terms, such as indices, shards and documents.

First and foremost, Elasticsearch is built on top of Lucene. Lucene is Java based information retrieval software primarily designed for searching text based files. Lucene is able to achieve fast search responses because, instead of searching the text directly, it searches an 'index'. In this context, an index is a record of all the instances in which a keyword exists. To explain the theory, a typical example is to think about searching for a single word in a book where all the

attributes of the word are 'indexed' at the back of the book, so you know on which page, line and letter number that word exists, for all instances of the word. This act of 'reverse search' is called an inverted index. As the name suggests, it is the inverse of a forward index, where a word would be searched starting from the front of the book and sequentially working through all the pages. So where a forward index would resolve all instances of a word from searching pages to find words (pages > words), the inverted index uses a data-centric approach and searches words to find pages (words > pages). This is how data is retrieved from text based files quicker than a traditional database, where a single query sequentially searches a database record by record until it finds the match.

In Elasticsearch, an index is a single item that defines a collection of shards and each shard is an instance of a Lucene index. A shard is a basic scaling unit for an index, designed to sub-divide the whole index in to smaller pieces that can be spread across data nodes to prevent a single index exceeding the limits of a single host. A Lucene index consists of one or more segments, which are also fully functioning inverted indexes. The data itself is stored in a document, which is the top level serialized JSON object (with key-value pairs), stored in the index and the document is indexed and routed to a segment for searching. Lucene will search each of these segments and merge the results, which is returned to Elasticsearch.

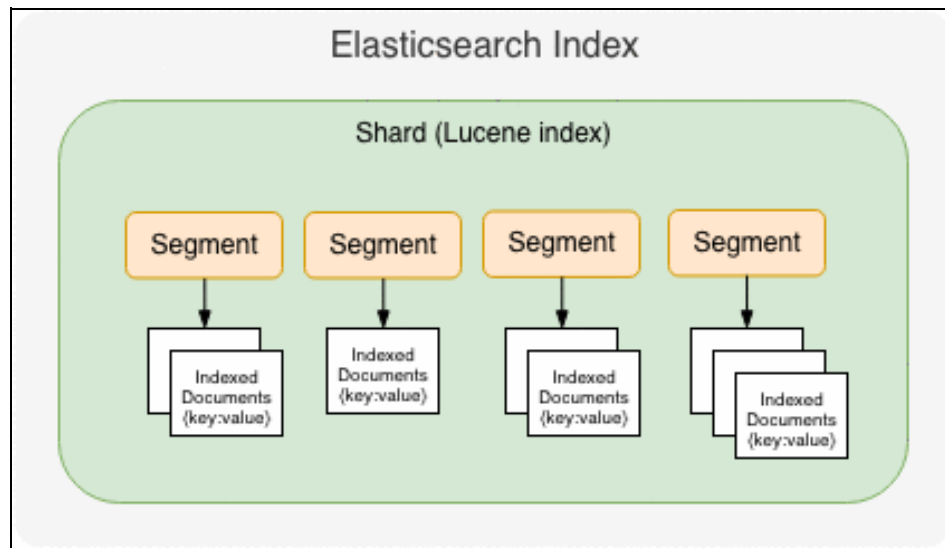


Figure 5-4 Anatomy of an Elasticsearch index

Elasticsearch also provides the capability to replicate shards, so 'primary' and 'replica' shards are spread across the data nodes in the cluster. If a data node hosting a primary shard goes down, the replica is promoted to primary, thus still able to serve search queries.

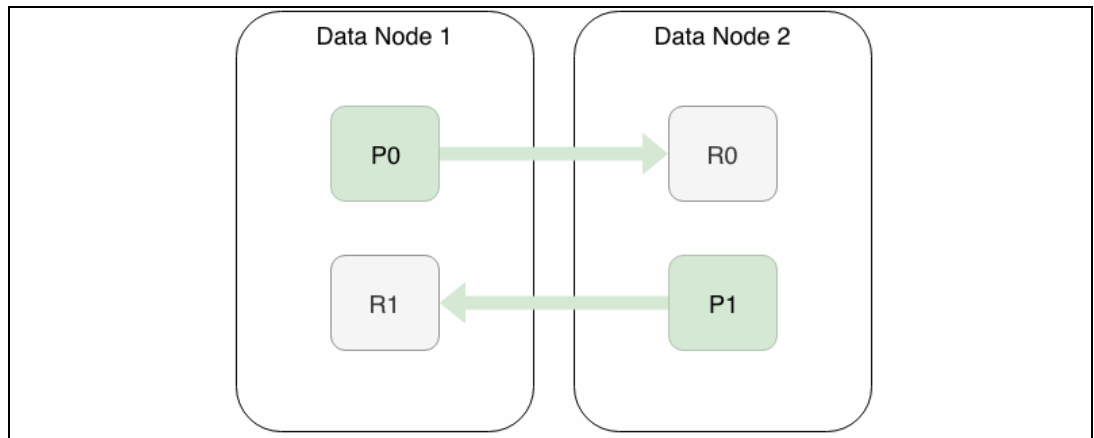


Figure 5-5 ELK primary-replica spread

For more information about the individual concepts, see the Elasticsearch documentation at https://www.elastic.co/guide/en/elasticsearch/reference/6.2/_basic_concepts.html with other useful articles available at <https://www.elastic.co/guide/en/elasticsearch/guide/current/inverted-index.html> and <https://www.elastic.co/blog/found-elasticsearch-from-the-bottom-up>

IBM Cloud Private, by default, will set 5 shards per index and 1 replica per shard. This means that in a cluster with 300 indices per day the system will, at any one time, host 3000 shards (1500 primary + 1500 replicas). To verify, in Example 5-2 an index is used to check the number of shards, replicas and the index settings.

Example 5-2 Verifying number of shards per index

```

#Get index settings
GET /logstash-2019.03.03/_settings

{
  "logstash-2019.03.03": {
    "settings": {
      "index": {
        "creation_date": "1551791346123",
        "number_of_shards": "5",
        "number_of_replicas": "1",
        "uuid": "xneG-iaWRiuNWHNH6osL8w",
        "version": {
          "created": "5050199"
        },
      },
      "provided_name": "logstash-2019.03.03"
    }
  }
}

#Get the index
GET _cat/indices/logstash-2019.03.03

health status index          uuid                pri rep docs.count
docs.deleted store.size pri.store.size
  
```

```
green open logstash-2019.03.03 xneG-iaWRiuNWHNH6osL8w 5 1 1332
0      2.8mb          1.4mb
```

```
#Get the shards for the index
GET _cat/shards/logstash-2019.03.03
```

index	shard	prirep	state	docs	store	ip	node
logstash-2019.03.03	3	p	STARTED	251	363.3kb	10.1.25.226	logging-elk-data-1
logstash-2019.03.03	3	r	STARTED	251	319.3kb	10.1.92.13	logging-elk-data-0
logstash-2019.03.03	2	p	STARTED	219	294.8kb	10.1.25.226	logging-elk-data-1
logstash-2019.03.03	2	r	STARTED	219	225.1kb	10.1.92.13	logging-elk-data-0
logstash-2019.03.03	4	p	STARTED	263	309.4kb	10.1.25.226	logging-elk-data-1
logstash-2019.03.03	4	r	STARTED	263	271.2kb	10.1.92.13	logging-elk-data-0
logstash-2019.03.03	1	p	STARTED	230	599.1kb	10.1.25.226	logging-elk-data-1
logstash-2019.03.03	1	r	STARTED	230	324.9kb	10.1.92.13	logging-elk-data-0
logstash-2019.03.03	0	p	STARTED	216	283.7kb	10.1.25.226	logging-elk-data-1
logstash-2019.03.03	0	r	STARTED	216	340.8kb	10.1.92.13	logging-elk-data-0

Each time Logstash sends a request to Elasticsearch, it will create a new index if it does not exist, or the existing index will be updated with additional documents. The Elasticsearch data pod is responsible for indexing and storing data, so during this time the CPU utilization, memory consumption and disk I/O will increase.

5.2.3 Default logging configuration

Attention: Throughout this chapter, there are references to using the `ibm-icplogging-2.2.0` Helm chart for `helm upgrade` or `helm install` commands. This chart can be used in a variety of ways, but the examples in this chapter use a locally stored copy of the Helm chart. You can retrieve this by using the following methods:

1. Use `wget --no-check-certificate https://mycluster.icp:8443/mgmt-repo/requiredAssets/ibm-icplogging-2.2.0.tgz` to download the file locally, replacing `mycluster.icp` with your cluster name.
2. Add the `mgmt-charts` repository to your local Helm repositories by using `helm repo add icp-mgmt https://mycluster.icp:8443/mgmt-repo/charts --ca-file ~/.helm/ca.pem --key-file ~/.helm/key.pem --cert-file ~/.helm/cert.pem`. Replace `mycluster.icp` with your cluster name. The chart can then be referenced using `icp-mgmt/ibm-icplogging --version 2.2.0` in place of the `ibm-icplogging-2.2.0.tgz` file. Helm should be configured to access the cluster.

The default logging configuration is designed to be a baseline and it provides the minimum resources required to effectively run a *small* IBM Cloud Private cluster. The default resource limits are not the ‘production ready’ values and therefore the Cluster Administrator should thoroughly test and adjust these settings to find the optimal resource limits for the workloads that will be running on the production environment. IBM Cloud Private Version 3.1.2 logging installs with the following resource limits by default (See Table 5-1).

Table 5-1 Default ELK resource limits

Name	CPU	Memory
client	-	1.5GB (1GB Xmx/Xms)
master	-	1.5GB (1GB Xmx/Xms)
data	-	3GB (1.5GB Xmx/Xms)

Name	CPU	Memory
logstash	-	1GB (512MB Xmx/Xms)
filebeat	-	-

Tip: Some users experience high CPU utilization by the java processes on the host. This is due to no limit specified on the containers, allowing them to consume all the available host CPU, if necessary. This is intentional and setting limits may impact the ELK stack stability. It is worth noting that high CPU utilization may be an indication of memory pressure due to garbage collection and should be investigated.

100 GB of storage via a LocalVolume PersistentVolume (PV) is allocated to the Elasticsearch data pods, which resides at `/var/lib/icp/logging` on the management node filesystem. Each PV has affinity in place so that the PV is bound to one management node only, to ensure consistency across data nodes. Each Elasticsearch data node also has affinity rules in place so that only one data pod runs on one management node at any one time.

The size of the Elasticsearch cluster deployed in an environment entirely depends on the number of management nodes in the cluster. The default number of Elasticsearch master and data pods are calculated based on the available management nodes and take on the following rules:

- ▶ One Elasticsearch data pod per IBM Cloud Private management node
- ▶ Number of Elasticsearch master pods is equal to number of management nodes
- ▶ One Logstash pod per IBM Cloud Private management node
- ▶ One Elasticsearch client pod per IBM Cloud Private management node

Elasticsearch client and Logstash replicas can be temporarily scaled as required using the default Kubernetes scaling methods. If any scaling is permanent, it's recommended to use the Helm commands to update the number of replicas.

Data retention

The default retention period for logs stored in the platform ELK stack is 24 hours. A curator is deployed as a CronJob that will remove the logstash indices from Elasticsearch every day at 23:30 UTC. If Vulnerability Advisor is enabled, another CronJob runs at 23:59 UTC to remove the indices related to Vulnerability Advisor older than 7 days.

Modifying the default retention period without proper capacity planning may be destructive to the ELK stack. Increasing the retention period will increase the resources required to search and store the data in Elasticsearch, so ensure the cluster has the required resources to be able to do so. For more information about resource allocation, see "Capacity planning" on page 164.

Configuring data retention during installation

It's possible to specify the default data retention period before installing IBM Cloud Private by adding the curator configuration to the `config.yaml`. Use Example 5-3 to set a default index retention of 14 days.

Example 5-3 Curator config in config.yaml

```
logging:
  curator:
    schedule: "30 23 * * *"
```



```
app:
  unit: days
  count: 14
```

Configuring data retention after installation

The recommended way to permanently increase the default retention period is to use a `helm upgrade` command, passing the new value as a parameter. This ensures that any future chart or IBM Cloud Private upgrades do not overwrite the value with the default value used during chart installation. To update the retention period to 14 days using `helm upgrade` from the command line, use **`helm upgrade logging ibm-icplogging-2.2.0.tgz --reuse-values --recreate-pods --set curator.app.count=14 --force --no-hooks --tls`**

For testing purposes, the default retention period is easily customized by modifying the `logging-elk-elasticsearch-curator-config` ConfigMap. To modify the retention period from 1 day to 14 days edit the `logging-elk-elasticsearch-curator-config` ConfigMap and modify the `unit_count` in the first action named `delete_indices`. The result should look similar to Example 5-4.

Example 5-4 Curator modified configuration

```
actions:
  1:
    action: delete_indices
    description: "Delete user log indices that are older than 1 days. Cron
schedule: 30 23 * * *"
    options:
      timeout_override:
      continue_if_exception: True
      ignore_empty_list: True
      disable_action: False
    filters:
      - filtertype: pattern
        kind: prefix
        value: logstash-
      - filtertype: age
        source: name
        direction: older
        timestring: '%Y.%m.%d'
        unit: days
        unit_count: 14
```

After saving and closing the file, the new curator configuration will be automatically reloaded and indices will be retained for 14 days.

Also within this ConfigMap, cleanup actions are provided for the Vulnerability Advisor indices.

5.2.4 ELK security

The platform ELK is deployed with mutual TLS security enabled by default, using Search Guard to provide PKI. It's possible to disable this during IBM Cloud Private installation by adding the following to the `config.yaml`

```
logging:
  security:
    enabled: false
```

If you already have a valid X-Pack license, you can use the X-Pack security features instead, by adding the following to the `config.yaml` at installation time

```
logging:
  security:
    provider: xpack
```

The Certificate Authority (CA) is created during installation, but the IBM Cloud Private installer offers the capability to supply your own CA that will be used for all other certificates in the cluster. For more information, see

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/installing/create_cert.html

5.2.5 Capacity planning

Capacity planning for Elasticsearch is one of the most over-looked topics and one of the most common causes of a failing Elasticsearch cluster. Lack of resources is usually the root-cause of failures at cluster installation, when audit logging is enabled or during random surges of log traffic. Allocating sufficient resources towards the capture, storage and management of logging and metrics is crucial, especially under stressful conditions. No universally, cost-effective recommendation for the capture, storage and management of logs and metrics is available, but this section provides some insights based on observations of workload behavior in IBM Cloud Private.

The default configuration should not be relied upon to meet the needs of every deployment. It is designed to provide a baseline that the Cluster Administrator should use as a starting point to determine the resources required for their environment. There are a number of factors that affect the logging performance, mostly centered around CPU and memory consumption. Elasticsearch is based on Lucene, which uses Java, and therefore has a requirement for sufficient memory to be allocated to the JVM heap. The entire JVM heap is assigned to Lucene and the Lucene engine will consume all of the available memory for its operations, which can lead to out-of-memory errors if the heap size is not sufficient for all the indexing, searching and storing that the Lucene engine is trying to do. By default, there are no CPU limits on the Elasticsearch containers, as the requirements vary depending on workload. On average, the CPU load is generally low, but will significantly rise during active periods where heavy indexing or searches are taking place and could consume the entire available CPU from the host. Restricting the CPU usage to only a few cores will create too much of a backlog of logs to process, increasing the memory usage and ultimately resulting in an unusable Elasticsearch cluster during this time. Therefore, it is almost impossible to predict the required resources for every use case and careful analysis should be made in a pre-production environment to determine the required configuration for the workload that will be running, plus additional margins for spikes in traffic.

It is not uncommon for the logging pods to be unresponsive immediately after installation of an IBM Cloud Private cluster, especially when left at the default configuration. In a basic installation, whilst all pods are starting up, they are producing hundreds of logs per second that all need to be catered for by the logging pods, which are also trying to start up at the same time. During cluster installation, an observed average across several installations by the development team was around 1500 messages per second and it takes around 30 minutes for the logging platform to stabilise with the normal rate of 50-100 messages per second in a lightly active cluster. When Vulnerability Advisor is enabled, the rate during installation can rise to observed rate of around 2300 messages per second, taking Elasticsearch up to 90

minutes to fully stabilise. When audit logging is enabled, the default resources are not sufficient and if audit logging will be enabled during cluster installation, it's recommended that increased resource limits for logging are applied at the same time, using the `config.yaml`.

Estimating the required CPU and memory

Estimating the CPU and memory required for any environment really comes down to one thing - experience. Without knowing what applications will be running, how much data the application produces, the rate at which data is produced etc it's difficult to know what resources are required. This information plays a valuable role in setting the values for a production cluster and these values must be determined from testing the workload and analyzing the data in a pre-production cluster. There are several tools at hand that enable analysis of the current resource consumption. Prior to analyzing how much storage is being consumed by an index, it's important to ensure that a realistic workload is deployed, preferably with the ability to test failover or other tests than can simulate a rise in log traffic, to allow visibility of the additional resource margins required.

X-Pack monitoring

IBM Cloud Private logging comes with a trial license for X-Pack enabled by default, but the trial functionality is not enabled during deployment. The trial is aimed at users who need more advanced capabilities that may eventually need to purchase the full X-Pack license. Information about X-Pack can be found at <https://www.elastic.co/guide/en/x-pack/current/xpack-introduction.html> as it is not covered in this chapter. However, for the purpose of estimating the logging requirements, the X-Pack monitoring can be enabled.

To enable the X-Pack monitoring, if it is not already enabled at installation time, use the **helm upgrade** command.

The logging Helm chart is located in a Helm repository called `mgmt-charts`. Instead of adding the `mgmt-charts` repository to the local machine, the URL of the chart can be used instead.

Run the **helm upgrade** command and set the `xpack.monitoring` value to `true`. You'll need to pass the default installation values in this command too, found in the cluster installation directory.

```
helm upgrade logging ibm-icplogging-2.2.0.tgz --reuse-values --recreate-pods --set xpack.monitoring=true --force --tls
```

Attention: Using `helm upgrade` can result in some down time for the logging service while it is replaced by new instances, depending on what resources have changed. Any changes made to the current logging configuration that was not changed through Helm will be lost. In some cases, the cluster may be unresponsive and requires initialization by `sgadmin`. This is a known issue, so if it occurs follow the IBM Knowledge Center steps at https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/getting_started/know_issues.html.

Restriction: The `helm upgrade` command may not run without first deleting the `logging-elk-kibana-init` job. The use of `--force` in the above command typically removes the need for this, but if you receive an error related to the `logging-elk-kibana-init` job, delete this prior to running **helm upgrade** using `kubectl -n kube-system delete job logging-elk-kibana-init`.

Once the upgrade is complete, the monitoring data is now available in Kibana. The monitoring dashboards present a wealth of information about the Elasticsearch cluster, including valuable information about how much log data is running through it.

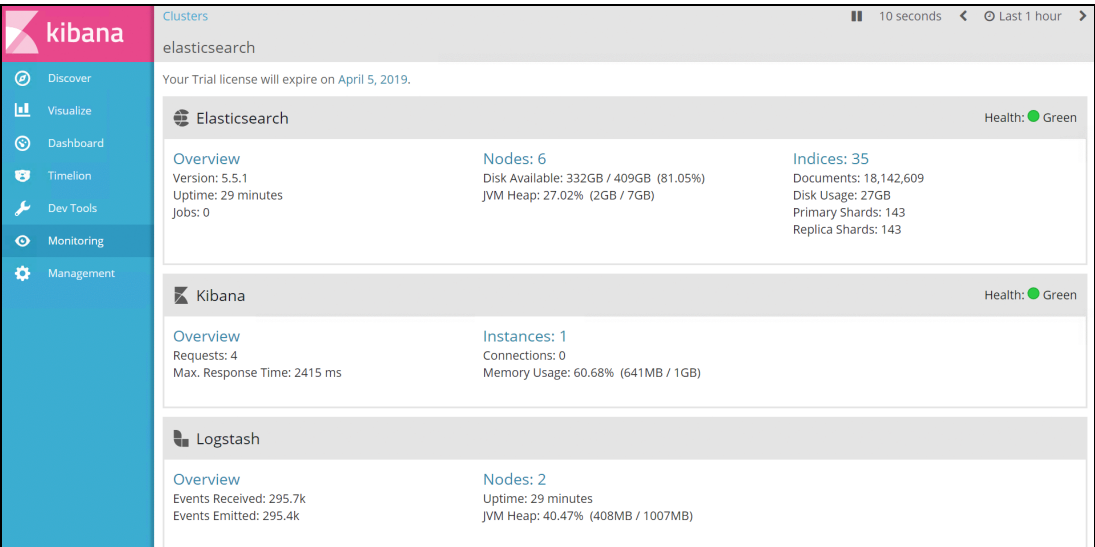


Figure 5-6 Kibana monitoring

Navigate to the **Monitoring** →**Elasticsearch (cluster)** →> **Nodes** section. All the nodes that make up the current Elasticsearch cluster are displayed, along with extremely useful data about JVM memory consumption and CPU usage per node. Assuming that the workload in the current cluster is realistic, it's more clear to see if the current resource allocation for the management node CPU cores, or the JVM heap, is enough to handle the current workload plus additional spikes.

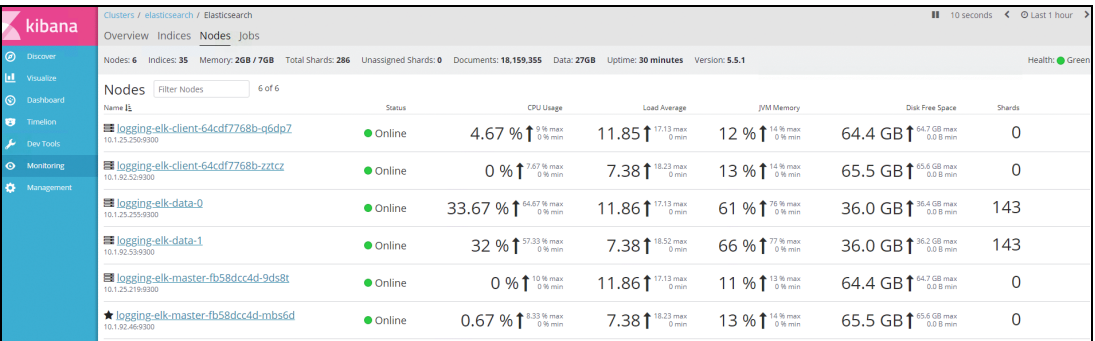


Figure 5-7 Monitoring dashboard for Elasticsearch nodes

Currently, in the above cluster, JVM memory usage for the data nodes is around 60% which is suitable for the current environment, plus a 40% overhead to handle temporary excess log data. If this value was around 80% or 90% consistently, it would be beneficial to double the allocated resources in the logging-elk-data StatefulSet. The current data nodes' CPU usage is around 30% and the maximum value the nodes have hit is around 60%, so for the current workload, the CPU cores assigned to the management nodes are sufficient. In all cases, these values should be monitored for changes and adjusted as required to keep the logging services in a stable state. High CPU usage is an indicator that there is memory pressure in the data nodes due to garbage collection processes.

Navigate to the **Monitoring** → **Elasticsearch** → **Indices** section. Here, all the currently stored indices in the system are displayed and gives a good overview of what storage is taken up by each index.

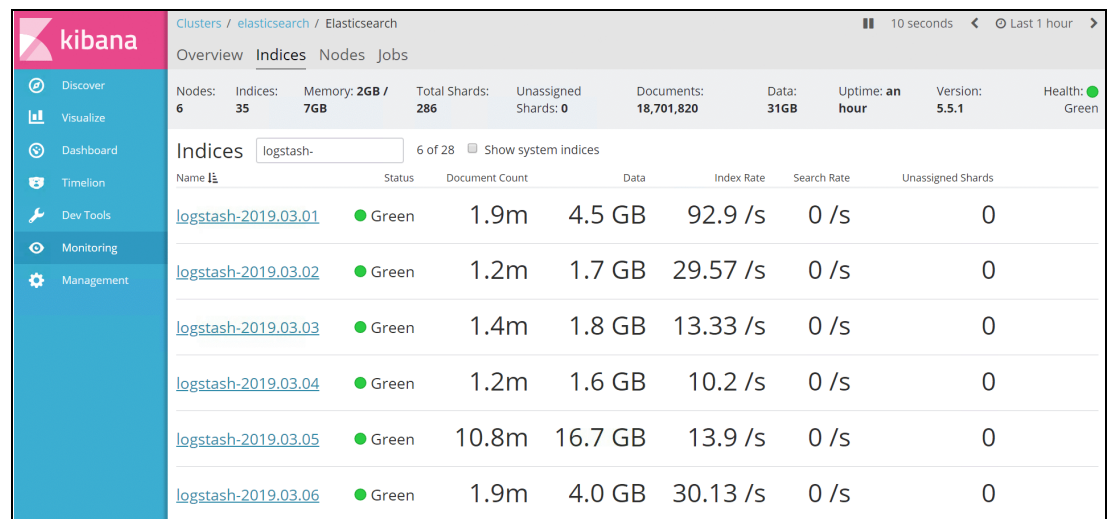


Figure 5-8 Monitoring indices list

Note that this may only be the current and previous day's indices due to the curator cleaning up indices older than the default 24 hours (unless configured differently). Selecting the previous day's index will provide information about how much storage the index is using, the rate at which data is indexed, growth rate over time and other metrics. Assuming that these values are realistic representations of the actual log data the production workloads would produce, it becomes easier to realize whether or not the current storage is sufficient for the length of time the data should be retained. Figure 5-8 shows that 6 days worth of data retention has yielded around 30GB of storage, so the default value of 100GB is sufficient, for now.

The advanced dashboard provides further analytics and insights in to some key metrics that enable better fine tuning of the cluster. Tuning Elasticsearch for better performance during indexing and querying is a big subject and is not covered in this book. More information about tuning Elasticsearch can be found in the Elasticsearch documentation at <https://www.elastic.co/guide/en/elasticsearch/reference/5.5/general-recommendations.html>.

Prometheus monitoring

Elasticsearch exposes metrics to Prometheus, which can be viewed using a default Grafana dashboard (Figure 5-9 on page 168), or queried using the Prometheus User Interface (UI). An *Elasticsearch dashboard* is provided by the development team out-of-the-box with IBM Cloud Private Version 3.1.2 to easily view various metrics about the Elasticsearch cluster, similar to those provided by the X-Pack monitoring.

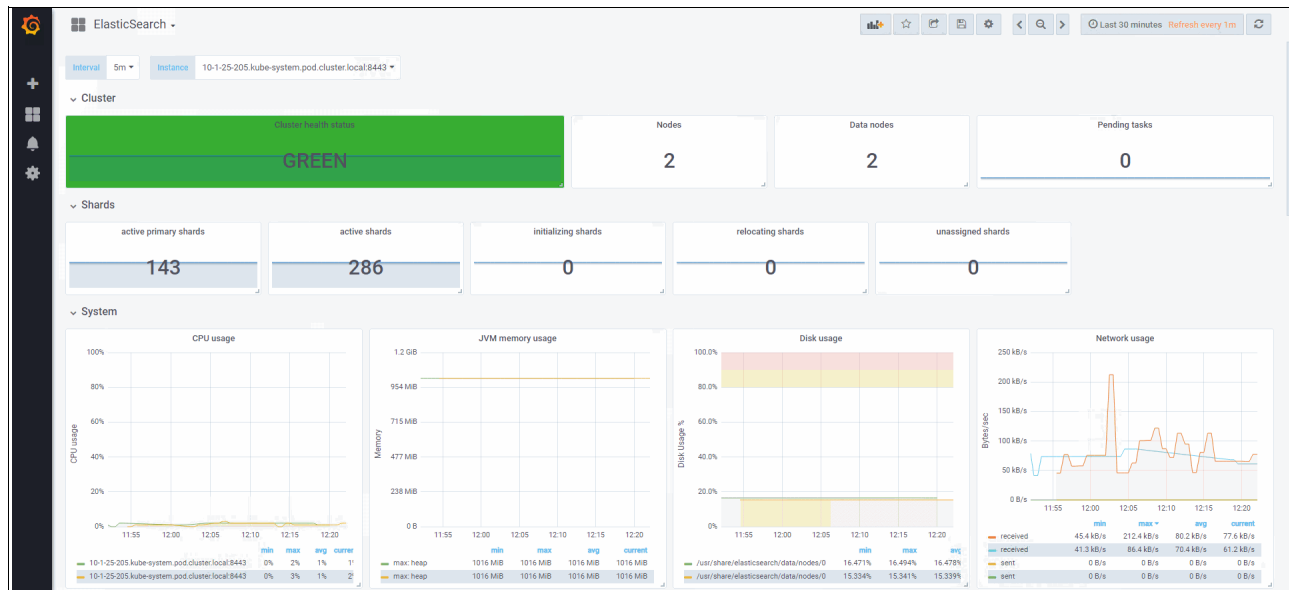


Figure 5-9 Grafana Elasticsearch dashboard

The Prometheus UI is also available, but is used to submit Prometheus search queries and return the raw results, with no additional layers on top. Access the Prometheus UI at <https://<icp-cluster-ip>:8443/prometheus> and this should display the graph page allowing you to input queries and view the results as values or a time-series based graph.

There are numerous metrics available to use, from both the standard Prometheus container metrics scraped by Kubernetes containers and those provided by the Elasticsearch application. As an example, start typing 'elasticsearch_' in to a search box and a list of available Elasticsearch specific metrics to use will show in the drop down list. It's worth noting that Grafana also uses these metrics to construct the Elasticsearch dashboard mentioned previously, so all the same data is available here too.

As a starting point, use the query in Example 5-5 to view the current memory usage as a percentage of all master, data, client and logstash pods. Copy the query to the search box, select **Execute** and then **Graph**, to see the time-series data graphically.

Example 5-5 Prometheus query for Elasticsearch pod memory consumption %

```
sum(round((container_memory_working_set_bytes{pod_name=~"logging-elk-.",container_name=~"es-.|logstash"} /
container_spec_memory_limit_bytes{pod_name=~"logging-elk-.",container_name=~"es-.|logstash"})*100)) by (pod_name)
```

In the current environment, this produces the output in Figure 5-10 on page 169.



Figure 5-10 Prometheus graph

Here we can see the overall memory consumption for the whole container, not just the JVM heap usage. The query in Example 5-6 also displays the current memory usage in GB.

Example 5-6 Prometheus query for Elasticsearch pod memory consumption GB

```
sum(container_memory_working_set_bytes{pod_name=~\"logging-elk-.\",container_name=~\"es-+|logstash\"}) by (pod_name)
```

Example 5-7 shows a query for viewing the average rate of the current CPU usage of the same Elasticsearch containers, also providing valuable information about how accurate the CPU resource allocation is in this cluster.

Example 5-7 Prometheus query for Elasticsearch pod CPU consumption

```
sum(rate(container_cpu_usage_seconds_total{pod_name=~\"logging-elk-.\",container_name=~\"es-+|logstash\"}[2m])) by (pod_name)
```

Figure 5-11 on page 170 shows the output of this query for this cluster.

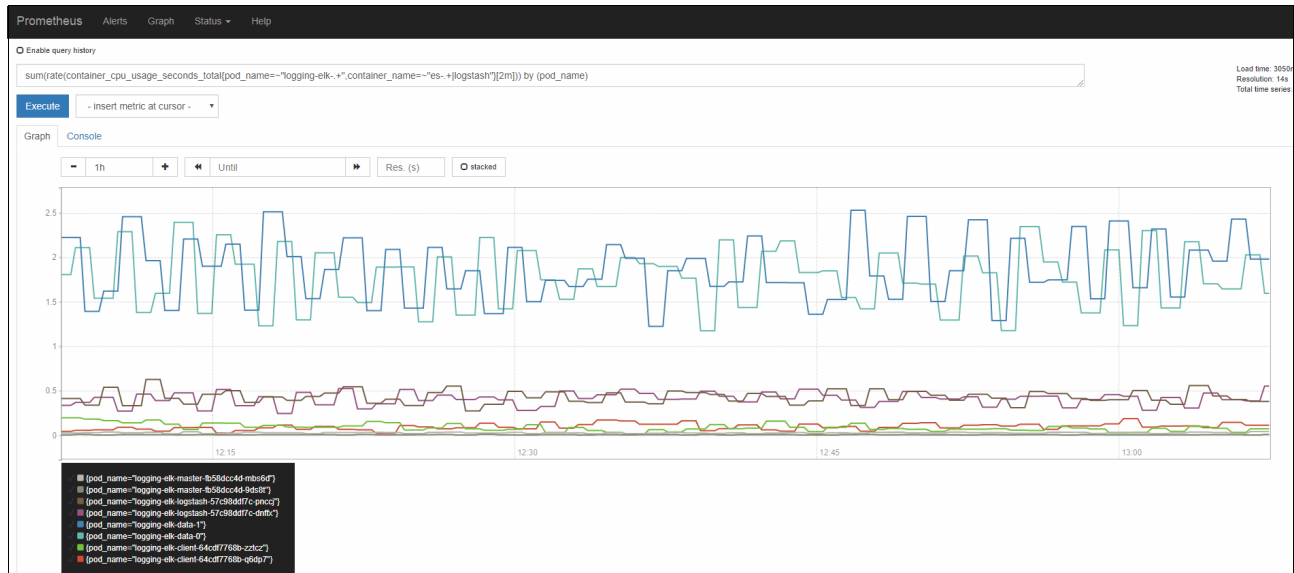


Figure 5-11 CPU usage of the Elasticsearch containers

The `container_cpu_usage_seconds_total` metric used here contains the total amount of CPU seconds consumed by container, by core. The output represents the number of cores used by the container, so a value of 2.5 in this example would mean 2500 millicores in terms of Kubernetes resource requests and limits.

Estimating the required storage capacity

The default storage value is 100GB, which is typically sufficient for normal cluster operations, however this should scale with the amount of data that applications are expected to generate plus an additional margin for spikes and multiplied by the number of days the data is retained.

When planning the amount of storage required, use the following rule:

$$(\text{Total index GB}) \times (\text{retention period}) = (\text{GB per day})$$

In this equation, there is an unknown; the Total index GB, which is the sum of the storage consumed for each index. Without this information, it's difficult to estimate how much storage to allocate to each data node. However, using the same monitoring methods from the previous sections it's possible to see the current storage usage for each index and therefore making it possible to measure the amount of storage capacity used on a daily basis. In the default Elasticsearch deployment in IBM Cloud Private 3.1.2, there is only one index configured to store logging data, but if audit logging and Vulnerability Advisor is enabled there are additional indices to cater for, each varying in size.

As an example, based on the information gathered by the X-Pack monitoring dashboard, for the `logstash-2019.03.05` index, it takes up ~18GB storage space. This includes the replicas, so the primary shards consume 9GB and the replica shards also consume 9GB. Depending on the configuration (e.g. HA) this will be spread across management nodes. Typically, if there is only one management node, then use the 9GB value as replica shards will remain unassigned. Therefore, retaining 14 days worth of logging data (using this as an average) requires a minimum of ~126GB. This figure however, does not represent a real production value, only that of a minimally used lab environment. In some cases, a single days worth of logging data may be around 100GB for a single index. Therefore in a typical cluster with 2 management nodes, 50GB per management node per day of retention is required. In a use

case where 14 days of retention is required, each management node would need 700GB each at a minimum to simply retain the log data.

Similar methods can be used by using the Kibana UI and executing API commands directly to Elasticsearch to gather data on the storage used per index. For example, the storage size on disk can be retrieved using `GET _cat/indices/logstash-2019.03.05/`, producing the output in Example 5-8.

Example 5-8 API response

health	status	index	uuid	pri	rep	store.size	pri.store.size
green	open	logstash-2019.03.05	qgH-QK35QLiBSmGHe0eRWw	5	1	18gb	9gb

There does appear to be some correlation between the storage capacity and the data node resource consumption during idle indexing. A simple advisory rule is to set the data node memory to around 15% of the storage capacity required on disk. So for example, storing 100GB of log data would mean setting the data node memory limits to around 15G. So in practice, this would be 8GB JVM Xmx/Xms and 16GB container memory for each data node.

Based on the examples in this section, the daily index requires around 18GB of storage per node, per day, and the observed memory usage of the data nodes is around 2.5GB, which is roughly 14%.

In another cluster, similar behavior was observed where the index storage size was at 61GB for the current day and the observed memory consumption for the data node (configured with 6gb JVM Xmx/Xms and 12GB container memory) was 8.1GB, so the memory usage was around 13%.

This is just a baseline to help start with a simple estimate and the ‘15%’ value is likely to increase if users perform a lot of queries on large sets of data.

JVM heap sizing

A common question when deciding on the memory requirements for an Elasticsearch data node is how much JVM heap to allocate in proportion to the amount of memory given to the whole container. Whilst it may seem like something to overlook, the JVM heap size plays an extremely important role. Elasticsearch is built on the Java programming language and when the node starts up, the heap is allocated to the Java processes. Java uses this heap to store data in-memory, enabling faster data processing. Lucene on the other hand is designed to use the host Operating System (OS) for caching data, so allocating too much heap leaves the OS and Lucene without enough memory to function correctly, which can cause out-of-memory (OOM) errors and resulting in an OOM Killed signal from Kubernetes to restart the container.

When setting the heap size, the following rules should be followed:

- ▶ JVM heap should be no more than 50% of the total container memory
- ▶ JVM heap should not exceed 32GB (which means the maximum size for an Elasticsearch node is 64GB memory)

More information about JVM heap sizing can be found at <https://www.elastic.co/guide/en/elasticsearch/guide/master/heap-sizing.html>.

These rules also apply to the master, client and Logstash containers however, there is no Lucene engine running on these nodes so around 60% of the total memory can be allocated to the JVM heap.

Garbage collection

Java is a garbage-collected language, which means the JVM is in charge of allocating memory and releasing it. The garbage collection process has some ‘halt’ scenarios where the JVM halts execution of Elasticsearch so it can trace and collect dead objects in memory to release it. During this time, nothing happens within Elasticsearch, so no requests are processed and shards are not relocated. A cluster experiencing long garbage collection processes will be under heavy loads and Elasticsearch nodes will appear to go offline for brief periods of time. This instability causes shards to relocate frequently as Elasticsearch tries to keep the cluster balanced and enough replicas available, as well as increased network traffic and disk I/O. Therefore, long garbage collection processes are more obvious when Elasticsearch is slow to respond to requests and experiences high CPU utilization.

Elasticsearch reports that garbage collection is configured to start when the JVM heap usage exceeds 75% full. If the Elasticsearch nodes are constantly above 75%, the nodes are experiencing memory pressure, which means not enough is allocated to the heap. If nodes constantly exceed 85-95% the cluster is at high risk of becoming unstable with frequent response delays and even out-of-memory exceptions. Elasticsearch provides useful information about the garbage collector usage on each node using the `_nodes/stats` API. In particular, the `heap_used_percent` metric in the `jvm` section is worth looking at if you're experiencing issues to ensure it is generally below 75% during normal operation. See Example 5-9.

Example 5-9 Partial API Response for `_node/stats` endpoint

```
...
"jvm": {
  "timestamp": 1552999822233,
  "uptime_in_millis": 100723941,
  "mem": {
    "heap_used_in_bytes": 967083512,
    "heap_used_percent": 62,
    "heap_committed_in_bytes": 1556938752,
    "heap_max_in_bytes": 1556938752,
    "non_heap_used_in_bytes": 118415128,
    "non_heap_committed_in_bytes": 127152128,
    "pools": {
      "young": {
        "used_in_bytes": 244530624,
        "max_in_bytes": 429522944,
        "peak_used_in_bytes": 429522944,
        "peak_max_in_bytes": 429522944
      },
      "survivor": {
        "used_in_bytes": 5855696,
        "max_in_bytes": 53673984,
        "peak_used_in_bytes": 53673984,
        "peak_max_in_bytes": 53673984
      },
      "old": {
        "used_in_bytes": 716697192,
        "max_in_bytes": 1073741824,
        "peak_used_in_bytes": 844266000,
        "peak_max_in_bytes": 1073741824
      }
    }
  }
}
```

...

More information on garbage collection monitoring can be found in the Elasticsearch documentation at

https://www.elastic.co/guide/en/elasticsearch/guide/current/_monitoring_individual_nodes.html.

Example sizing of an Elasticsearch data node

This section will attempt to provide a base sizing for the Elasticsearch master and data nodes in a production environment based on the observations in the previous sections. Whilst the values can't really be carried over to other environments, the methodology is the same. This section assumes that workloads representing a realistic production cluster have been used to test resource consumption, as this analysis is based on that information. The following tests will use 6 WebSphere Liberty deployments, each one executing a loop to log data to stdout indefinitely, to simulate a constant log stream. At random periods, a burst of log traffic is executed by scaling 2 of the deployments temporarily to 2 replicas to simulate some event and replacement of pods. Whilst this test is performed over only one day, it will be assumed that this is the normal operational behavior and therefore provide a baseline for the Elasticsearch cluster sizing.

3 hours into load testing, the index size was monitored and logs were, on average, generated at a rate of about 0.9GB per hour. See Figure 5-12.

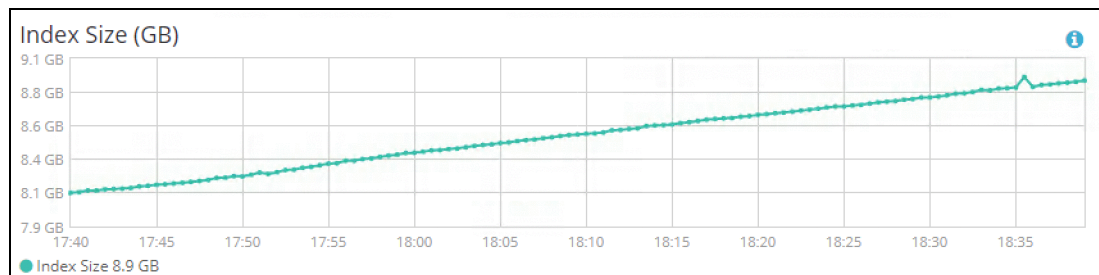


Figure 5-12 Index rate increase

Over 24 hours, this gives 21.6GB of data per day. Retaining this for 14 days gives 302GB and as this cluster is installed with two management nodes, each management node requires at least 151GB available storage. Based on this information, the memory for data nodes can also be estimated using the '15%' rule mentioned in the previous section. 15% of 21.6GB is 3.25GB, so in this example, setting the JVM Xmx/Xms to 2GB and container limit to 4GB is a good estimate.

Configuring the resources for Elasticsearch containers

The resource for the ELK stack can be provided during installation, via the `config.yaml`, or post-installation using `kubectl patch` commands. It's recommended to set the values during installation, especially if audit logging is enabled, but it does typically require existing knowledge of the resources needed for the current cluster configuration. Depending on the resources available on the management nodes, setting the resources for logging too high may hinder the successful installation of the other management services that may be enabled, so be sure to have planned the sizing of the management nodes correctly.

Configuring resources during installation

This is the recommended approach, especially if audit logging is enabled as the default configuration is too low to cater for the platform plus audit logs. To configure the logging

resources in the config.yaml, use XX as a template to set the resources for each component. The installer will then use these values during installation of the logging chart.

Example 5-10 config.yaml values for logging

```
logging:
  logstash:
    heapSize: "512m"
    memoryLimit: "1024Mi"
  elasticsearch:
    client:
      heapSize: "1024m"
      memoryLimit: "1536Mi"
    data:
      heapSize: "1536m"
      memoryLimit: "3072Mi"
      storage:
        size: "100Gi"
    master:
      heapSize: "1024m"
      memoryLimit: "1536Mi"

elasticsearch_storage_size: "100Gi"
```

Note that the additional parameter `elasticsearch_storage_size` is required in addition to when setting the Elasticsearch data node volume sizes. Additional information and values for other components such as Filebeat and the curator can be found at https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_metrics/logging_elk.html.

Configuring resources after installation

Using the `kubectl patch` command, it's simple to increase the resources for each component after they are deployed in IBM Cloud Private. Each component has a rolling update strategy, so the logging services should still be available while Kubernetes is redeploying a new node with the additional resources.

Use the below examples as templates to set the memory limits for a given component, replacing values where necessary.

Example 5-11 kubectl patch for Elasticsearch client node

```
kubectl -n kube-system patch deployment logging-elk-client --patch '{ "spec": {
"template": { "spec": { "containers": [ { "env": [ { "name": "ES_JAVA_OPTS",
"value": "-Xms2048m -Xmx2048m" } ] }, "name": "es-client", "resources": { "limits":
{ "memory": "3072M" } } } ] } } } }
```

Example 5-12 kubectl patch for Elasticsearch Logstash node

```
kubectl -n kube-system patch deployment logging-elk-logstash --patch '{ "spec": {
"template": { "spec": { "containers": [ { "env": [ { "name": "ES_JAVA_OPTS",
"value": "-Xms1024m -Xmx1024m" } ] }, "name": "logstash", "resources": { "limits": {
"memory": "2048M" } } } ] } } } }
```

Example 5-13 kubectl patch for Elasticsearch data node

```
kubectl -n kube-system patch statefulset logging-elk-data --patch '{ "spec": {
"template": { "spec": { "containers": [ { "env": [ { "name": "ES_JAVA_OPTS",
```

```
"value": "-Xms3072m -Xmx3072m" } ], "name": "es-data", "resources": { "limits": {  
"memory": "6144M" } } } ] } } } }
```

Example 5-14 kubectl patch for Elasticsearch master node

```
kubectl -n kube-system patch deployment logging-elk-master --patch '{ "spec": {  
"template": { "spec": { "containers": [ { "env": [ { "name": "ES_JAVA_OPTS",  
"value": "-Xms2048m -Xmx2048m" } ], "name": "es-master", "resources": { "limits":  
{ "memory": "4096M" } } } ] } } } } }
```

These commands are sufficient to scale the containers vertically, but not horizontally. Due to the cluster settings being integrated in to the Elasticsearch configuration, you cannot simply add and remove master or data nodes using standard Kubernetes capabilities such as the HorizontalPodAutoscaling resource.

Using `kubectl patch` should be used for testing to find a suitable value. Updating the logging resources in this way will not persist during chart or IBM Cloud Private upgrades. Permanent configuration changes can only be made using `helm upgrade`. To update the JVM and container memory for the data nodes, for example, run the upgrade:

```
helm upgrade logging ibm-icplogging-2.2.0.tgz --reuse-values --set  
elasticsearch.data.heapSize="3072m" --set elasticsearch.data.memoryLimit="6144Mi"  
--force --no-hooks --tls
```

Scaling an Elasticsearch cluster

Scaling an Elasticsearch node in IBM Cloud Private entirely depends on the type of node, the volume of logs and the performance of the Elasticsearch cluster against the logs generated. Sending thousands of smaller, simple log messages per second may have a different impact than a fewer number of larger log messages, so the data nodes may perform differently when configured as several smaller nodes or a few larger nodes. What works best in an environment generally comes down to experience by tuning the Elasticsearch cluster against your workloads to find which configuration provides the best performance for the type of log data the deployed applications are producing. Consider the following scenarios if a banking application is sending thousands of transactions and transaction logs to Elasticsearch every second:

- ▶ Logstash may be the bottle neck (assuming the host network can handle this volume of traffic anyway) and will require either more memory or additional replicas deployed to meet the high demand.
- ▶ if the transaction data generated is a very large data set with lots of fields, then the data node may be the bottle neck as it has to index such a large volume of data, so additional memory or even additional data nodes may be required.
- ▶ Logstash may also struggle with low resources and large volumes of JSON formatted data, as it will parse the JSON and bring fields to the root level.
- ▶ As the master node is responsible for tracking which data nodes all the documents are stored on, thousands of logs means the master node has to track thousands of documents which can be difficult to achieve without enough memory and CPU.
- ▶ When querying the stored data, the client node has to parse the potentially complex or time-based lucene query and lack of memory can result in request timeouts.

With the above in mind it's important to be able to scale the components as and when it is required.

- ▶ Logstash and client pods - increase the number of replicas by using Helm:

```
helm upgrade logging ibm-icplogging-2.2.0.tgz --reuse-values --set  
elasticsearch.client.replicas=3 --set logstash.replicas=3 --force --no-hooks  
--tls
```

- Master pods - run the `helm upgrade` command passing the desired number of replicas (this should ideally be the number of management nodes in the cluster, however it will still work).

```
helm upgrade logging ibm-icplogging-2.2.0.tgz --reuse-values --set  
elasticsearch.master.replicas=3 --recreate-pods --force --tls
```

- Data pods - ensure there are sufficient management nodes available. Data nodes have an affinity to management nodes and a hard anti-affinity to other data pods, so no more than one data pod will run on a single management node at any one time.

```
helm upgrade logging ibm-icplogging-2.2.0.tgz --reuse-values --set  
elasticsearch.data.replicas=3 --recreate-pods --force --no-hooks --tls
```

Impacts of audit logging

Planning for audit logging is crucial to allow the logging services to start successfully and have enough resources to handle the large volume of data being sent to the Elasticsearch cluster. This section shows the how enabling audit logging can affect a cluster and the tests were performed in a lab environment, which may differ from your results. The tests, however can provide a helpful insight in to methodology used for analyzing the cluster and deriving the resources required when enabling audit logging by using the trial X-Pack monitoring and Prometheus tools already available in IBM Cloud Private. The audit logging in this example was configured *after* installation of IBM Cloud Private, following the IBM Knowledge Center article at

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/user_management/auditing_icp_serv.html.

Whilst audit logging was being enabled, the workloads from the previous sections were left running to determine the effect of enabling audit logging has on a running cluster with an already active Elasticsearch.

Figure 5-13 on page 177 shows that the cluster became unresponsive for a short period of time whilst the main pods that generate audit logs (plus kubernetes api-server audit logging) were restarted and began generating data in the newly added audit index.

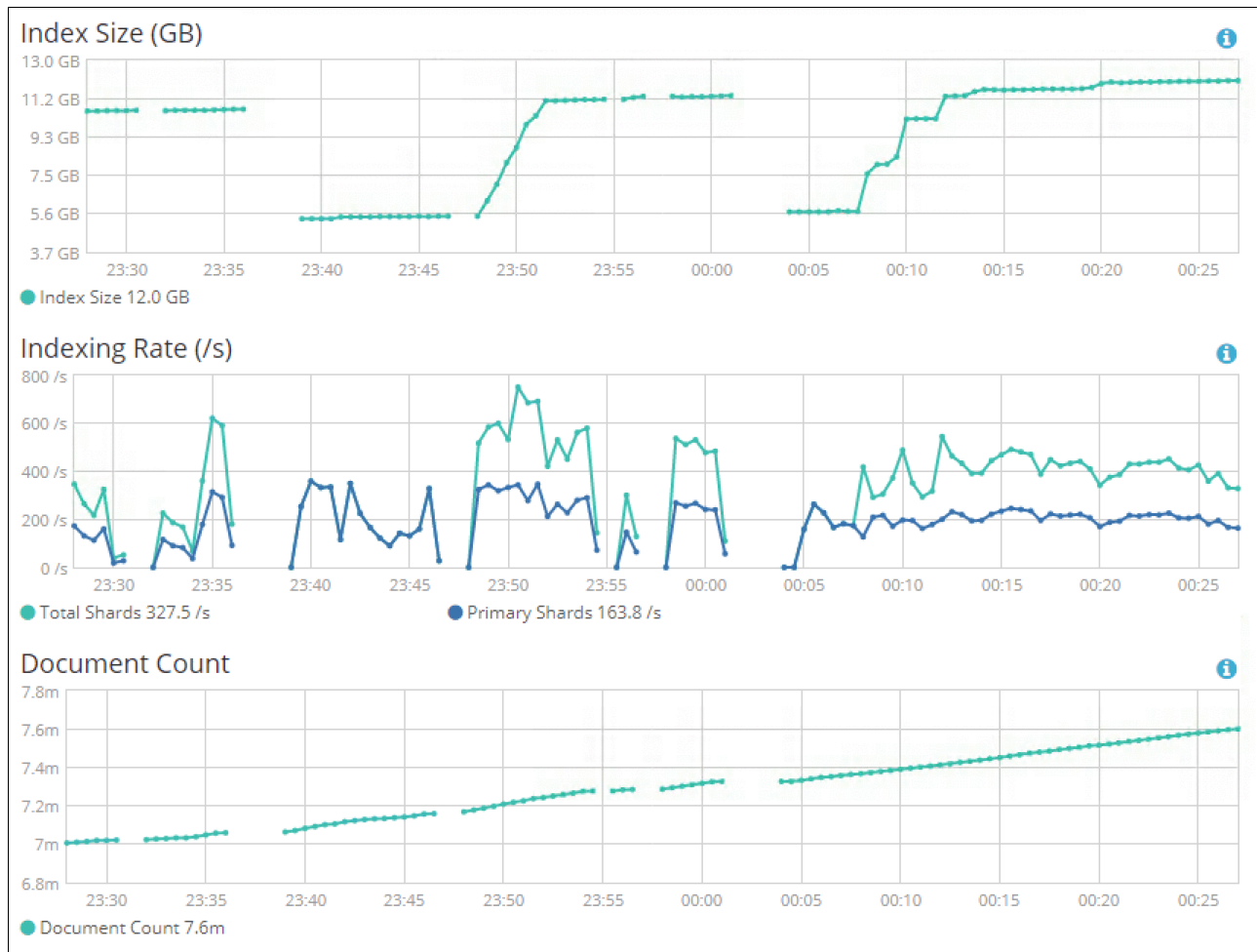


Figure 5-13 Audit log enabled

Figure 5-14 shows one of the data nodes becoming unresponsive after being overloaded due to high CPU utilization.

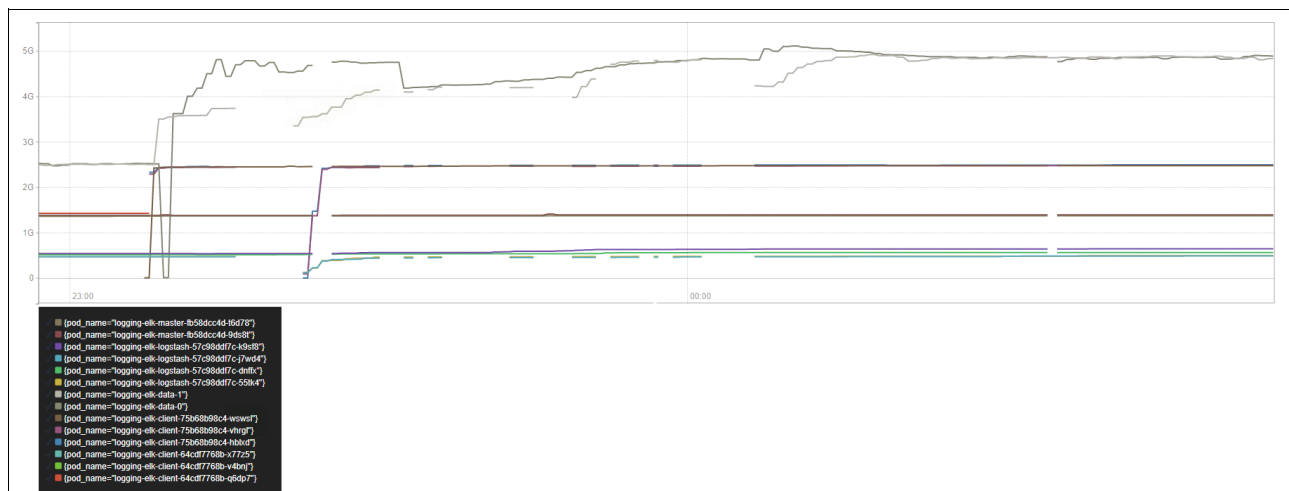


Figure 5-14 Data node becoming unresponsive

The graph of the JVM heap size on a data node shows that the value of 3GB Xmx/Xms and 6GB container memory was sufficient, as it maintained a steady maximum of around 2.3GB (Figure 5-15).

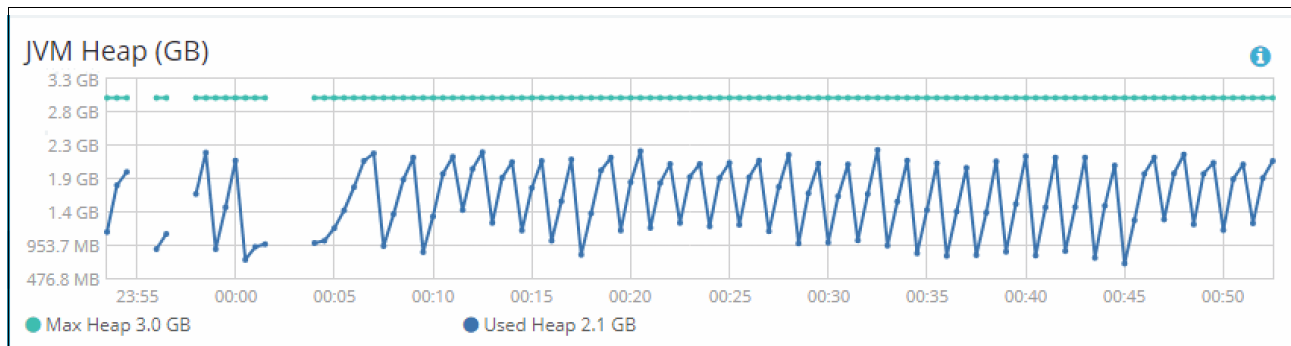


Figure 5-15 JVM heap size

Checking the total memory consumed during this time period in Prometheus also shows similar results, with the container memory peaking at around 4.5GB. See Figure 5-16.

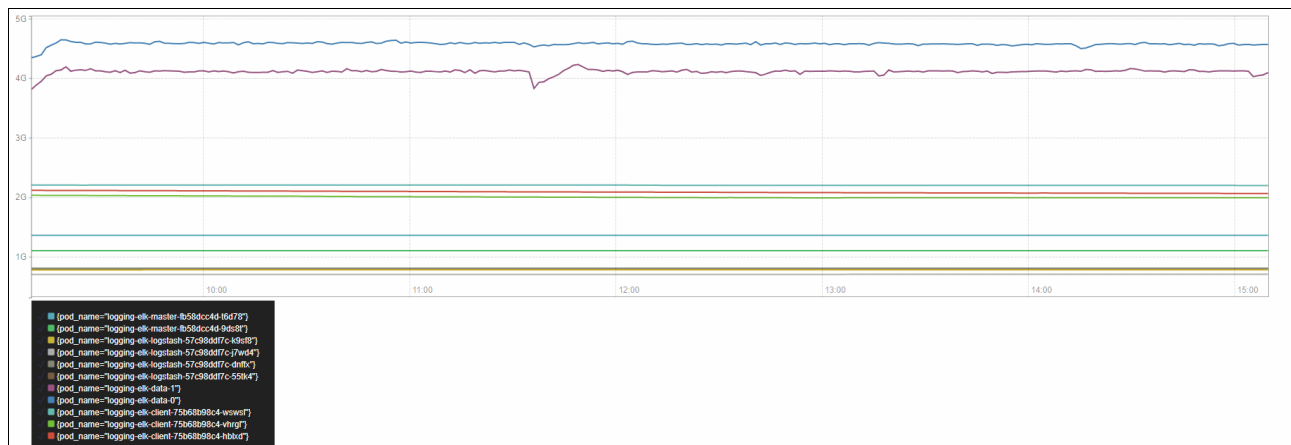


Figure 5-16 Container memory usage with audit logging

As the audit log generation uses Fluentd, data is sent directly to Elasticsearch, so there is no need to consider Logstash.

The client and master nodes generally stayed stable throughout the test. The sudden hike for one of the client nodes in Figure 5-17 on page 179 is due to a new replica starting in advance of enabling audit logging.

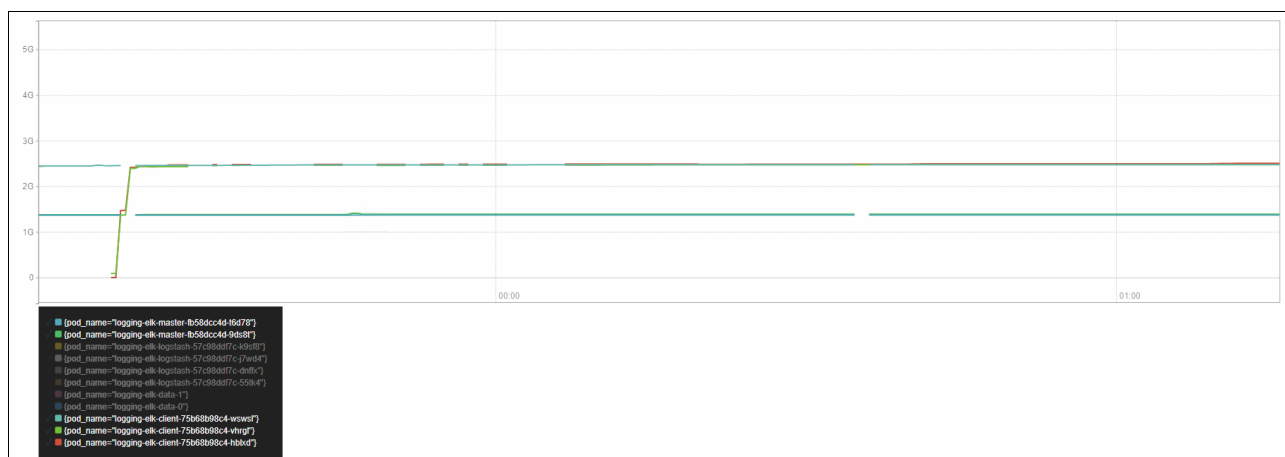


Figure 5-17 New replica starting

Based on the above results, Table 5-2 shows the minimum advised resources for a small High Availability cluster using 3 master nodes, 2 proxy nodes, 2 management nodes, 1 Vulnerability Advisor node and 3 worker nodes.

Table 5-2 Advised ELK resource limits for audit logging with high availability

Name	Number of Nodes	Memory
client	2	3GB (2GB Xmx/Xms)
master	2	1.5GB (1GB Xmx/Xms)
data	2	6GB (3GB Xmx/Xms)
logstash	2	1GB (512MB Xmx/Xms)

In smaller clusters, resources can be reduced. Audit logging was tested on a smaller, 6 node cluster (1 master, 1 proxy, 1 management and 3 workers) and the minimum resource limits in Table 5-3 are sufficient for audit logging.

Table 5-3 Minimum ELK resource limits for audit logging in small clusters

Name	Number of Nodes	Memory
client	1	3GB (2GB Xmx/Xms)
master	1	1.5GB (1GB Xmx/Xms)
data	1	4GB (2GB Xmx/Xms)
logstash	1	1GB (512MB Xmx/Xms)

In larger, more active clusters, or during installation of IBM Cloud Private with audit logging and Vulnerability Advisor enabled these values will almost certainly increase, so plan accordingly and be patient when finding the correct resource limits. During times of heavy use, it may appear as though Elasticsearch is failing, but it usually does a good job of catching up after it has had time to process, provided it has the appropriate resources. An easy mistake users often make is thinking that it's taking too long and pods are stuck, so they are forcefully restarted and the process takes even longer to recover from. During busy periods of heavy indexing or data retrieval, it's common to observe high CPU utilization on the management nodes, however this is usually an indicator that there is not enough memory

allocated to the JVM heap and it should be monitored carefully to ensure that the CPU returns to a normal idle value.

Careful monitoring of the container memory is advised and container memory should be increased if there are several sudden dips or gaps in the Prometheus graphs while the container is under heavy load, as this is typically a sign that the container is restarting. Evidence of this is in Figure 5-18 on page 180, where a data pod had reached its 3GB container limit (monitored using Prometheus) and suddenly dropped, indicating an out-of-memory failure.

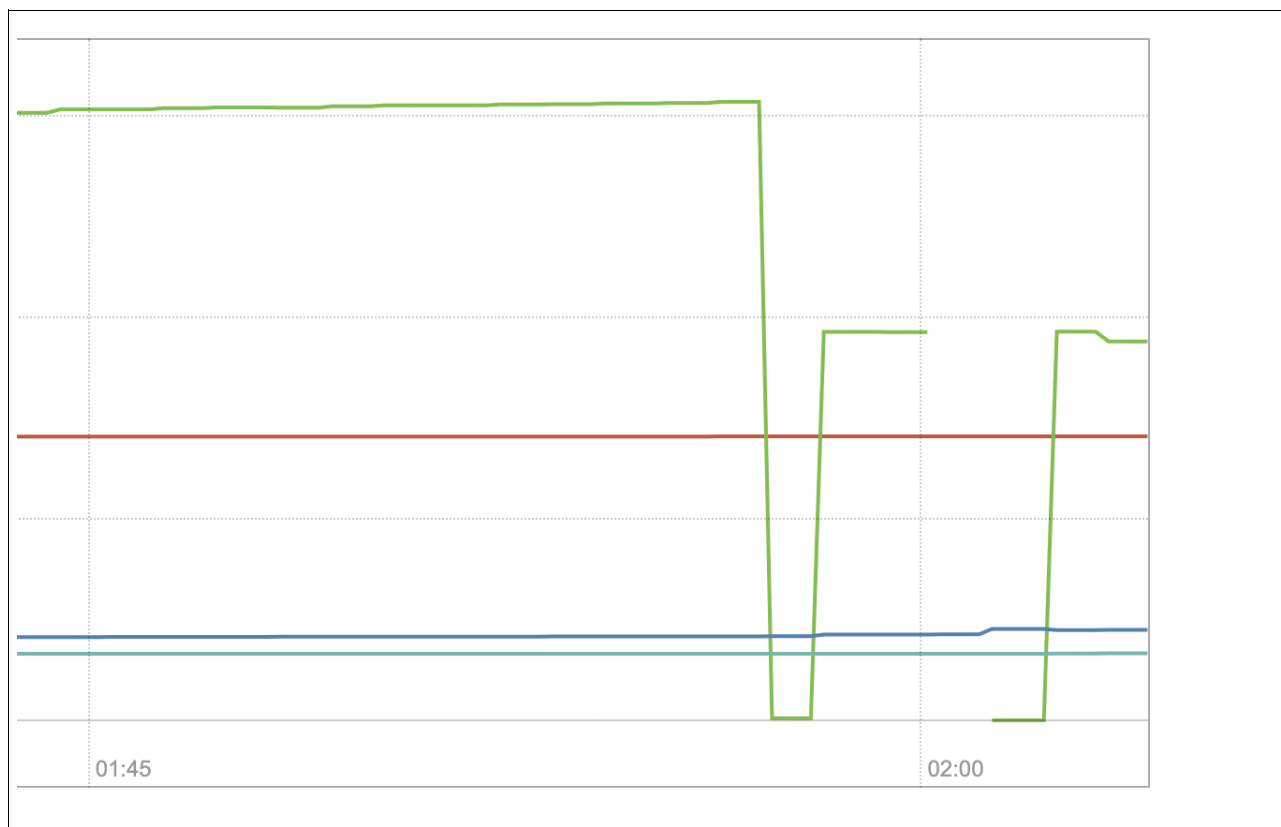


Figure 5-18 Out-of-memory failure

As a reference, Example 5-15 provides the base YAML to add to the config.yaml to set the logging resources to the above recommendations.

Example 5-15 Recommended initial logging values in the config.yaml

```
logging:
  logstash:
    heapSize: "512m"
    memoryLimit: "1024Mi"
  elasticsearch:
    client:
      heapSize: "2048m"
      memoryLimit: "3072Mi"
    data:
      heapSize: "3072m"
      memoryLimit: "6144Mi"
    master:
      heapSize: "1024m"
```

Other factors to consider

As mentioned throughout this chapter, tuning the resources for the platform ELK can take considerable effort if capacity planning has not been done for this configuration before. Despite audit logging, a range of other factors can also play a part in deciding the resources required for both the ELK stack and the management nodes hosting the components:

- ▶ Data-at-rest encryption - IBM Cloud Private provides the capability to encrypt data stored on disk using dm-crypt. This, for many Cluster Administrators, is a must-have feature to be security compliant. This does, however, mean that there may be a performance penalty in terms of CPU utilization on the management nodes hosting ELK, so plan for additional resources on the management nodes to cater for encryption. More information can be found at https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/installing/fips_encrypt_volumes.html.
- ▶ Data-in-transit encryption - IBM Cloud Private provides the capability to encrypt data traffic across the host network using IPsec. Similarly to data-at-rest encryption, this may have a performance impact on the CPU utilization. More information can be found at https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/installing/fips_encrypt_volumes.html.
- ▶ Number of nodes - the number of nodes in a cluster largely affects the performance, mainly due to the number of additional Filebeat containers all streaming data to Logstash. Consider scaling Logstash to more replicas or increasing CPU when additional worker nodes with workloads are added to the cluster.
- ▶ Storage - Elasticsearch performs better with high performance storage. It's recommended to use the highest performing storage available for more efficient indexing, storage and retrieval of data.

5.2.6 Role based access control

The platform ELK has role based access control (RBAC) built in to the deployments that will filter API responses to the Elasticsearch client pods to only return results relevant to the requesting users accessible namespaces. This means that a user that only has access to namespace1 should not see logs related to namespace2.

The RBAC modules consists of Nginx containers bundled with the client and Kibana pods to provide authentication and authorization to use the ELK stack. These Nginx containers use the following rules

1. A user with the role ClusterAdministrator can access any resource, whether audit or application log.
2. A user with the role Auditor is only granted access to audit logs in the namespaces for which that user is authorized.
3. A user with any other role can access application logs only in the namespaces for which that user is authorized.
4. Any attempt by an auditor to access application logs, or a non-auditor to access audit logs, is rejected.

The RBAC rules provide basic data retrieval control for users that access Kibana. The rules do not prevent seeing metadata such as log field names or saved Kibana dashboards. User-saved artifacts in Kibana are all saved in Elasticsearch in the same default index of `/.kibana`. This means that all users using the same instance of Kibana can access each

others saved searches and dashboards and view any other custom fields added to the data. Without an X-Pack or Search Guard Enterprise license, no other native multi-tenant features are available to address this situation in a single Kibana instance. For information about deploying multiple Kibana instances, see 5.2.8, “Management” on page 188.

5.2.7 Using Kibana

This section will cover some basic tools available in the Kibana User Interface (UI) to provide some useful functions that allow a user to view logs, create graphical representations of the data and custom dashboards.

The Kibana User Interface (UI) is split in to 6 sections:

- ▶ Discover
- ▶ Visualize
- ▶ Dashboards
- ▶ Timelion
- ▶ Dev Tools
- ▶ Management

If X-Pack functions were enabled during deployment, they will also appear in the UI.

Discover

This is the first page shown automatically when logging in to Kibana. By default, this page will display all of your ELK stack’s 500 most recently received logs in the last 15 minutes from the namespaces you are authorized to access. Here, you can filter through and find specific log messages based on search queries.

The Kibana Discover UI contains the following elements:

- ▶ **Search Bar:** Use this to search specific fields and/or entire messages
- ▶ **Time Filter:** Top-right (clock icon). Use this to filter logs based on various relative and absolute time ranges
- ▶ **Field Selector:** Left, under the search bar. Select fields to modify which ones are displayed in the Log View
- ▶ **Log Histogram:** Bar graph under the search bar. By default, this shows the count of all logs, versus time (x-axis), matched by the search and time filter. You can click on bars, or click-and-drag, to narrow the time filter

The search bar is the most convenient way to search to string of text in log data. It uses a fairly simple language structure, called the *Lucene Query Syntax*. The query string is parsed into a series of terms and operators. A term can be a single word or a phrase surrounded by double quotes which searches for all the words in the phrase, in the same order. Figure 5-19 on page 183 shows searching Kibana for log data containing the phrase “websphere-liberty”

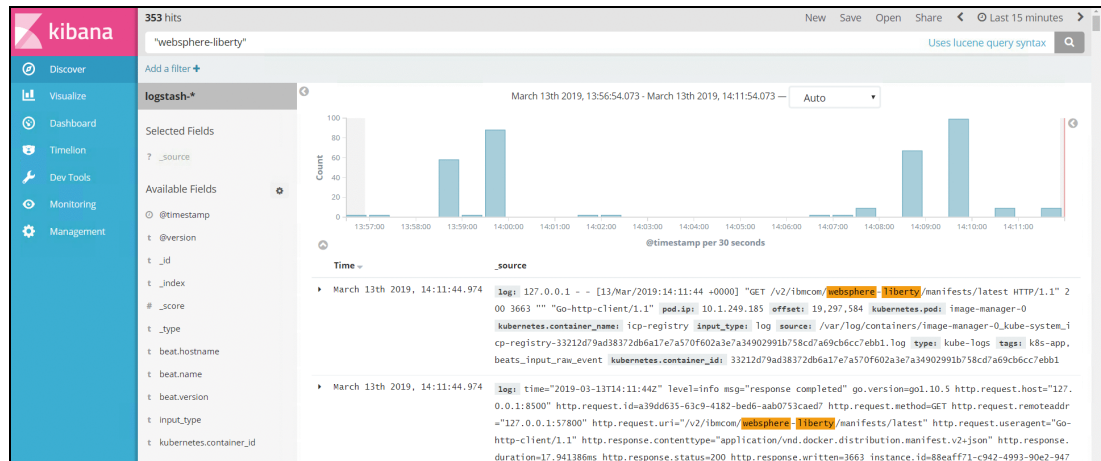


Figure 5-19 Searching logs in Kibana

Searches can be further refined, for example by searching only for references to the default namespace. Using filters enables users to restrict the results shown only to contain the relevant field filters, as shown in Figure 5-20.

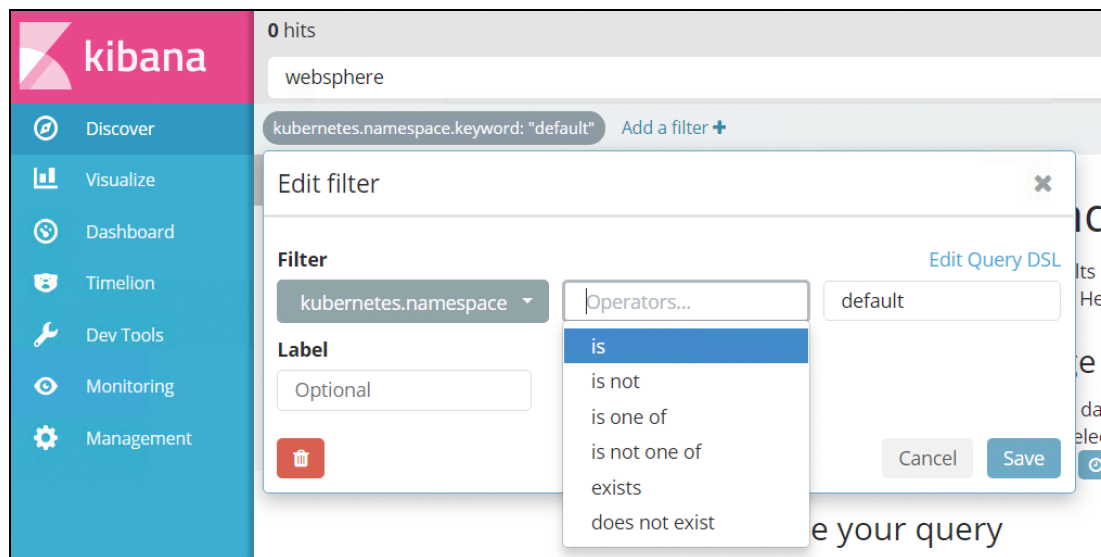


Figure 5-20 Kibana search filter

And the results are restricted to this filter only, as seen in Figure 5-21.

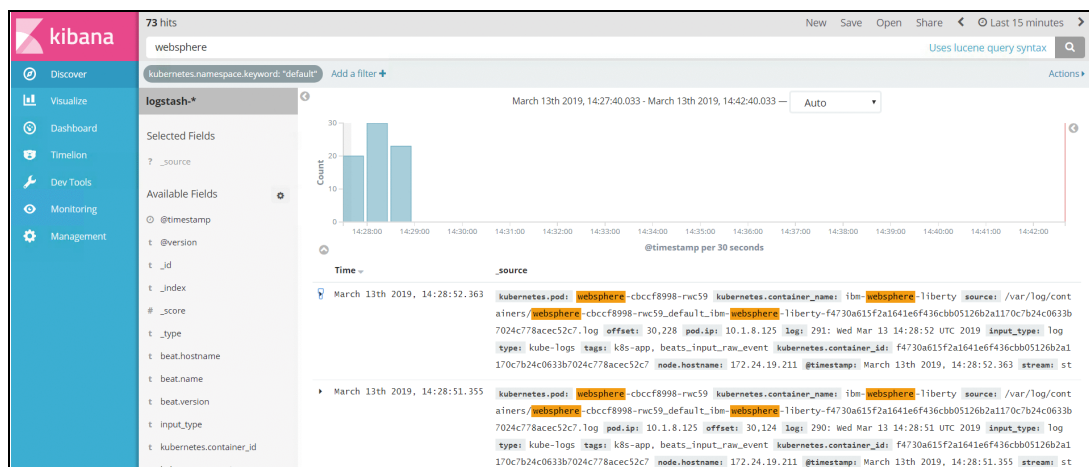


Figure 5-21 Kibana filtered search

The use of fields also allows more fine grained control over what is displayed in the search results, as seen in Figure 5-22 where the fields **kubernetes.namespace**, **kubernetes.container_name** and **log** are selected.

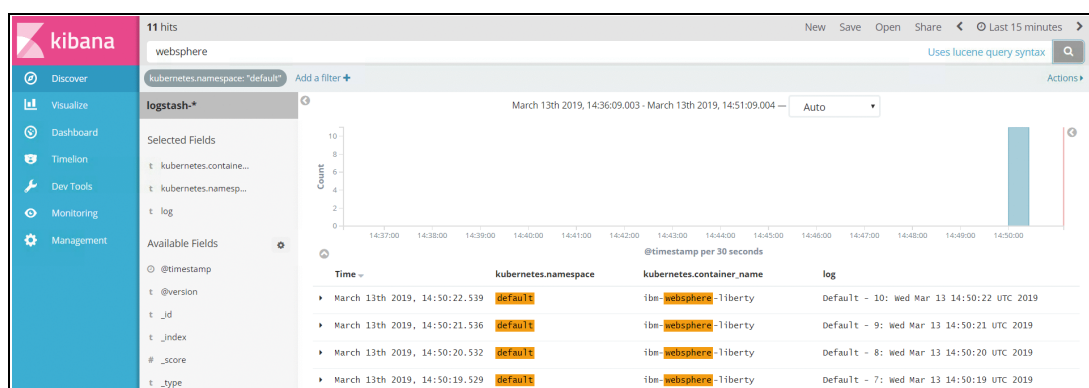


Figure 5-22 Kibana filtered search with selected fields

More detailed information is available in the Elasticsearch documentation at <https://www.elastic.co/guide/en/elasticsearch/reference/5.5/query-dsl-query-string-query.html#query-string-syntax>.

Visualize

The Visualize page lets you create graphical representations of search queries in a variety of formats, such as bar charts, heat maps, geographical maps and gauges.

This example will show how a simple pie chart is created, configured to show the top 10 containers that produce the highest number of logs per namespace from a Websphere Liberty Deployment that is running in different namespaces.

To create a visualization, select the + icon, or **Create Visualization**, if none exist, then perform the following steps

1. Select the indices that this visualization will apply to, such as `logstash`. You may only have one if the default configuration has not been modified or audit logging is not enabled.
2. Select Pie Chart from the available chart selection.

3. In the search box, enter the query that will generate the data and apply the relevant filters. This is the same as writing a query and filters in the Discover section. In this example the term 'websphere' is used to search for instances of 'websphere' in the log data.
4. In the left side pane, select the **data** tab, select **metrics**, then set slice size to **Count**.
5. In the buckets section, select **Split Slices** and set the following values:
 - a. Set **Aggregation** to **Terms**.
 - b. Set **Field** to **kubernetes.namespace.keyword**.
 - c. Set **Order By** to **metric: Count**.
 - d. Set **Order** to **Descending** and **Size** to **10**.
6. Select the **Play** icon at the top of this pane and the pie chart should display with the data from the search query. In this case, it will show the top 10 namespaces that contain the highest number of logs generated by websphere containers. You can use the filters to exclude certain namespaces, such as kube-system.

The result should look similar to Figure 5-23. Save the visualization using the **Save** link at the top of the page, as this will be used later to add to a dashboard.

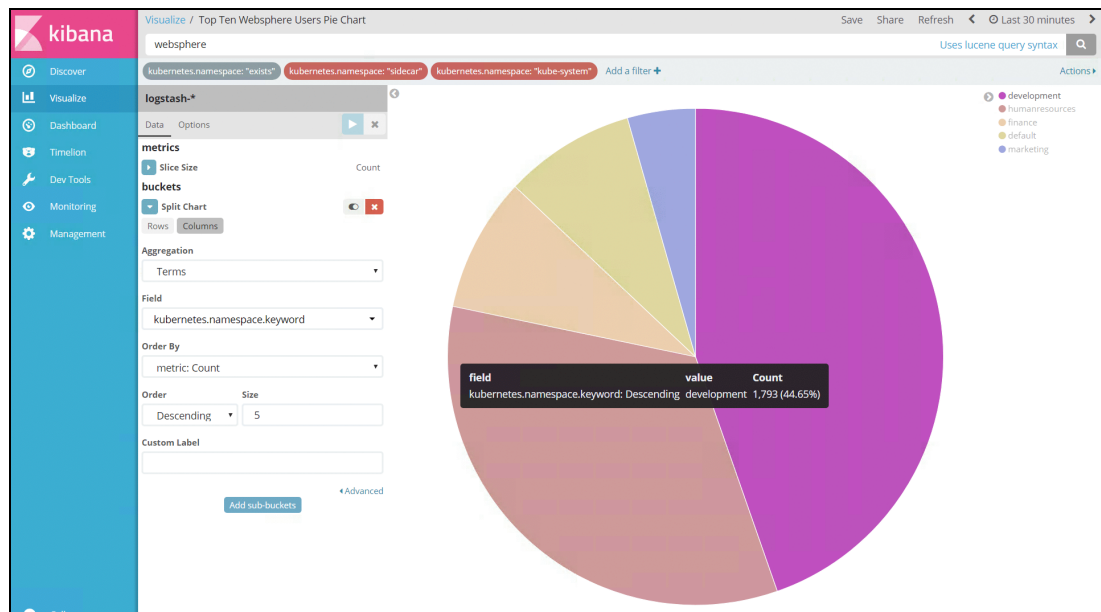


Figure 5-23 Kibana visualization pie chart

Explore the other visualizations available, ideally creating a few more as these can be used to create a more meaningful Kibana Dashboard.

Dashboards

The Kibana Dashboard page is where you can create, modify and view your own custom dashboards where multiple visualizations can be combined on to a single page and filter them by providing a search query or by selecting filters. Dashboards are useful for when you want to get an overview of your logs and make correlations among various visualizations and logs.

To create a new Dashboard, select the **+** icon, or **Create Visualization**, if none exist. Select **Add Visualization**, then select the visualizations created earlier you want to display in this dashboard. From here, you further filter the data shown in the individual visualizations by entering a search query, changing the time filter, or clicking on the elements within the visualization. The search and time filters work just like they do in the Discover page, except

they are only applied to the data subsets that are presented in the dashboard. You can save this dashboard by selecting **Save** at the top of the page. Figure 5-24 shows how this dashboard looks with two visualizations added.

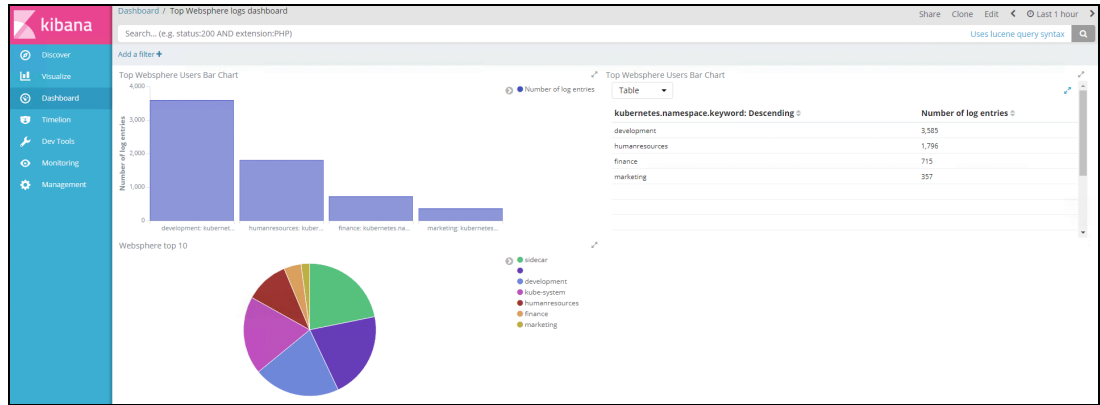


Figure 5-24 Kibana dashboard

Timelion

Timelion is a time series data visualizer that enables you to combine totally independent data sources within a single visualization. It's driven by a simple expression language you use to retrieve time series data, perform calculations to tease out the answers to complex questions and visualize the results.

The following example uses Timelion to compare time-series data about the number of logs generated in the past hour, compared to the number of logs generated this hour. From this, you can compare trends and patterns in data at different periods of time. For IBM Cloud Private, this can be useful to see trends in the logs generated on a per-pod basis. For example, the chart in Figure 5-25 compares the total number of logs in the current hour compared with the previous hour and the chart in Figure 5-26 on page 187 compares the number of logs generated by the image-manager pods in the last hour.

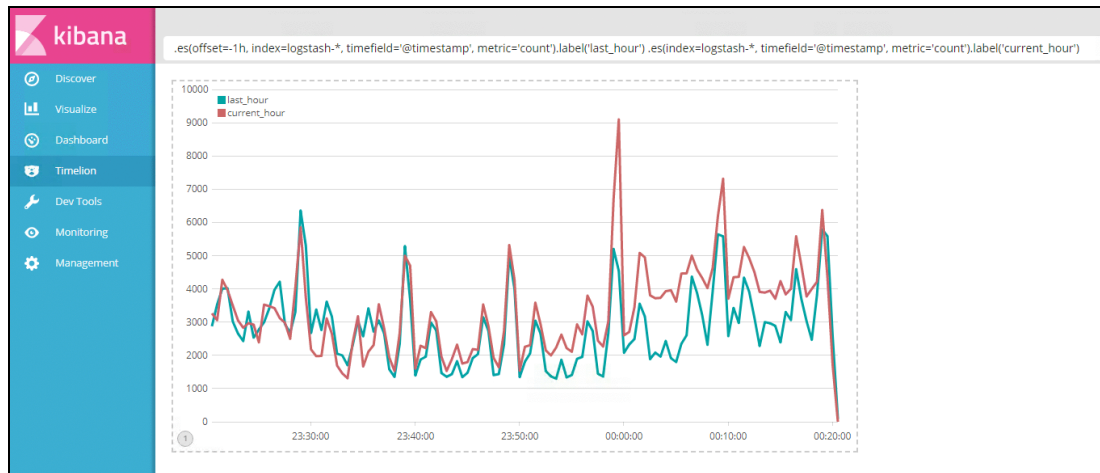


Figure 5-25 Timelion chart comparing total logs in the current hour compared to the previous hour

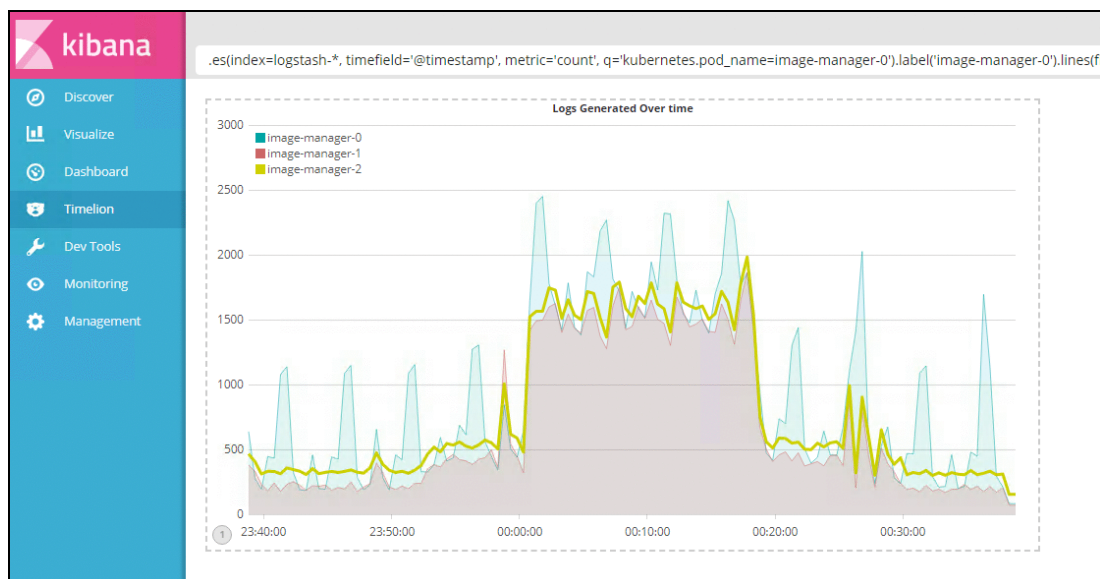


Figure 5-26 Timelion chart comparing image-manager log count in the last hour

These charts can also be saved and added to a Dashboard to provide even more analysis on the log data stored in Elasticsearch.

More information about Timelion can be found in the Elasticsearch documentation at <https://www.elastic.co/guide/en/kibana/5.5/timelion.html>.

Dev Tools

The Dev Tools page is primarily used to query the Elasticsearch API directly and can be used in a multitude of ways to retrieve information about the Elasticsearch cluster. This provides an easy way to interact with the Elasticsearch API without having to execute the commands from within the Elasticsearch client container. For example, Figure 5-27 shows executing the `_cat/indices` API command to retrieve a list of indices

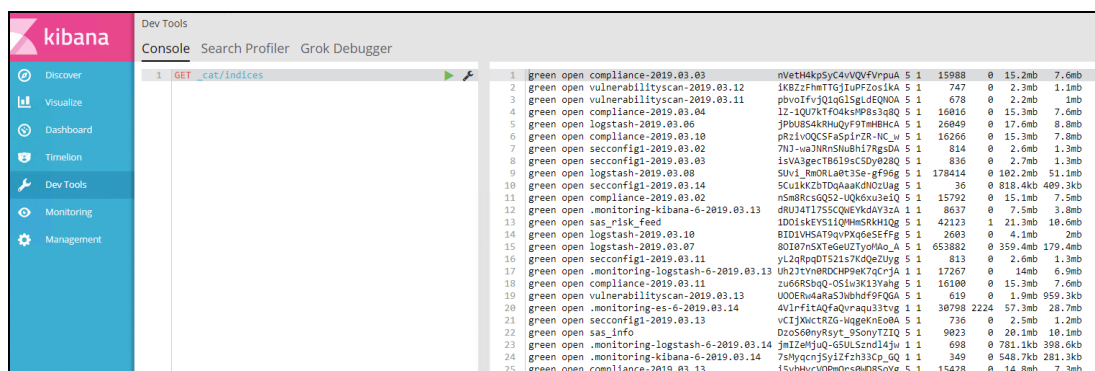


Figure 5-27 API command to retrieve the list of indices

Important: One thing to note about this tool is that all users with access to Kibana currently have access to the Elasticsearch API, which at the time of writing is not RBAC filtered, so all users can run API commands against Elasticsearch. It is possible to disable the Dev Tools page, by adding `console.enabled: false` to the `kibana.yml` content in the `logging-elk-kibana-config` ConfigMap in the `kube-system` namespace and restarting the Kibana pod.

Elasticsearch extensively covers the use of the API in its documentation at <https://www.elastic.co/guide/en/elasticsearch/reference/5.5/cat.html>, so it is not covered in this chapter.

5.2.8 Management

The Management page allows for modifying the configuration of several aspects of Kibana. Most of the settings are not configurable, as they are controlled by the use of a configuration file within the Kibana container itself, but the Management page lets you modify settings related to the stored user data, such as searches, visualizations, dashboards and indices.

Deploying ELK in IBM Cloud Private

Deploying ELK stacks to an IBM Cloud Private cluster is a common practice when there is a need to segregate the platform logging from the application logging. Elasticsearch is excellent at handling many user deployments at scale, but it makes sense to use separate logging systems, especially when a specific application has a high demand for logging and could potentially disrupt the platform logging services by overloading it in the event of a disaster.

Adding additional ELK stacks does not come without its price. As described throughout this chapter, logging takes a toll on the resources available within a cluster, favoring a lot of memory to function correctly. When designing a cluster, it's important to take in to consideration whether multiple ELK stacks are required, and if so, the resources required using the capacity planning methods discussed in "Capacity planning" on page 164. The same care should be taken when designing clusters for production that include multiple ELK stacks for applications and the configuration should be thoroughly tested in the same way the platform ELK is tested for resiliency. Failure to do so will result in the loss of logging services and potentially loss of data if Kubernetes has to restart some components due bad design by not having enough memory to fulfill the logging requirements for an application.

Planning for multiple ELK stacks should, in theory, be a little easier than figuring out how much the platform ELK should be scaled to meet the needs of both the platform and the applications it is hosting. This is because developers typically know how much log data their application (or group of applications) produces based on experience or native monitoring tools. In this situation, you can solely focus on what the application needs, as opposed to catering for the unknowns that the platform brings.

Architectural considerations

Before deploying additional ELK stacks to the cluster, think about how it will affect any resources available for the namespace. Resource requirements for ELK can be quite high, so if you are deploying ELK to an individual namespace where users operate, take this into consideration when designing Resource Quotas for that namespace. Best practice is to deploy ELK to its own namespace to isolate it from user workloads, and users themselves if necessary.

The ELK stack requires elevated privileges in order to function correctly, in particular, it requires the `IPC_LOCK` privilege which is not included in the default Pod Security Policy. If ELK is not being deployed to the `kube-system` namespace, the Service Account that ELK will use (typically the default Service Account for the hosting namespace) should be configured to use a Pod Security Policy that permits the use of `IPC_LOCK`. This can be done by creating a new Pod Security Policy for ELK and by creating a new ClusterRole and RoleBinding. Here there are two options to consider:

1. Deploying the ELK stack to the `kube-system` namespace
2. Deploying the ELK stack to the user namespace

3. Deploying the ELK stack to another dedicated namespace

Deploying the ELK stack to the users namespace means that the users that have access to the namespace also have access to view the resources within it. This means that users will be able to perform operations on the ELK pods (depending on the user roles) including viewing the CA certificates used to deploy the ELK stack security configuration (if ELK is deployed with security). By giving a service account within the user namespace elevated privileges, you're also allowing users to acquire those privileges, so ensure that the `IPC_LOCK` capability does not conflict with any security policies. `IPC_LOCK` enables `mlock` to protect the heap memory from being swapped. You can read more about `mlock` at https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_MRG/1.3/html/Real_time_Reference_Guide/sect-Realtime_Reference_Guide-Memory_allocation-Using_mlock_to_avoid_memory_faults.html.

If this is an issue, consider deploying the additional ELK stacks to a separate namespace. Additional namespaces can be used to host the additional ELK stacks and utilise the standard RBAC mechanisms within IBM Cloud Private to prevent unauthorized users from accessing the ELK pods. However in this scenario (providing security is enabled), users in other namespaces would not be able to access the Kibana pods to view logs or the status for troubleshooting. If, as a Cluster Administrator, you do not require users to monitor the Kibana pods, then restricting access to it is recommended. If users should be able to view logs and the status of the Kibana pod (to troubleshoot access issues), an additional namespace can be used to host the Kibana pod, where users can be given a 'Viewer' role. This still provides access to the Kibana pods for diagnostics but prevents users from making any changes to its state or configuration, further protecting the ELK stack from malicious intent.

For development clusters, deploying the ELK stack to user namespaces may be sufficient. For production clusters where access to the core ELK components should be restricted, deploying the ELK stack to a dedicated management namespace is recommended.

Standard versus managed mode

The ELK Helm chart provides a `mode` parameter that defines whether or not ELK is deployed as 'managed' or 'standard'. Managed mode is generally reserved to replace the platform ELK and contains several core functions that are only enabled when the ELK stack is deployed to the `kube-system` namespace, which of course is where the platform ELK is hosted. Deploying several managed ELK stacks will not work without additional modification and is not a recommended configuration. It is possible to deploy a managed ELK stack to another namespace, but additional configuration is still needed and is not covered in this book. The main functional differences between managed and managed mode are summarized in Table 5-4.

Table 5-4 Comparison of managed and standard mode

Attribute	Standard	Managed
Kibana Access method	NodePort	Ingress
Access Security	None	ICP Management Ingress
RBAC	No	Yes
Host Affinity	Worker nodes	Management nodes

Each mode has pros and cons and the mode to choose entirely depends on what the ELK requirements are. For example, if the requirement is to deploy only one additional ELK stack dedicated to application logs, but it should be secure, implement RBAC mechanisms and not consume worker node resources, then the managed mode is suitable. The drawback is that it

requires modifications to the deployed resources to work alongside the platform ELK. If multiple ELK stacks are needed for team or application dedicated ELK stacks where the use of NodePort (no authentication required) is acceptable, then standard mode is suitable.

The recommended way to deploy additional ELK stacks is by using the standard mode. Managed mode is possible to achieve, but introduces a lot of additional configuration to enable all the beneficial features of managed mode and it is not covered in this book.

Storage

Data node replicas in each deployment of ELK require a dedicated PersistentVolume (PV) and PersistentVolumeClaim (PVC). If a dynamic storage provider is available, ELK can be configured to use this during deployment. If dynamic provisioning is not available, then suitable PVs must be created first.

Deploying ELK stacks

This section will deploy a new standard mode ELK stack to the `elk` namespace and this namespace will be used in all subsequent example commands.

To create the namespace, use `kubectl create namespace elk` and then label is using `kubectl label namespace elk -l name=elk`. Alternatively, use the YAML definition in Example 5-16.

Example 5-16 elk namespace YAML definition

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    name: elk
  name: elk
```

RBAC

The ELK stack requires privileged containers, `IPC_LOCK` and `SYS_RESOURCE` capabilities, which means giving the default Service Account (SA) in the `elk` namespace elevated privileges, as the default restricted policy is too restrictive for ELK to function. To allow this, a new Pod Security Policy (PSP) is required, as well as a Cluster Role and Role Binding to the new PSP. To create the required RBAC resources perform the following steps

1. Copy the `elk-psp.yaml` in Example 5-17 to a local file called `elk-psp.yaml` and create it in Kubernetes using `kubectl create -f elk-psp.yaml`.

Example 5-17 elk-psp.yaml

```
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: ibm-elk-psp
spec:
  allowPrivilegeEscalation: true
  privileged: true
  allowedCapabilities:
    - CHMOD
    - CHOWN
    - IPC_LOCK
    - SYS_RESOURCE
  forbiddenSysctls:
```

```

- '*'
fsGroup:
  rule: RunAsAny
runAsUser:
  rule: RunAsAny
seLinux:
  rule: RunAsAny
supplementalGroups:
  rule: RunAsAny
volumes:
- configMap
- emptyDir
- projected
- secret
- downwardAPI
- persistentVolumeClaim
- hostPath

```

Alternatively, download it from

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-S-Guide/tree/master/Ch7-Logging-and-monitoring/Deploying-ELK/elk-psp.yaml>

2. Create a Cluster Role to enable the use of the new PSP

```

kubectl -n elk create clusterrole elk-clusterrole --verb=use
--resource=podsecuritypolicy --resource-name=ibm-elk-psp

```

3. In the elk namespace, create a Role Binding to the ibm-elk-psp Cluster Role

```

kubectl -n elk create rolebinding elk-rolebinding --clusterrole=elk-clusterrole
--serviceaccount=elk:default

```

4. Verify the default Service Account in the elk namespace can use the ibm-elk-psp PSP

```

kubectl auth can-i --as=system:serviceaccount:elk:default -n elk use
podsecuritypolicy/ibm-elk-psp

```

This should output a simple yes or no. If the output is no, check the correct names have been used when creating the Cluster Role or Role Binding.

After the above steps, RBAC is configured with the correct privileges for ELK.

Pulling images

As ELK is deployed to a dedicated namespace, it's necessary to create an image pull secret so that it can pull the ELK images from the ibmcom namespace that contains the platform images.

Create an image pull secret in the elk namespace using the platform admin credentials.

```

kubectl -n elk create secret docker-registry ibmcomregkey
--docker-server="mycluster.icp:8500/ibmcom" --docker-username="admin"
--docker-password="admin" --docker-email="admin@docker.io"

```

Replace mycluster.icp and the username/password credentials with the values for the environment. This Secret will be used later to allow the ELK pods to pull images from the ibmcom namespace.

Security

It's recommended to enable security on all deployed ELK stacks. You have the option of using the platform generated Certificate Authority (CA), supplying your own or letting Elasticsearch

generate certificates internally. The recommended approach is to supply your own CA, or create an entirely new CA specifically for each ELK stack to provide as much isolation as possible in clusters using multiple ELK stacks. If a malicious user has access to the CA key-pair used for each ELK deployment, it's possible for that user to gain access to the other ELK stacks. If the cluster is a 'trusted environment' model, then this may not be a problem, but for other clusters where security and isolation of log data is a key requirement, it is recommended to use a new CA key-pair for each ELK deployment.

To create a dedicated CA for each ELK stack use the **openssl** command in Example 5-18 replacing the subject details if necessary.

Example 5-18 Creating a new CA using openssl

```
openssl req -newkey rsa:4096 -sha256 -nodes -keyout ca.key -x509 -days 36500 -out ca.crt -subj "/C=US/ST=NewYork/L=Armonk/O=IBM Cloud Private/CN=www.ibm.com"
```

This will output a ca.crt and ca.key file to your local machine. Run the command in Example 5-20 to create a new key-pair Kubernetes Secret from these files.

Example 5-19 Create secret from new CA

```
kubectl -n elk create secret generic elk-ca-secret --from-file=ca.crt --from-file=ca.key
```

This secret can be used later when deploying ELK.

To specify your own CA to use when deploying additional ELK stacks, three requirements must be met:

- ▶ The CA must be stored in a Kubernetes secret.
- ▶ The secret must exist in the namespace to which the ELK stack is deployed.
- ▶ The contents of the certificate and its secret key must be stored in separately named fields (or keys) within the Kubernetes secret.

If the keys are stored locally, run the command in Example 5-20 to create a new secret, replacing <path-to-file> with the file path of the files.

Example 5-20 Create secret from custom CA

```
kubectl -n elk create secret generic elk-ca-secret --from-file=<path-to-file>/my_ca.crt --from-file=<path-to-file>/my_ca.key
```

Alternatively, Example 5-21 shows the YAML for a sample secret using defined CA certificates. You'll need to paste the contents of the my_ca.crt and my_ca.key in the YAML definition in your preferred editor.

Example 5-21 YAML definition for my-ca-secret

```
apiVersion: v1
kind: Secret
metadata:
  name: my-ca-secret
type: Opaque
data:
  my_ca.crt: ...
  my_ca.key: ...
```

This secret can be used later when deploying ELK.

If your own CA is not supplied, and the default IBM Cloud Private cluster certificates are suitable, they are located in the `cluster-ca-cert` Secret in the `kube-system` namespace. As ELK is deployed to another namespace, it's important to copy this Secret to that namespace, otherwise the deployment will fail trying to locate it. This is only relevant if you are not using your own (or generated) CA for ELK. Use the command in Example 5-22 to copy the `cluster-ca-cert` to the `elk` namespace, using `sed` to replace the namespace name without manually editing it.

Example 5-22 Copying cluster-ca-cert to elk namespace

```
kubectl -n kube-system get secret cluster-ca-cert -o yaml | sed 's/namespace: kube-system/namespace: elk/g' | kubectl -n elk create -f -
```

Deploying ELK

The `ibm-icplogging` Helm chart contains all the Elasticsearch, Logstash, Kibana and Filebeat components required to deploy an ELK stack to an IBM Cloud Private cluster. The chart version used in this deployment is 2.2.0, which is the same as the platform ELK stack deployed during cluster installation in IBM Cloud Private Version 3.1.2 to ensure the images used throughout the platform are consistent between ELK deployments. This Helm chart can be retrieved from the `mgmt-charts` repository, as it is not (at the time of writing) published in the IBM public Helm chart repository.

The chart will install the following ELK components as pods in the cluster:

- ▶ client
- ▶ data
- ▶ filebeat (one per node)
- ▶ logstash
- ▶ master
- ▶ kibana (optional)

The goal of this example ELK deployment is to provide logging capabilities for applications running in the production namespace. ELK will be configured to monitor the dedicated production worker nodes, retrieving log data from applications in that namespace only.

To deploy the ELK Helm chart, perform the following steps:

1. Log in to IBM Cloud Private using `cloudctl` to ensure `kubectl` and `helm` command line utilities are configured.
2. If dynamic storage provisioning is enabled in the cluster, this can be used. If dynamic storage is not available, PersistentVolumes should be created prior to deploying ELK. In this deployment, the namespace isolation features in IBM Cloud Private 3.1.2 have been used to create a dedicated worker node for ELK. This means that a LocalVolume PersistentVolume can be used, as ELK will be running on only one node. Example 5-23 is a YAML definition for a PersistentVolume that uses LocalVolume, so the data node uses the `/var/lib/icp/applogging/elk-data` file system on the hosting dedicated worker node.

Example 5-23 YAML definition for a Persistent Volume using LocalVolume on a management node

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: applogging-datanode-172.24.19.212
```

```
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 20Gi
  local:
    path: /var/lib/icp/applogging/elk-data
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - 172.24.19.212
  persistentVolumeReclaimPolicy: Retain
  storageClassName: logging-storage-datanode
```

In this deployment, two of these volumes are created; one for each management node in the cluster.

3. Retrieve the current platform ELK chart values and save to a local file. These values can be used to replace the defaults set in the Helm chart itself so that the correct images are used.

```
helm get values logging --tls > default-values.yaml
```

4. Create a file called `override-values.yaml`. This will be the customized configuration for ELK and is required to override some of the default values to tailor the resource names, curator duration, security values or number of replicas of each component in this deployment. Use the values in Example 5-24 as a template.

Example 5-24 override-values.yaml

```
cluster_domain: cluster.local
mode: standard
nameOverride: elk
general:
  mode: standard
image:
  pullPolicy: IfNotPresent
  pullSecret:
    enabled: true
    name: ibmcomregkey
curator:
  name: log-curator
  app:
    count: 28
elasticsearch:
  client:
    replicas: "1"
    name: client
  data:
    name: data
    replicas: "1"
    storage:
      persistent: true
      size: 20Gi
```



```

        storageClass: logging-storage-datanode
        useDynamicProvisioning: false
    master:
        name: master
        replicas: "1"
    name: elasticsearch
    filebeat:
        name: filebeat-ds
        scope:
            namespaces:
            - production
        nodes:
            production: "true"
    kibana:
        name: kibana
        install: true
        external:
    logstash:
        name: logstash
        replicas: "1"
    security:
        ca:
            external:
                certFieldName: ca.crt
                keyFieldName: ca.key
                secretName: elk-ca-secret
            origin: external
        enabled: true
    xpack:
        monitoring: true

```

These values should be tailored to meet the requirements. If dynamic provisioning is enabled in the cluster, set `elasticsearch.data.storageClass` to the appropriate storage class name and `elasticsearch.data.useDynamicProvisioning` value to `true`.

In this values file, the `kibana.external` is intentionally left empty, so that Kubernetes will automatically assign a `NodePort` value from the default `NodePort` range. At the time of writing, the Helm chart does not support automatic `NodePort` assignment when deploying through the IBM Cloud Private catalog user interface, due to validation on empty fields. Therefore auto-assignment is only possible by deploying the Helm chart through the Helm CLI.

Additional values not in the `override-values.yaml` can also be assigned using the `--set` parameter in the `helm install` command.

5. Deploy the `ibm-icplogging-2.2.0.tgz` Helm chart using `helm install`, passing the values files created earlier. The values files order is important as Helm will override the values from each file in sequence. For example, the values set in `override-values.yaml` will replace any values in the `default-values.yaml` file as priority is given to the right-most file specified.

```
helm install ibm-icplogging-2.2.0.tgz --name app-logging --namespace elk -f
default-values.yaml -f override-values.yaml --tls
```

After some time, all resources should have been deployed to the `elk` namespace. To view all pods in the release, use `kubectl -n elk get pods -l release=app-logging`

```
[root@icp-ha-boot cluster]# kubectl -n elk get pods -l release=app-logging
```

NAME	READY	STATUS	RESTARTS	AGE
------	-------	--------	----------	-----

app-logging-elk-client-868db5cbd9-6mhjl	1/1	Running	0	1h
app-logging-elk-data-0	1/1	Running	0	1h
app-logging-elk-elasticsearch-pki-init-78bzt	0/1	Completed	0	1h
app-logging-elk-kibana-86df58d79d-wfgwx	1/1	Running	0	1h
app-logging-elk-kibana-init-m9r92	0/1	CrashLoopBackOff	22	1h
app-logging-elk-logstash-68f996bc5-92gpd	1/1	Running	0	1h
app-logging-elk-master-6c64857b5b-x4j9b	1/1	Running	0	1h

Tip: If the kibana-init pod fails, it's because it could not initialize the default index in Kibana. This is not a problem, as the default index can be set through the Kibana UI.

- Retrieve the NodePort for the Kibana service:

```
kubectl -n elk get service kibana
-o=jsonpath='{.spec.ports[?(@.port==5601)].nodePort}'
```

- Use the returned port to access Kibana via an IBM Cloud Private node. For example using the proxy

```
http://<proxy-ip>:<nodeport>
```

This will display the Kibana dashboard. See Figure 5-28.

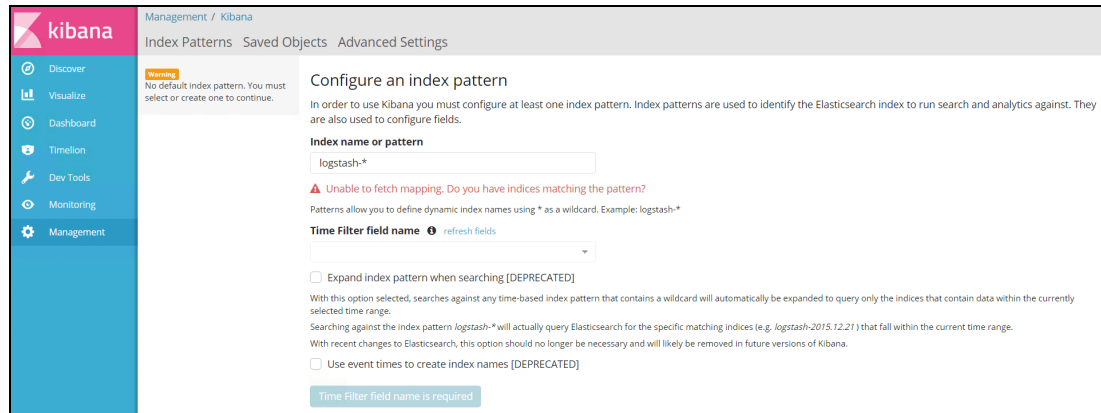


Figure 5-28 New Kibana user interface

- Set the default index to whichever value you choose. The default is `logstash-` but this may change depending on how you modify Logstash in this instance. Note that it is not possible to set the default index until data with that index actually exists in Elasticsearch, so before this can be set, ensure log data is sent to Elasticsearch first.

Configuring namespace based indices

The default Logstash configuration forwards all log data to an index in the format `logstash-<year-month-day>`. Whilst this is suitable for the platform logs, it makes sense for additional ELK stacks, designed to collect logs for specific namespaces, to create indices based on the namespace the logs originate from. This can be achieved by editing the Logstash ConfigMap and modifying the output to use the `kubernetes.namespace` field as the index name instead of the default `logstash`. To do this for the ELK stack deployed in previous sections, edit the `app-logging-elk-logstash-config` ConfigMap in the `elk` namespace and change the `output.elasticsearch.index` section from

```
output {
  elasticsearch {
    hosts => "elasticsearch:9200"
    index => "logstash-%{+YYYY.MM.dd}"
    document_type => "%{[@metadata][type]}"
  }
}
```

```

        ssl => true
        ssl_certificate_verification => true
        keystore =>
"/usr/share/elasticsearch/config/tls/logstash-elasticsearch-keystore.jks"
        keystore_password => "${APP_KEYSTORE_PASSWORD}"
        truststore => "/usr/share/elasticsearch/config/tls/truststore.jks"
        truststore_password => "${CA_TRUSTSTORE_PASSWORD}"
    }
}

to
output {
  elasticsearch {
    hosts => "elasticsearch:9200"
    index => "%{kubernetes.namespace}-%{+YYYY.MM.dd}"
    document_type => "%{[@metadata][type]}"
    ssl => true
    ssl_certificate_verification => true
    keystore =>
"/usr/share/elasticsearch/config/tls/logstash-elasticsearch-keystore.jks"
    keystore_password => "${APP_KEYSTORE_PASSWORD}"
    truststore => "/usr/share/elasticsearch/config/tls/truststore.jks"
    truststore_password => "${CA_TRUSTSTORE_PASSWORD}"
  }
}

```

Save and close to update the ConfigMap. Logstash will automatically reload the new configuration. If it does not reload after 3 minutes, delete the Logstash pod(s) to restart them.

As Logstash does the buffering and transformation of log data, it contains a variety of useful functions to translate, mask or remove potentially sensitive fields and data from each log message, such as passwords or host data. More information about mutating the log data can be found at

<https://www.elastic.co/guide/en/logstash/5.5/plugins-filters-mutate.html>.

Configuring the curator

If the default logstash index has been removed in favor of namespace based indices, The curator with this ELK stack also needs to be modified to cleanup the log data older than the number of days specified in the ELK deployment. As this Elasticsearch may contain a single or multiple namespace logs that should be deleted after a certain time period, the default 'logstash-' prefix filter can be removed to catch all time-based indices. Use `kubectl -n elk edit configmap app-logging-elk-elasticsearch-curator-config` to edit the curator configuration and remove the first filtertype in action 1

```

- filtertype: pattern
  kind: prefix
  value: logstash-

```

The resulting filters should look similar to the following, which applies to all indices in this ELK deployment

```

filters:
- filtertype: age
  source: name
  direction: older
  timestring: '%Y.%m.%d'

```

```
unit: days
unit_count: 28
```

If only namespace1, namespace2 and namespace3 indices should be deleted, you can use a regex pattern, similar to the following

```
filters:
- filtertype: pattern
  kind: regex
  value: '^(namespace1-|namespace2-|namespace3-).*$'
- filtertype: age
  source: name
  direction: older
  timestring: '%Y.%m.%d'
  unit: days
  unit_count: 28
```

Securing access to Kibana

As Kibana was installed with NodePort as the access method, it leaves Kibana exposed to users outside of the cluster, which in most environment is not suitable. To secure Kibana, the NodePort should be removed and an Ingress added with the appropriate annotations to restrict access to Kibana only to users of IBM Cloud Private. At the time of writing, there is no segregation of access to Ingress resources via RBAC. To do this, perform the following steps

1. Edit the kibana service to remove the spec.ports.nodePort field and value and change spec.type to ClusterIP, using `kubectl -n elk edit service kibana`. The Service should look similar to Example 5-25.

Example 5-25 Kibana service definition

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: app-logging-elk-elasticsearch
    chart: ibm-icplogging-2.2.0
    component: kibana
    heritage: Tiller
    release: app-logging
  name: kibana
  namespace: elk
spec:
  clusterIP: 10.0.144.80
  ports:
  - port: 5601
    protocol: TCP
    targetPort: ui
  selector:
    app: app-logging-elk-elasticsearch
    component: kibana
    role: kibana
  type: ClusterIP
```

2. Create an ingress using Example 5-26 as a template, replacing the ingress name and path if necessary.

Example 5-26 Kibana secure ingress

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: app-kibana-ingress
  namespace: elk
  annotations:
    icp.management.ibm.com/auth-type: "access-token"
    icp.management.ibm.com/rewrite-target: "/"
    icp.management.ibm.com/secure-backends: "true"
    kubernetes.io/ingress.class: "ibm-icp-management"
spec:
  rules:
  - http:
      paths:
      - path: "/app-kibana"
        backend:
          serviceName: "kibana"
          servicePort: 5601
```

3. Modify the Kibana ConfigMap to add the `server.basePath` value. This is required to prevent targeting the default Kibana instance when the `/app-kibana` ingress path is used in the browser. Edit the Kibana ConfigMap using `kubectl -n elk edit configmap app-logging-elk-kibana-config` and add `'server.basePath: "/app-kibana"'` anywhere in the data section.
4. Test the new URL by logging out of IBM Cloud Private and trying to access `https://<master-ip>:8443/app-kibana`. This should redirect you to the log in page for IBM Cloud Private.

Scaling Filebeat namespaces

Filebeat is in control of deciding from which namespaces log data is sent to Elasticsearch. After deploying ELK, you can modify the Filebeat ConfigMap to add or remove namespaces.

To do this with Helm, create a file called `fb-ns.yaml` with the content in Example 5-27

Example 5-27 fb-ns.yaml

```
filebeat:
  scope:
    namespaces:
    - production
    - pre-production
```

Use Helm to push the new configuration, passing the default parameters used during initial chart installation. These values are required as the default chart values are different, and Helm will use the default chart values as a base for changes.

Run the helm upgrade by using `helm upgrade`:

```
helm upgrade app-logging ibm-icplogging-2.2.0.tgz --reuse-values -f fb-ns.yaml
--namespace elk --recreate-pods --tls
```

You can also do this without Helm by modifying Kubernetes resources directly. Edit the following input paths in the `app-logging-elk-filebeat-ds-config` ConfigMap in the `elk` namespace. For example, to add the `pre-production` namespace, add the line

- `"/var/log/containers/*_pre-production_*.log"` to the input paths

For example:

```
filebeat.prospectors:
- input_type: log
  paths:
  - "/var/log/containers/*_production_*.log"
  - "/var/log/containers/*_pre-production_*.log"
```

This method may be preferred if additional modifications have been made to the ELK stack.

5.2.9 Forwarding logs to external logging systems

A common requirement in IBM Cloud Private installations is to integrate with an existing logging solution, where the platform (and all workloads) logs should be sent to a another external logging system, either instead of, or as well as, the platform deployed Elasticsearch.

To achieve this, there are two options to consider:

- Logs are sent *only* to the external logging system, Elasticsearch is redundant.
- Logs are sent to *both* Elasticsearch and the external logging system.

In the case of option 1, the current Filebeat and Logstash components already deployed can be repurposed to ship all log data to the external logging system. Depending on the external system, only Filebeat may need to be retained. The output configuration for Filebeat and Logstash can be redirected to the external system instead of the platform Elasticsearch, but additional security certificates may need to be added as volumes and volume mounts to both Filebeat and/or Logstash, if the external system uses security.

For option 2, depending on the external system, it's recommended to deploy an additional Filebeat and Logstash to IBM Cloud Private. It's possible to add an additional 'pipeline' to Logstash, so that it can stream logs to the platform Elasticsearch and the external system simultaneously, but this also introduces additional effort to debug in the event one pipeline fails. With separate deployments, it's easy to determine which Logstash is failing and why.

Both Filebeat and Logstash have a number of plugins that enable output to a variety of endpoints. More information about the available output plugins for Filebeat can be found at <https://www.elastic.co/guide/en/beats/filebeat/current/configuring-output.html> and information about the available output plugins for Logstash can be found at <https://www.elastic.co/guide/en/logstash/5.5/output-plugins.html>.

If Vulnerability Advisor is deployed, it's recommended to retain the current Elasticsearch, as Vulnerability Advisor stores data within Elasticsearch and may not function properly if it cannot reach it.

Deploying Filebeat and Logstash

At the time of writing, there is no independent Helm chart available to deploy Filebeat or Logstash separately. The deployment configuration will entirely depend on the level of security the external system has for example basic authentication versus TLS authentication.

In this example, an external Elasticsearch has been set up using HTTP demonstrate how Filebeat and Logstash can be used in IBM Cloud Private to send logs to an external ELK. Filebeat and Logstash will use the default certificates generated by the platform to secure the communication between these components. Filebeat and Logstash are deployed to a dedicated namespace. Perform the following steps

1. Create a new namespace called `external-logging` that will host the Filebeat Daemonset and Logstash Deployment containers:

```
kubectl create namespace external-logging
```

2. Create a new namespace called `external` that will host the containers to generate logs for this example:

```
kubectl create namespace external
```

3. Copy the required Secret and ConfigMaps to the `external-logging` namespace. Note that doing this provides any users with access to this namespace the ability to view the authentication certificates for the platform ELK. In this example, only the required certificates are extracted from the `logging-elk-certs` in `kube-system`.

- a. Copy the required files from the `logging-elk-certs` Secret

- i. Copy the `logging-elk-certs` Secret:

```
kubectl -n kube-system get secret logging-elk-certs -o yaml | sed  
"s/namespace: kube-system/namespace: external-logging/g" | kubectl -n  
external-logging create -f -
```

- ii. Remove all entries in data *except* for the following, using `kubectl -n external-logging edit secret logging-elk-certs`:

```
- ca.crt  
- logstash.crt  
- logstash.key  
- filebeat.crt  
- filebeat.key
```

- b. Copy the `logging-elk-elasticsearch-pki-secret` Secret to the `external-logging` namespace:

```
kubectl -n kube-system get secret logging-elk-elasticsearch-pki-secret -o  
yaml | sed "s/namespace: kube-system/namespace: external-logging/g" |  
kubectl -n external-logging create -f -
```

- c. Copy the `logging-elk-elasticsearch-entriypoint` ConfigMap:

```
kubectl -n kube-system get configmap logging-elk-elasticsearch-entriypoint -o  
yaml | sed "s/namespace: kube-system/namespace: external-logging/g" |  
kubectl -n external-logging create -f -
```

4. Create a RoleBinding to the `ibm-anyuid-hostaccess-clusterrole` so that the Logstash pod can use the host network to reach the external Elasticsearch:

```
kubectl -n external-logging create rolebinding  
ibm-anyuid-hostaccess-rolebinding --clusterrole  
ibm-anyuid-hostaccess-clusterrole --serviceaccount=external-logging:default
```

Tip: This step is not needed if Logstash is being deployed to `kube-system` namespace, as it automatically inherits this privilege from the `ibm-privileged-psp` Pod Security Policy.

5. Create the `filebeat-config.yaml` in Example 5-28.

Example 5-28 filebeat-config.yaml

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  labels:  
    app: filebeat-ds
```

```

name: filebeat-config
namespace: external-logging
data:
  filebeat.yml: |-
    filebeat.prospectors:
    - input_type: log
      paths:
        - /var/log/containers/*_external_*.log
      scan_frequency: 10s
      symlinks: true
      json.message_key: log
      json.keys_under_root: true
      json.add_error_key: true
      multiline.pattern: '^\s'
      multiline.match: after
      fields_under_root: true
      fields:
        type: kube-logs
        node.hostname: ${NODE_HOSTNAME}
        pod.ip: ${POD_IP}
      tags:
        - k8s-app
    filebeat.config.modules:
      # Set to true to enable config reloading
      reload.enabled: true
    output.logstash:
      hosts: logstash:5044
      timeout: 15
      ssl.certificate_authorities: ["/usr/share/elasticsearch/config/tls/ca.crt"]
      ssl.certificate: "/usr/share/elasticsearch/config/tls/filebeat.crt"
      ssl.key: "/usr/share/elasticsearch/config/tls/filebeat.key"
      ssl.key_passphrase: "${APP_KEYSTORE_PASSWORD}"
    logging.level: info

```

Alternatively, download the file from

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch7-Logging-and-monitoring/Deploying-Filebeat-and-Logstash/filebeat-config.yaml>.

6. Create the logstash-config.yaml in Example 5-29, replacing output.elasticsearch.hosts with your own Elasticsearch host.

Example 5-29 logstash-config.yaml

```

apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    app: logstash
  name: logstash-config
  namespace: external-logging
data:
  k8s.conf: |-
    input {
      beats {
        port => 5044

```



```

        ssl => true
        ssl_certificate_authorities =>
["/usr/share/elasticsearch/config/tls/ca.crt"]
        ssl_certificate => "/usr/share/elasticsearch/config/tls/logstash.crt"
        ssl_key => "/usr/share/elasticsearch/config/tls/logstash.key"
        ssl_key_passphrase => "${APP_KEYSTORE_PASSWORD}"
        ssl_verify_mode => "force_peer"
    }
}

filter {
    if [type] == "kube-logs" {
        mutate {
            rename => { "message" => "log" }
            remove_field => ["host"]
        }

        date {
            match => ["time", "ISO8601"]
        }

        dissect {
            mapping => {
                "source" =>
"/var/log/containers/%{kubernetes.pod}_%{kubernetes.namespace}_%{container_file_ext}"
            }
        }

        dissect {
            mapping => {
                "container_file_ext" => "%{container}_%{?file_ext}"
            }
            remove_field => ["host", "container_file_ext"]
        }

        grok {
            "match" => {
                "container" =>
"^%{DATA:kubernetes.container_name}-(?<kubernetes.container_id>[0-9A-Za-z]{64,64})"
            }
            remove_field => ["container"]
        }
    }
}

filter {
    # Drop empty lines
    if [log] =~ /\s*$/ {
        drop { }
    }
    # Attempt to parse JSON, but ignore failures and pass entries on as-is
    json {
        source => "log"
    }
}

```

```

        skip_on_invalid_json => true
      }
    }

    output {
      elasticsearch {
        hosts => ["9.30.123.123:9200"]
        index => "%{kubernetes.namespace}-%{+YYYY.MM.dd}"
        document_type => "%{[@metadata][type]}"
      }
    }
  }
}

logstash.yml: |-
  config.reload.automatic: true
  http.host: "0.0.0.0"
  path.config: /usr/share/logstash/pipeline
  xpack.monitoring.enabled: false
  xpack.monitoring.elasticsearch.url: "http://9.30.123.123:9200"

```

Important: This configuration does not use security. To enable security using keystores, add the following section to `output.elasticsearch`

```

ssl => true
ssl_certificate_verification => true
keystore => "/usr/share/elasticsearch/config/tls/keystore.jks"
keystore_password => "${APP_KEYSTORE_PASSWORD}"
truststore => "/usr/share/elasticsearch/config/tls/truststore.jks"
truststore_password => "${CA_TRUSTSTORE_PASSWORD}"

```

Alternatively, download the file from

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch7-Logging-and-monitoring/Deploying-Filebeat-and-Logstash/logstash-config.yaml>.

7. Create the `logstash-service.yaml` in Example 5-30.

Example 5-30 logstash-service.yaml

```

apiVersion: v1
kind: Service
metadata:
  labels:
    app: logstash
  name: logstash
  namespace: external-logging
spec:
  ports:
    - name: beats
      port: 5044
      protocol: TCP
      targetPort: 5044
  selector:
    app: logstash
  type: ClusterIP

```

Alternatively, download the file from

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch7-Logging-and-monitoring/Deploying-Filebeat-and-Logstash/logstash-service.yaml>.

8. Create the `logstash-deployment.yaml` in Example 5-31. In this deployment, the Logstash container will run on the management node, so the management node in your environment should have network connectivity to the external ELK. It is scheduled to the management node to prevent unauthorized access to the Logstash pod and protect the target Elasticsearch from unauthorized commands if a worker node is compromised.

Example 5-31 logstash-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: logstash
  name: logstash
  namespace: external-logging
spec:
  replicas: 1
  selector:
    matchLabels:
      app: logstash
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      annotations:
        productID: none
        productName: Logstash
        productVersion: 5.5.1
        scheduler.alpha.kubernetes.io/critical-pod: ""
      labels:
        app: logstash
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: beta.kubernetes.io/arch
                    operator: In
                    values:
                      - amd64
                      - ppc64le
                      - s390x
              - key: management
                operator: In
                values:
                  - "true"
      containers:
        - command:
```

```

- /bin/bash
- /scripts/entrypoint.sh
env:
- name: LS_JAVA_OPTS
  value: -Xmx512m -Xms512m
- name: CFG_BASEDIR
  value: /usr/share/logstash
- name: CA_TRUSTSTORE_PASSWORD
  valueFrom:
    secretKeyRef:
      key: caTruststorePassword
      name: logging-elk-elasticsearch-pki-secret
- name: APP_KEYSTORE_PASSWORD
  valueFrom:
    secretKeyRef:
      key: appKeystorePassword
      name: logging-elk-elasticsearch-pki-secret
image: ibmcom/icp-logstash:5.5.1-f2
imagePullPolicy: IfNotPresent
name: logstash
ports:
- containerPort: 5044
  protocol: TCP
resources:
  limits:
    memory: 1Gi
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
volumeMounts:
- mountPath: /usr/share/logstash/pipeline
  name: pipeline-config
- mountPath: /usr/share/logstash/config/logstash.yml
  name: logstash-config
  subPath: logstash.yml
- mountPath: /usr/share/logstash/data
  name: data
- mountPath: /scripts
  name: entrypoint
- mountPath: /usr/share/elasticsearch/config/tls
  name: certs
  readOnly: true
restartPolicy: Always
securityContext: {}
terminationGracePeriodSeconds: 30
tolerations:
- effect: NoSchedule
  key: dedicated
  operator: Exists
volumes:
- configMap:
    defaultMode: 420
    items:
    - key: k8s.conf
      path: k8s.conf
    name: logging-elk-logstash-config

```

```

    name: pipeline-config
  - configMap:
    defaultMode: 420
    items:
    - key: logstash.yml
      path: logstash.yml
      name: logging-elk-logstash-config
    name: logstash-config
  - configMap:
    defaultMode: 365
    items:
    - key: logstash-entrypoint.sh
      path: entrypoint.sh
    - key: map-config.sh
      path: map-config.sh
      name: logging-elk-elasticsearch-entrypoint
    name: entrypoint
  - name: certs
    secret:
    defaultMode: 420
    secretName: logging-elk-certs
  - emptyDir: {}
    name: data

```

Alternatively, download the file from

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch7-Logging-and-monitoring/Deploying-Filebeat-and-Logstash/logstash-deployment.yaml>

9. Create the `filebeat-ds.yaml` in Example 5-32.

Example 5-32 filebeat-ds.yaml

```

apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  labels:
    app: filebeat
  name: filebeat-ds
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: filebeat
  template:
    metadata:
      annotations:
        productID: none
        productName: filebeat
        productVersion: 5.5.1
        scheduler.alpha.kubernetes.io/critical-pod: ""
      labels:
        app: filebeat
    spec:
      containers:
      - env:

```

```

- name: NODE_HOSTNAME
  valueFrom:
    fieldRef:
      apiVersion: v1
      fieldPath: spec.nodeName
- name: POD_IP
  valueFrom:
    fieldRef:
      apiVersion: v1
      fieldPath: status.podIP
- name: CA_TRUSTSTORE_PASSWORD
  valueFrom:
    secretKeyRef:
      key: caTruststorePassword
      name: logging-elk-elasticsearch-pki-secret
- name: APP_KEYSTORE_PASSWORD
  valueFrom:
    secretKeyRef:
      key: appKeystorePassword
      name: logging-elk-elasticsearch-pki-secret
image: ibmcom/icp-filebeat:5.5.1-f2
imagePullPolicy: IfNotPresent
livenessProbe:
  exec:
    command:
      - sh
      - -c
      - ps aux | grep '[f]ilebeat' || exit 1
  failureThreshold: 3
  periodSeconds: 30
  successThreshold: 1
  timeoutSeconds: 1
name: filebeat
readinessProbe:
  exec:
    command:
      - sh
      - -c
      - ps aux | grep '[f]ilebeat' || exit 1
  failureThreshold: 3
  initialDelaySeconds: 10
  periodSeconds: 10
  successThreshold: 1
  timeoutSeconds: 1
resources: {}
terminationMessagePath: /dev/termination-log
terminationMessagePolicy: File
volumeMounts:
- mountPath: /usr/share/filebeat/filebeat.yml
  name: config
  subPath: filebeat.yml
- mountPath: /usr/share/filebeat/data
  name: data
- mountPath: /usr/share/elasticsearch/config/tls
  name: certs

```

```

      readOnly: true
    - mountPath: /var/log/containers
      name: container-log
      readOnly: true
    - mountPath: /var/log/pods
      name: pod-log
      readOnly: true
    - mountPath: /var/lib/docker/containers/
      name: docker-log
      readOnly: true
  restartPolicy: Always
  securityContext:
    runAsUser: 0
  terminationGracePeriodSeconds: 30
  tolerations:
    - effect: NoSchedule
      key: dedicated
      operator: Exists
  volumes:
    - configMap:
        defaultMode: 420
        items:
          - key: filebeat.yml
            path: filebeat.yml
        name: logging-elk-filebeat-ds-config
      name: config
    - name: certs
      secret:
        defaultMode: 420
        secretName: logging-elk-certs
    - emptyDir: {}
      name: data
    - hostPath:
        path: /var/log/containers
        type: ""
        name: container-log
    - hostPath:
        path: /var/log/pods
        type: ""
        name: pod-log
    - hostPath:
        path: /var/lib/docker/containers
        type: ""
        name: docker-log
  updateStrategy:
    rollingUpdate:
      maxUnavailable: 1
    type: RollingUpdate

```

Alternatively, download the file from

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch7-Logging-and-monitoring/Deploying-Filebeat-and-Logstash/filebeat-ds.yaml>.

10. After all resources are deployed, check the pods are running:

```
[root@icp-ha-boot ~]# kubectl -n external-logging get pods
NAME                                READY   STATUS    RESTARTS   AGE
filebeat-ds-4pf52                   1/1     Running   0           81s
filebeat-ds-7hshw                   1/1     Running   0           87s
filebeat-ds-bm2dd                   1/1     Running   0           89s
filebeat-ds-ddk55                   1/1     Running   0           87s
filebeat-ds-l5d2v                   1/1     Running   0           84s
filebeat-ds-t26gt                   1/1     Running   0           90s
filebeat-ds-x5kx                    1/1     Running   0           84s
logstash-6d7f976b97-f85ft          1/1     Running   0           11m
```

11. Create some workload in the external namespace, so that some log data is generated. In this example, two WebSphere Liberty Helm charts were deployed from the IBM public Helm chart repository. During start up the containers create log data which is sent to the external Elasticsearch, as shown in Figure 5-29.

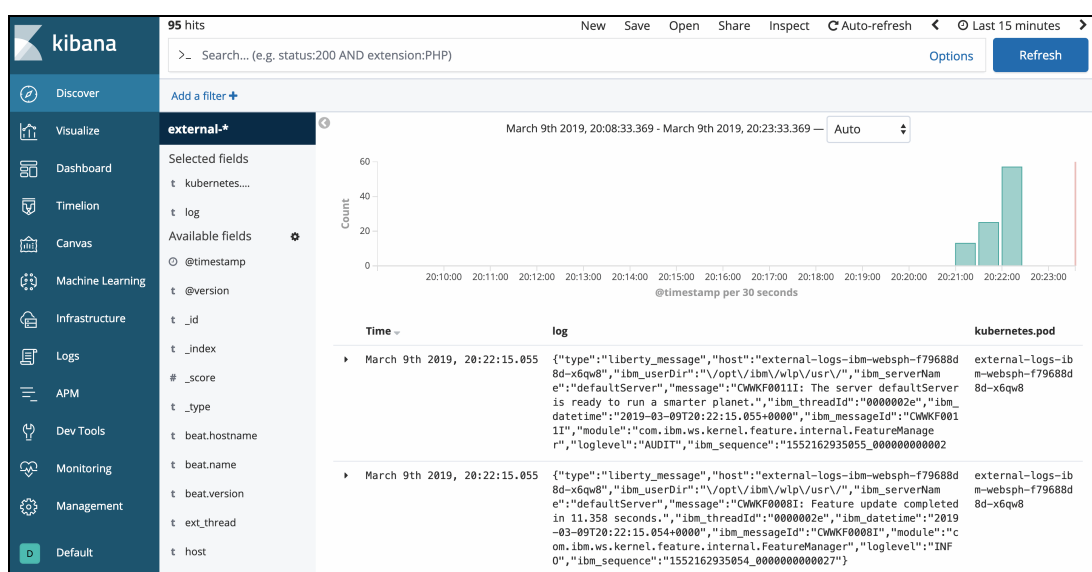


Figure 5-29 Logs sent to external ELK

Reconfiguring the platform Logstash

If all platform and application logs are sent to an external system, requiring no use of the platform ELK, you can modify the Logstash outputs to send logs to the external system instead. This can be done by modifying the output section in the logging-elk-logstash-config ConfigMap in the kube-system namespace. The default configuration looks like the following:

```
output {
  elasticsearch {
    hosts => ["9.30.231.235:9200"]
    index => "%{kubernetes.namespace}-%{+YYYY.MM.dd}"
    document_type => "%{[metadata][type]}"
  }
}
```

This output uses the Elasticsearch plug-in. A list of all available plug-ins can be found at <https://www.elastic.co/guide/en/logstash/5.5/output-plugins.html>. For example, to use a generic HTTP endpoint, use the following example:

```
output {
```



```

http {
  http_method => "post"
  url => "http://<external_url>"
  <other_options>
  ...
  ...
}

```

5.2.10 Forwarding logs from application log files

In many containerised applications today, not all the log data generated by the application is sent to stdout and stderr and instead writes to a log file. Unless this particular log file is on a filesystem mounted from a PersistentVolume, every time Kubernetes restarts the container, the log data will be lost. There are several ways in which log data from a log file in a container can make it's way to Elasticsearch, but this section will focus on two commons methods:

1. Using a Filebeat side car to read data from specific directories or files and stream the content to stdout or stderr.
2. Using a Filebeat side car to read data from specific directories or files and stream the output directly to Logstash.

Option 1 is the recommended option as all the log data is sent to stdout and stderr, which also gives the advantage of allowing tools such as `kubectl logs` to read the log data, as Docker handles storing stdout and stderr data on the host, which is read by kubelet. This data is also automatically sent to Elasticsearch using the default logging mechanisms. See Figure 5-30.

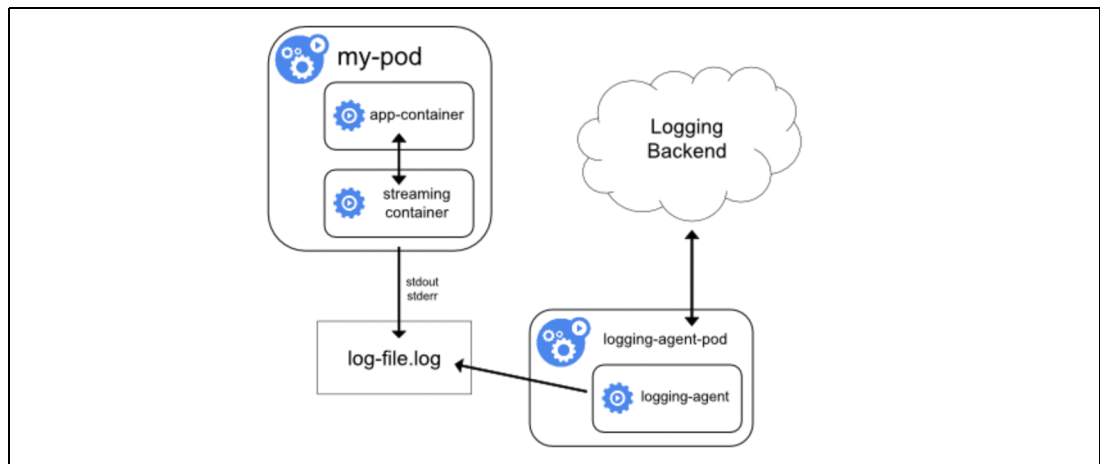


Figure 5-30 Side car logging to stdout and stderr

Option 2 is also a useful solution, as the data from specific log files can be parsed or transformed in the side car and pushed directly to a logging solution, whether it's the platform ELK, an application dedicated logging system or an external logging system entirely. See Figure 5-31.

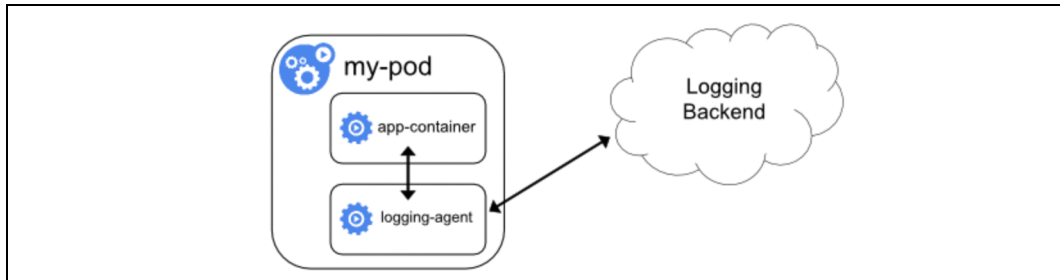


Figure 5-31 Side car logging directly to logging system

Important: At the time of writing, the current Filebeat image runs as the root user within the container, so ensure this complies with your security policies before giving the namespace access to a PodSecurityPolicy that allows containers to be run with the root user.

Using a Filebeat side car to forward log file messages to stdout and stderr

This example will use a simple WebSphere Liberty deployment and Filebeat side car to send log data from a file populated by WebSphere within the WebSphere container to stdout, to simulate typical application logging. This functionality can be achieved in a similar way, by mounting another image, such as busybox, to tail the file and redirect to stdout in the busybox container, but this is not scalable and requires multiple busybox side car containers for multiple log files. Filebeat has a scalability advantage, as well as advanced data processing to output the data to stdout and stderr flexibly.

To create a WebSphere and Filebeat side car Deployment, perform the following steps:

1. Create a new namespace called sidecar for this example
`kubectl create namespace sidecar`
2. Creating a ConfigMap for the Filebeat configuration allows you to reuse the same settings for multiple deployments without redefining an instance of Filebeat every time. Alternatively, you can create one ConfigMap per deployment if your deployment requires very specific variable settings in Filebeat. This ConfigMap will be consumed by the Filebeat container as its core configuration data.

Create the ConfigMap in Example 5-33 to store the Filebeat configuration using `kubectl create -f filebeat-sidecar-config.yaml`.

Example 5-33 *filebeat-sidecar-config.yaml*

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: filebeat-sidecar-config
  namespace: sidecar
data:
  filebeat.yml: |-
    filebeat.prospectors:
    - input_type: log
      paths: '${LOG_DIRS}'
      exclude_lines: '${EXCLUDE_LINES:[]}'
      include_lines: '${INCLUDE_LINES:[]}'
  
```

```

ignore_older: '${IGNORE_OLDER:0}'
scan_frequency: '${SCAN_FREQUENCY:10s}'
symlinks: '${SYMLINKS:true}'
max_bytes: '${MAX_BYTES:10485760}'
harvester_buffer_size: '${HARVESTER_BUFFER_SIZE:16384}'

multiline.pattern: '${MULTILINE_PATTERN:^\s}'
multiline.match: '${MULTILINE_MATCH:after}'
multiline.negate: '${MULTILINE_NEGATE:false}'

filebeat.config.modules:
  # Set to true to enable config reloading
  reload.enabled: true

output.console:
  codec.format:
    string: '%{[message]}'

logging.level: '${LOG_LEVEL:info}'

```

Alternatively, download from

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch7-Logging-and-monitoring/Forwarding-logs-from-application-log-files/stdout/filebeat-sidecar-config.yaml>.

This configuration will capture all log messages from the specified files and relay them to stdout. It's worth noting that this will simply relay all message types to stdout, which is later captured by the Filebeat monitoring the docker logs. If you require additional formatting of log messages, then consider using the approach to send formatted log data directly to something such as Logstash.

3. Create a RoleBinding to the `ibm-anyuid-clusterrole` to enable the Filebeat container to run as the root user:

```

kubectl -n sidecar create rolebinding sidecar-anyuid-rolebinding
--clusterrole=ibm-anyuid-clusterrole --serviceaccount=sidecar:default

```

4. Create some workloads that writes data to a file, with a Filebeat side car. Example 5-34 uses a simple WebSphere Liberty deployment and a Filebeat side car to output the contents of `/var/log/applogs/app.log` to stdout. In this example, the string `Logging data to app.log - <number>: <current-date-time>` is output to the `app.log` file every second. Create the Deployment using `kubectl create -f websphere-liberty-fb-sidecar.yaml`.

Example 5-34 websphere-liberty-fb-sidecar-deployment.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: websphere-sidecar
  name: websphere-sidecar
  namespace: sidecar
spec:
  replicas: 1
  selector:
    matchLabels:
      app: websphere-sidecar
  strategy:

```

```

rollingUpdate:
  maxSurge: 1
  maxUnavailable: 1
  type: RollingUpdate
template:
  metadata:
    labels:
      app: websphere-sidecar
  spec:
    securityContext:
      runAsUser: 0
    containers:
      - name: ibm-websphere-liberty
        args: [/bin/sh, -c,'i=0; while true; do echo "Logging data to app.log -
$i: $(date)" >> /var/log/app.log; i=$((i+1)); sleep 1; done']
        image: websphere-liberty:latest
        imagePullPolicy: IfNotPresent
        env:
          - name: JVM_ARGS
          - name: WLP_LOGGING_CONSOLE_FORMAT
            value: json
          - name: WLP_LOGGING_CONSOLE_LOGLEVEL
            value: info
          - name: WLP_LOGGING_CONSOLE_SOURCE
            value: message,trace,accessLog,ffdc
          - name: POD_IP
            valueFrom:
              fieldRef:
                apiVersion: v1
                fieldPath: status.podIP
          - name: HTTPENDPOINT_HTTPSPORT
            value: "9443"
          - name: KEYSTORE_REQUIRED
            value: "false"
        resources: {}
        volumeMounts:
          - name: was-logging
            mountPath: /var/log
      - name: filebeat-sidecar
        image: ibmcom/icp-filebeat:5.5.1-f2
        env:
          - name: LOG_DIRS
            value: /var/log/applogs/app.log
          - name: NODE_HOSTNAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
        volumeMounts:
          - name: was-logging
            mountPath: /var/log/applogs
          - name: filebeat-config

```

```

        mountPath: /usr/share/filebeat/filebeat.yml
        subPath: filebeat.yml
volumes:
- name: was-logging
  emptyDir: {}
- name: filebeat-config
  configMap:
    name: filebeat-sidecar-config
  items:
    - key: filebeat.yml
      path: filebeat.yml

```

Alternatively, download from

<https://github.com/IBMRRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch7-Logging-and-monitoring/Forwarding-logs-from-application-log-files/stdout/websphere-liberty-fb-sidecar-deployment.yaml>.

In this deployment, there are two volumes specified. The `filebeat-config` volume mounts the ConfigMap data and the `was-logging` volume stores the application logs in the container. For each folder containing logs, you should create a new volume in a similar way for fine grained control over each file. For deployments that require persistent storage, replace `emptyDir: {}` with a `PersistentVolumeClaim` definition, as this deployment will lose its log data if it is restarted. The `was-logging` volume is mounted to the container running WebSphere and both the `was-logging` and `filebeat-config` volumes are mounted to the Filebeat container.

When adding this to an existing deployment, there are several new sections that should be added. This section defines the Filebeat container with the environment variables that set the directories for Filebeat to use. For the variables `LOG_DIRS` you can provide a single directory path or a comma-separated list of directories. The `was-logging` volume is mounted on the `/var/log/applogs` directory, which will be equal to the `/var/log` directory after mounting the same volume on the `ibm-websphere-liberty` container. In a real world example, this would be the filepath to the log file(s) you want Filebeat to scan and should match up with the mount path on the volume mount for the WebSphere container.

```

- name: filebeat-sidecar
  image: ibmcom/icp-filebeat:5.5.1-f2
  env:
    - name: LOG_DIRS
      value: /var/log/applogs/app.log
    - name: NODE_HOSTNAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
  volumeMounts:
    - name: was-logging
      mountPath: /var/log/applogs
    - name: filebeat-config
      mountPath: /usr/share/filebeat/filebeat.yml
      subPath: filebeat.yml

```

The key integration here is the volume mounts between the `ibm-websphere-liberty` container and the `filebeat-sidecar` container. This is what provides the Filebeat side car container access to the log files from the `ibm-websphere-liberty` container. If multiple log

files need to be monitored from different locations within the file system, consider using multiple volumeMounts and update the filebeat-sidecar-config paths accordingly.

After a few minutes (to cater for the containers starting up) the log data should now be visible in Kibana after the platform Filebeat instance has successfully collected logs from Docker and pushed them through the default logging mechanism. See Figure 5-32 on page 216.

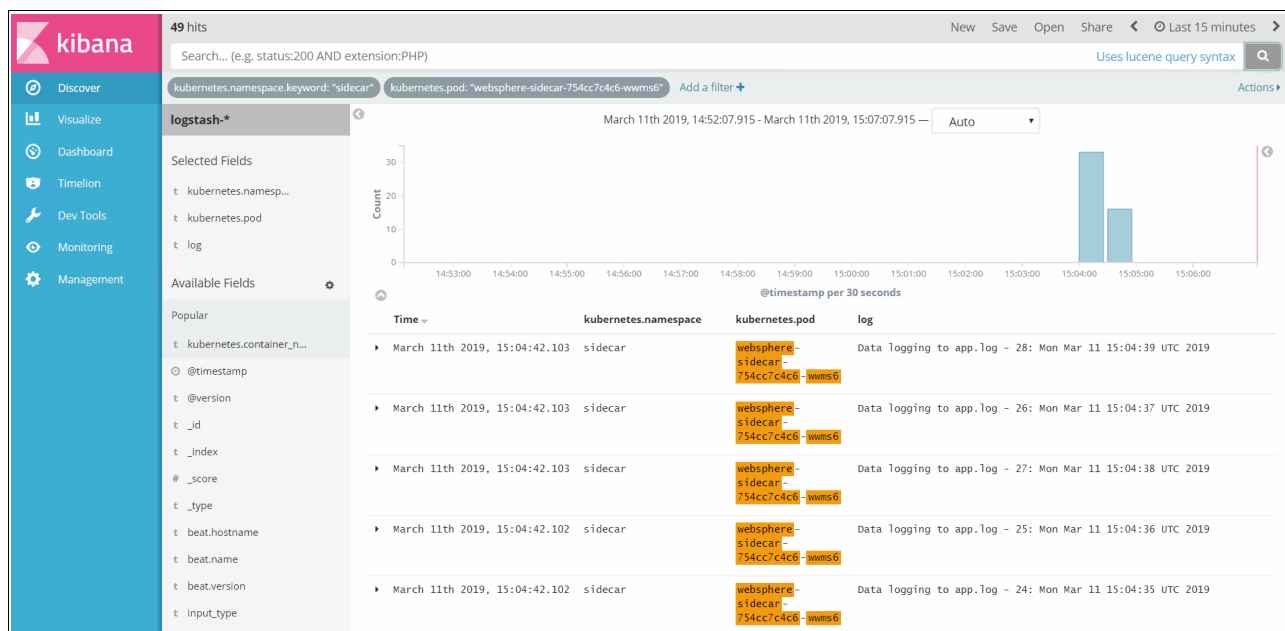


Figure 5-32 Data collected by the Filebeat

Using a Filebeat side car to forward log file messages to Logstash

This example will use a simple WebSphere Liberty deployment and Filebeat side car to send log data from a file populated by WebSphere within the WebSphere container to the external Elasticsearch used earlier in this chapter, to simulate typical application logs that need to be sent off-site. As the instance of Logstash uses TLS encryption to secure communications within the cluster, it's important to understand that some certificates will need to be imported to allow this solution to work. This example will use only the minimum certificates required to allow communication between the Filebeat sidecar and Logstash. To use a Filebeat side car to stream logs directly to Logstash, perform the following steps:

1. Copy the required Secret and ConfigMaps to the sidecar namespace. Note that doing this provides any users with access to this namespace the ability to view the authentication certificates for the platform ELK. In this example, only the required certificates are extracted from the logging-elk-certs in kube-system.
 - a. Copy the required files from the logging-elk-certs Secret:
 - i. Copy the logging-elk-certs Secret:


```
kubectl -n kube-system get secret logging-elk-certs -o yaml | sed "s/namespace: kube-system/namespace: sidecar/g" | kubectl -n sidecar create -f -
```
 - ii. Remove all entries in data *except* for the following, using `kubectl -n sidecar edit secret logging-elk-certs`:
 - ca.crt
 - filebeat.crt

- filebeat.key
- b. Copy the logging-elk-elasticsearch-pki-secret Secret to the external-logging namespace:


```
kubectl -n kube-system get secret logging-elk-elasticsearch-pki-secret -o
yaml | sed "s/namespace: kube-system/namespace: sidecar/g" | kubectl -n
sidecar create -f -
```

Important: These commands will copy the certificates used within the target ELK stack, which could be used to access Elasticsearch itself. If this does not conform to security standards, consider deploying a dedicated ELK stack for a specific application, user or namespace and provide these certificates for that stack.

2. Create the filebeat-sidecar-logstash-config.yaml ConfigMap in Example 5-35 using `kubectl create -f filebeat-sidecar-logstash-config.yaml`.

Example 5-35 filebeat-sidecar-logstash-config.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: filebeat-sidecar-logstash-config
  namespace: sidecar
data:
  filebeat.yml: |-
    filebeat.prospectors:
    - input_type: log
      paths: '${LOG_DIRS}'
      exclude_lines: '${EXCLUDE_LINES:[]}'
      include_lines: '${INCLUDE_LINES:[]}'

      ignore_older: '${IGNORE_OLDER:0}'
      scan_frequency: '${SCAN_FREQUENCY:10s}'
      symlinks: '${SYMLINKS:true}'
      max_bytes: '${MAX_BYTES:10485760}'
      harvester_buffer_size: '${HARVESTER_BUFFER_SIZE:16384}'

      multiline.pattern: '${MULTILINE_PATTERN:^\s}'
      multiline.match: '${MULTILINE_MATCH:after}'
      multiline.negate: '${MULTILINE_NEGATE:false}'

      fields_under_root: '${FIELDS_UNDER_ROOT:true}'
      fields:
        type: '${FIELDS_TYPE:kube-logs}'
        node.hostname: '${NODE_HOSTNAME}'
        pod.ip: '${POD_IP}'
        kubernetes.namespace: '${NAMESPACE}'
        kubernetes.pod: '${POD_NAME}'
        tags: '${TAGS:sidecar-ls}'

    filebeat.config.modules:
      # Set to true to enable config reloading
      reload.enabled: true

    output.logstash:
```

```

hosts: '${LOGSTASH:logstash.kube-system:5044}'
timeout: 15
ssl.certificate_authorities: ["/usr/share/elasticsearch/config/tls/ca.crt"]
ssl.certificate: "/usr/share/elasticsearch/config/tls/filebeat.crt"
ssl.key: "/usr/share/elasticsearch/config/tls/filebeat.key"
ssl.key_passphrase: ${APP_KEYSTORE_PASSWORD}

```

```
logging.level: '${LOG_LEVEL:info}'
```

Alternatively, download from

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch7-Logging-and-monitoring/Forwarding-logs-from-application-log-files/logstash/filebeat-sidecar-logstash-config.yaml>

This ConfigMap applies JSON formatted output to the platform Logstash, with added field identifiers that Logstash will use to filter data.

3. Create the websphere-sidecar-logstash-deployment.yaml in Example 5-36 using `kubectl create -f websphere-sidecar-logstash-deployment.yaml`.

Example 5-36 websphere-liberty-fb-sidecar-logstash-deployment.yaml

```

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: websphere-sidecar-logstash
  name: websphere-sidecar-logstash
  namespace: sidecar
spec:
  replicas: 1
  selector:
    matchLabels:
      app: websphere-sidecar-logstash
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: websphere-sidecar-logstash
    spec:
      securityContext:
        runAsUser: 0
      containers:
        - name: ibm-websphere-liberty
          args: [/bin/sh, -c,'i=0; while true; do echo "Using Logstash - Logging
data to app.log - $i: $(date)" >> /var/log/app.log; i=$((i+1)); sleep 1; done']
          image: websphere-liberty:latest
          imagePullPolicy: IfNotPresent
          env:
            - name: JVM_ARGS
            - name: WLP_LOGGING_CONSOLE_FORMAT
              value: json
            - name: WLP_LOGGING_CONSOLE_LOGLEVEL

```



```

    value: info
  - name: WLP_LOGGING_CONSOLE_SOURCE
    value: message,trace,accessLog,ffdc
  - name: POD_IP
    valueFrom:
      fieldRef:
        apiVersion: v1
        fieldPath: status.podIP
  - name: HTTPENDPOINT_HTTPSPORT
    value: "9443"
  - name: KEYSTORE_REQUIRED
    value: "false"
resources: {}
volumeMounts:
  - name: was-logging
    mountPath: /var/log
- name: filebeat-sidecar
  image: ibmcom/icp-filebeat:5.5.1-f2
  env:
    - name: LOG_DIRS
      value: /var/log/applogs/app.log
    - name: NODE_HOSTNAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    - name: POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    - name: CA_TRUSTSTORE_PASSWORD
      valueFrom:
        secretKeyRef:
          key: caTruststorePassword
          name: logging-elk-elasticsearch-pki-secret
    - name: APP_KEYSTORE_PASSWORD
      valueFrom:
        secretKeyRef:
          key: appKeystorePassword
          name: logging-elk-elasticsearch-pki-secret
    - name: NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    - name: POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
  volumeMounts:
    - name: was-logging
      mountPath: /var/log/applogs
    - name: filebeat-config
      mountPath: /usr/share/filebeat/filebeat.yml
      subPath: filebeat.yml
    - mountPath: /usr/share/elasticsearch/config/tls
      name: certs

```

```

      readOnly: true
volumes:
- name: was-logging
  emptyDir: {}
- name: filebeat-config
  configMap:
    name: filebeat-sidecar-logstash-config
    items:
      - key: filebeat.yml
        path: filebeat.yml
- name: certs
  secret:
    defaultMode: 420
    secretName: logging-elk-certs

```

Alternatively, download at

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/Ch7-Logging-and-monitoring/Forwarding-logs-from-application-log-files/logstash/websphere-liberty-fb-sidecar-logstash-deployment.yaml>.

This is similar to Example 5-34, but with a few key differences. First, the certificates in the logging-elk-certs Secret mounted as volumes in the filebeat-sidecar container definition, allowing it to communicate securely using TLS with the Logstash instance. Second, the introduction of additional environment variables provides the Filebeat side car with additional information that is forwarded to Logstash.

```

- name: NAMESPACE
  valueFrom:
    fieldRef:
      fieldPath: metadata.namespace
- name: POD_NAME
  valueFrom:
    fieldRef:
      fieldPath: metadata.name

```

Without this, the log entries in Elasticsearch would not contain the namespace and pod name fields.

After a few minutes (to cater for the containers starting up) the log data should now be visible in Kibana after the platform Filebeat instance has successfully collected logs from Docker and pushed them through the default logging mechanism. See Figure 5-33.

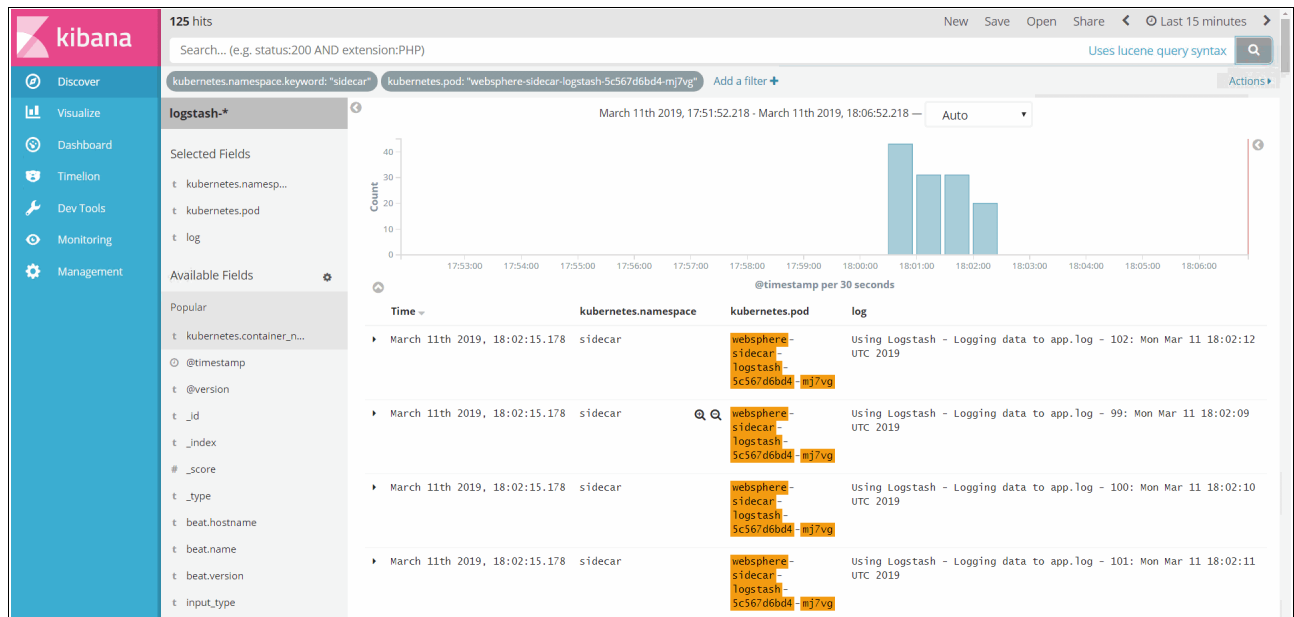


Figure 5-33 Logs collected by the Filebeat instance

5.3 IBM Cloud Private Monitoring

IBM Cloud Private Version 3.1.2 uses AlertManager, Prometheus, and Grafana stack for system monitoring. It uses components listed in Table 5-5 on page 222 on the management nodes.

Table 5-5 Monitoring and alerting components

Component	Version	Role
AlertManager	- AlertManager (0.5.0)	Handles alerts sent by the Prometheus server. It takes care of de-duplicating, grouping, and routing them to the correct receiver integration such as slack, Email, or PagerDuty.
Grafana	- Grafana (5.2.0)	Data visualization & Monitoring with support for Prometheus as datasource.
Prometheus	<ul style="list-style-type: none">- Prometheus (2.3.1)- collectd_exporter (0.4.0)- node_exporter (0.16.0)- configmap_reload (0.2.2)- elasticsearch-exporter(1.0.2)- kube-state-metrics-exporter(1.3.0)	Collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and can trigger alerts if some condition is observed to be true.

5.3.1 How Prometheus works

Prometheus is a monitoring platform that collects metrics from monitored targets by scraping metrics HTTP endpoints on these targets. Prometheus offers a richer data model and query language, in addition to being easier to run and integrate into the environment to be monitored. Figure 5-34 shows the Prometheus internal architecture.

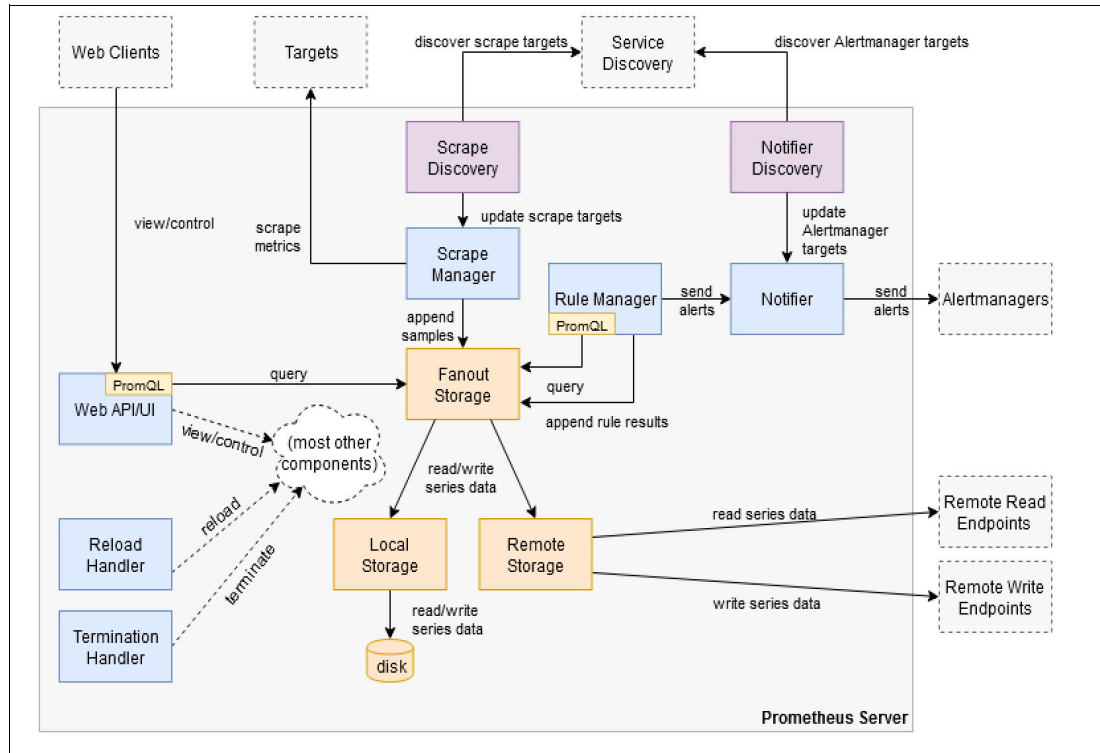


Figure 5-34 Prometheus internal architecture

Prometheus discovers targets to scrape from service discovery. The scrape discovery manager is a discovery manager that uses Prometheus's service discovery functionality to find and continuously update the list of targets from which Prometheus should scrape metrics. It runs independently of the scrape manager which performs the actual target scrape and feeds it with a stream of target group updates over a synchronization channel.

Prometheus stores time series samples in a local time series database (TSDB) and optionally also forwards a copy of all samples to a set of configurable remote endpoints. Similarly, Prometheus reads data from the local TSDB and optionally also from remote endpoints. The scrape manager is responsible for scraping metrics from discovered monitoring targets and forwarding the resulting samples to the storage subsystem.

Every time series is uniquely identified by its metric name and a set of key-value pairs, also known as labels. The metric name specifies the general feature of a system that is measured (for example `http_requests_total` - the total number of HTTP requests received). It may contain ASCII letters and digits, as well as underscores and colons. It must match the regex `[a-zA-Z_][a-zA-Z0-9_]*`.

The Prometheus client libraries offer four core metric types. These are currently only differentiated in the client libraries (to enable APIs tailored to the usage of the specific types) and in the wire protocol: These types are:

- **Counter** - A counter is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart. For example, you can use a counter to represent the number of requests served, tasks completed, or errors.
- **Gauge** - A gauge is a metric that represents a single numerical value that can arbitrarily go up and down. Gauges are typically used for measured values like temperatures or

current memory usage, but also “counts” that can go up and down, like the number of concurrent requests.

- **Histogram** - A histogram samples observations (usually things like request durations or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values.
- **Summary** - Similar to a histogram, a summary samples observations (usually things like request durations and response sizes). While it also provides a total count of observations and a sum of all observed values, it calculates configurable quantiles over a sliding time window.

The Prompt engine is responsible for evaluating Prompt expression queries against Prometheus’s time series database. The engine does not run as its own actor goroutine, but is used as a library from the web interface and the rule manager. PromQL evaluation happens in multiple phases: when a query is created, its expression is parsed into an abstract syntax tree and results in an executable query. The subsequent execution phase first looks up and creates iterators for the necessary time series from the underlying storage. It then evaluates the PromQL expression on the iterators. Actual time series bulk data retrieval happens lazily during evaluation (at least in the case of the local TSDB (time series database)). Expression evaluation returns a PromQL expression type, which most commonly is an instant vector or range vector of time series.

Prometheus serves its web UI and API on port 9090 by default. The web UI is available at / and serves a human-usable interface for running expression queries, inspecting active alerts, or getting other insight into the status of the Prometheus server.

For more information on Prometheus see the documentation in url below:

<https://prometheus.io/docs/introduction/overview/>

IBM Cloud Private provides the following exporters to provide metrics as listed in Table 5-6 below:

Table 5-6 IBM Cloud Private exporters

Exporter Types	Exporter Details
node-exporter	Provides the node-level metrics, including metrics for CPU, memory, disk, network, and other components
kube-state-metrics	Provides the metrics for Kubernetes objects, including metrics for pod, deployment, statefulset, daemonset, replicaset, configmap, service, job, and other objects
elasticsearch-exporter	Provides metrics for the IBM Cloud Private Elasticsearch logging service, including the status for Elasticsearch cluster, shards, and other components
collectd-exporter	Provides metrics that are sent from the collectd network plug-in

Role-based access control to IBM Cloud Private monitoring

A user with role ClusterAdministrator, Administrator or Operator can access monitoring service. A user with role ClusterAdministrator or Administrator can perform write operations in monitoring service, including deleting Prometheus metrics data, and updating Grafana configurations. Starting with version 1.2.0, the ibm-icpmonitoring Helm chart introduces an important feature. It offers a new module that provides role-based access controls (RBAC) for access to the Prometheus metrics data.

The RBAC module is effectively a proxy that sits in front of the Prometheus client pod. It examines the requests for authorization headers, and at that point, enforces role-based controls. It does this by retrieving the users current IBM Cloud Private access token, retrieved when logging in to the IBM Cloud Private dashboard. From the access token, the proxy can identify the user and their roles, and use this information to filter the query results, ensuring users can only see metrics they are authorized to see.

5.3.2 How AlertManager works

Alerting with Prometheus is separated into two parts. Alerting rules in Prometheus servers send alerts to an Alertmanager. The Alertmanager then manages those alerts, including silencing, inhibition, aggregation and sending out notifications via methods such as:

- ▶ Email
- ▶ Generic Webhooks
- ▶ HipChat
- ▶ OpsGenie
- ▶ PagerDuty
- ▶ Pushover
- ▶ Slack

The rule manager in Prometheus is responsible for evaluating recording and alerting rules on a periodic basis (as configured using the `evaluation_interval` configuration file setting). It evaluates all rules on every iteration using PromQL and writes the resulting time series back into the storage. The notifier takes alerts generated by the rule manager via its `Send()` method, enqueues them, and forwards them to all configured Alertmanager instances. The notifier serves to decouple generation of alerts from dispatching them to Alertmanager (which may fail or take time).

The Alertmanager handles alerts sent by client applications such as the Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration. The following describes the core concepts the Alertmanager implements:

Grouping - Grouping categorizes alerts of similar nature into a single notification. This is especially useful during larger outages when many systems fail at once and hundreds to thousands of alerts may be firing simultaneously.

Inhibition - Inhibition is a concept of suppressing notifications for certain alerts if certain other alerts are already firing.

Silences - Silences are a straightforward way to simply mute alerts for a given time. A silence is configured based on matchers, just like the routing tree. Incoming alerts are checked whether they match all the equality or regular expression matchers of an active silence. If they do, no notifications will be sent out for that alert.

For more information on Alertmanager refer the url below:

<https://prometheus.io/docs/alerting/overview/>

5.3.3 How Grafana works

Grafana includes built-in support for Prometheus. Grafana exposes metrics for Prometheus on the `/metrics` endpoint. IBM Cloud Private comes with a Grafana dashboard to display the health status and various metrics about the cluster. Users can access the Prometheus UI directly at `https://<master-ip>:8443/prometheus`.

Grafana.com maintains a collection of shared dashboards which can be downloaded and used with standalone instances of Grafana. Prebuilt dashboards could be downloaded from url below:

<https://grafana.com/dashboards?dataSource=prometheus>

Prometheus data source could be configured and then data could be used in graphs. For more details refer the document below:

<https://prometheus.io/docs/visualization/grafana/>

IBM Cloud Private following Grafana dashboards listed in Table 5-7 below.

Table 5-7 IBM Cloud Private Grafana dashboards

Dashboard type	Dashboard details
ElasticSearch	Provides information about ElasticSearch cluster statistics, shard, and other system information
Etdcd by Prometheus	Etdcd Dashboard for Prometheus metrics scraper
Helm Release Metrics	Provides information about system metrics such as CPU and Memory for each Helm release that is filtered by pods
ICP Namespaces Performance IBM Provided 2.5	Provides information about namespace performance and status metrics
Cluster Network Health (Calico)	Calico hosts workload and system metric performance information
ICP Performance IBM Provided 2.5	Provides TCP system performance information about Nodes, Memory, and Containers
Kubernetes Cluster Monitoring	Monitors Kubernetes clusters that use Prometheus. Provides information about cluster CPU, Memory, and File system usage. The dashboard also provides statistics for individual pods, containers, and systemd services
Kubernetes POD Overview	Monitors pod metrics such as CPU, Memory, Network pod status, and restarts
NGINX Ingress controller	Provides information about NGINX Ingress controller metrics that can be sorted by namespace, controller class, controller, and ingress
Node Performance Summary	Provides information about system performance metrics such as CPU, Memory, Disk, and Network for all nodes in the cluster
Prometheus Stats	Dashboard for monitoring Prometheus v2.x.x
Storage GlusterFS Health	Provides GlusterFS Health metrics such as Status, Storage, and Node
Rook-Ceph	Dashboard that provides statistics about Ceph instances
Storage Minio Health	Provides storage and network details about Minio server instances

5.3.4 Accessing Prometheus, Alertmanager and Grafana dashboards

To access the Prometheus, Alertmanager and Grafana dashboard first log in to the IBM Cloud Private management console.

- ▶ To access the Grafana dashboard, click **Menu** → **Platform** → **Monitoring**. Alternatively, you can open `https://<IP_address>:<port>/grafana`, where `<IP_address>` is the DNS or IP address that is used to access the IBM Cloud Private console. `<port>` is the port that is used to access the IBM Cloud Private console.
- ▶ To access the Alertmanager dashboard, click **Menu** → **Platform** → **Alerting**. Alternatively, you can open `https://<IP_address>:<port>/alertmanager`.
- ▶ To access the Prometheus dashboard, open `https://<IP_address>:<port>/prometheus`.

5.3.5 Configuring Prometheus Alertmanager and Grafana in IBM Cloud Private

Users can customise the monitoring service pre-installation or post-installation. If configuring pre-installation then make changes to `config.yaml` file located in the `/<installation_directory>/cluster` folder. Users can customize the values of the parameters, as required.

The `monitoring.prometheus` section has the following parameters:

- ▶ `prometheus.scrapeInterval` - is the frequency to scrape targets in Prometheus
- ▶ `prometheus.evaluationInterval` - is the frequency to evaluate rules in Prometheus
- ▶ `prometheus.retention` - is the duration of time to retain the monitoring data
- ▶ `prometheus.persistentVolume.enabled` - is a flag that users set to use a persistent volume for Prometheus. The flag false means that prometheus do not use a persistent volume
- ▶ `prometheus.persistentVolume.storageClass` - is the storage class to be used by Prometheus
- ▶ `prometheus.resources.limits.cpu` - is the CPU limit that you set for the Prometheus container. The default value is 500 millicpu.
- ▶ `prometheus.resources.limits.memory` - is the memory limit that you set for the Prometheus container. The default value is 512 million bytes.

The `monitoring.alertmanager` section has the following parameters:

- ▶ `alertmanager.persistentVolume.enabled` - is a flag that you set to use a persistent volume for Alertmanager. The flag false means that you do not use a persistent volume
- ▶ `alertmanager.persistentVolume.storageClass` - is the storage class to be used by Alertmanager
- ▶ `alertmanager.resources.limits.cpu` - is the CPU limit that you set for the Alertmanager container. The default value is 200 millicpu.
- ▶ `alertmanager.resources.limits.memory` - is the memory limit that you set for the Alertmanager container. The default value is 256 million bytes.

The `monitoring.grafana` section has the following parameters:

- ▶ `grafana.user` - is the user name that you use to access Grafana.
- ▶ `grafana.password` - is the password of the user who is specified in the `grafana.user` parameter.
- ▶ `grafana.persistentVolume.enabled` - is a flag that you set to use a persistent volume for Grafana. The flag false means that you do not use a persistent volume.

- ▶ *grafana.persistentVolume.storageClass* - is the storage class to be used by Grafana
- ▶ *grafana.resources.limits.cpu* - is the CPU limit that you set for the Grafana container. The default value is 500 millicpu.
- ▶ *grafana.resources.limits.memory* - is the memory limit that you set for the Grafana container. The default value is 512 million bytes.

The resulting entry in `config.yaml` might resemble the YAML in Example 5-37.

Example 5-37 Monitoring configuration in config.yaml

```
monitoring:
  prometheus:
    scrapeInterval: 1m
    evaluationInterval: 1m
    retention: 24h
    persistentVolume:
      enabled: false
      storageClass: "-"
    resources:
      limits:
        cpu: 500m
        memory: 2048Mi
      requests:
        cpu: 100m
        memory: 128Mi
  alertmanager:
    persistentVolume:
      enabled: false
      storageClass: "-"
    resources:
      limits:
        cpu: 200m
        memory: 256Mi
      requests:
        cpu: 10m
        memory: 64Mi
  grafana:
    persistentVolume:
      enabled: false
      storageClass: "-"
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 100m
        memory: 128Mi
```

For more information and additional parameters to set, see https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/installing/monitoring.html

Example 5-38, Example 5-39, and Example 5-40 show the **kubectl** command to verify the configurations set for Prometheus, Alertmanager, and Grafana after they are deployed.

Example 5-38 Command to verify the configurations for Prometheus

```
kubectl -n kube-system get configmap monitoring-prometheus -o yaml
```

Example 5-39 Command to verify the configuration for Alertmanager

```
kubectl -n kube-system get configmap monitoring-prometheus-alertmanager -o yaml
```

Example 5-40 Command to verify the configuration for Grafana

```
kubectl -n kube-system get configmap monitoring-grafana -o yaml
```

5.3.6 Creating Prometheus alert rules

You can use the Kubernetes custom resource, AlertRule, to manage alert rules in IBM Cloud Private. Example 5-41 shows an example of alert rule to trigger alert of the node memory consumption is greater than 60%.

Create a rule file `sample-rule.yaml`, as shown in Example 5-41.

Example 5-41 Alert rule to monitor node memory

```
apiVersion: monitoringcontroller.cloud.ibm.com/v1
kind: AlertRule
metadata:
  name: sample-redbook-rule
spec:
  enabled: true
  data: |-
    groups:
      - name: redbook.rules
        rules:
          - alert: NodeMemoryUsage
            expr: ((node_memory_MemTotal_bytes - (node_memory_MemFree_bytes +
node_memory_Buffers_bytes + node_memory_Cached_bytes))/
node_memory_MemTotal_bytes) * 100 > 60
            annotations:
              DESCRIPTION: '{{ $labels.instance }}: Memory usage is above the 60%
threshold. The current value is: {{ $value }}.'
              SUMMARY: '{{ $labels.instance }}: High memory usage detected'
```

To create the rule, run the command, as shown in Example 5-42.

Example 5-42 Create node memory monitoring rule

```
$ kubectl apply -f sample-rule.yaml -n kube-system
alertrule "sample-redbook-rule" configured
```

When the rule is created, you will be able to see the new alert in the Alerts tab in the Prometheus dashboard.

Figure 5-35 shows the Node Memory Alert rule created and started for one of the nodes.

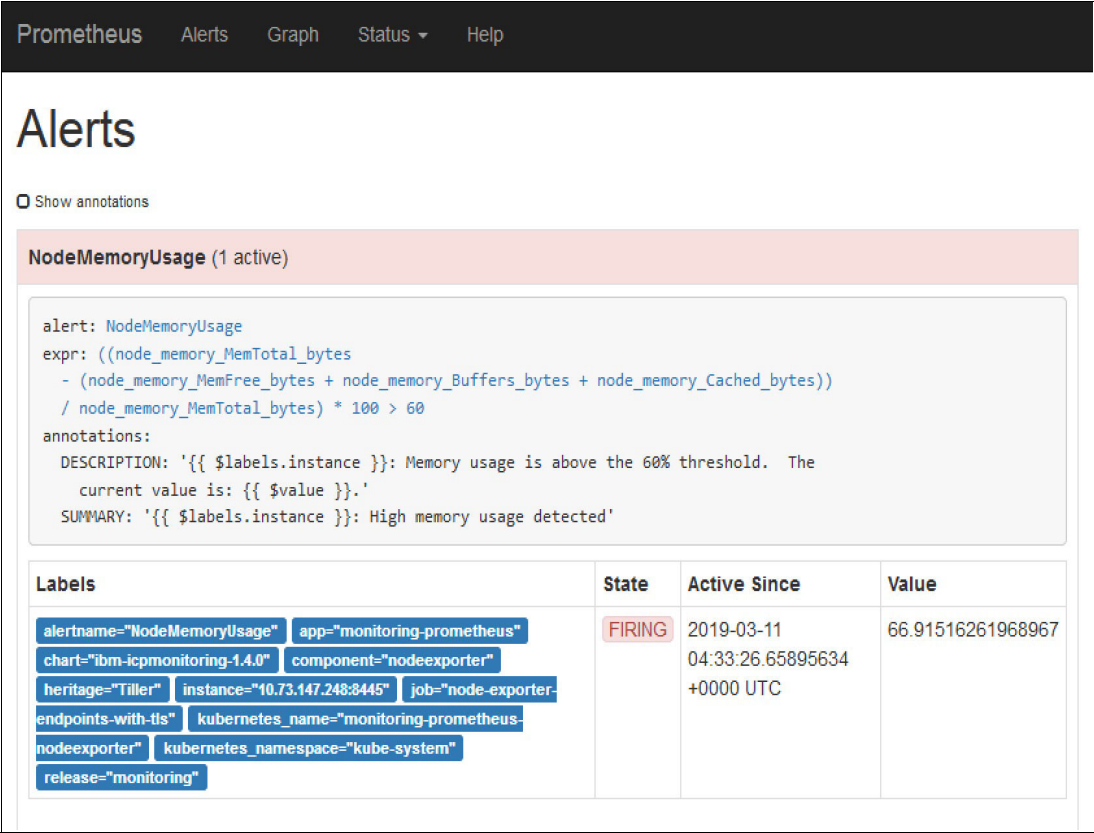


Figure 5-35 Node memory usage alert rule

5.3.7 Configuring Alertmanager to integrate external alert service receivers

IBM Cloud Private has a built-in Alertmanager that will provide the status and details of each triggered alert. You then have the options to view more details, or silence the alert. Figure 5-36 on page 231 shows alert triggered in Alertmanager dashboard.

The screenshot shows the Alertmanager alert details for a triggered alert. At the top, there are tabs for 'Filter' and 'Group'. To the right, it says 'Receiver: All' with checkboxes for 'Silenced' and 'Inhibited'. Below this is a search bar with a '+' button and a placeholder text 'Custom matcher, e.g. env="production"'. The alert name is 'alertname="NodeMemoryUsage"'. The alert was triggered at '04:33:26, 2019-03-11'. There are links for '+ Info', 'Source', and 'Silence'. Below the alert name, there are several labels in a grid: 'release="monitoring"', 'kubernetes_namespace="kube-system"', 'kubernetes_name="monitoring-prometheus-nodeexporter"', 'job="node-exporter-endpoints-with-tls"', 'instance="10.73.147.248:8445"', 'heritage="Tiller"', 'component="nodeexporter"', 'chart="ibm-icpmonitoring-1.4.0"', and 'app="monitoring-prometheus"'. Each label has a '+' button to its right.

Figure 5-36 Alert triggered for NodeMemoryUsage Alert rule

This example will configure Alertmanager to send notifications on Slack. The Alertmanager uses the Incoming Webhooks feature of Slack, so first we need to set that up. Go to the Incoming Webhooks page in the App Directory and click **Install** (or **Configure** and then **Add Configuration** if it is already installed). Once the channel is configured to the incoming webhook, slack will provide a webhook URL. This URL will have to be configured in the Alert rule configuration.

On the IBM Cloud Private boot node (or wherever **kubect1** is installed) run the following command to pull the current ConfigMap data into a local file. Example 5-43 shows how to get the alertmanager ConfigMap.

Example 5-43 Pull the current ConfigMap data into a local file

```
kubect1 get configmap monitoring-prometheus-alertmanager -n kube-system -o yaml >
monitoring-prometheus-alertmanager.yaml
```

Edit monitoring-prometheus-alertmanager.yaml as shown in Example 5-44.

Example 5-44 Slack configuration in Alertmanager ConfigMap

```
apiVersion: v1
data:
  alertmanager.yml: |-
    global:
      receivers:
        - name: default-receiver
          slack_configs:
```

```

      - api_url:
https://hooks.slack.com/services/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
        channel: '#icp-notification'
    route:
      group_wait: 10s
      group_interval: 5m
      receiver: default-receiver
      repeat_interval: 3h
kind: ConfigMap
metadata:
  creationTimestamp: 2019-02-20T17:07:20Z
  labels:
    app: monitoring-prometheus
    chart: ibm-icpmonitoring-1.4.0
    component: alertmanager
    heritage: Tiller
    release: monitoring
  name: monitoring-prometheus-alertmanager
  namespace: kube-system
  resourceVersion: "3894"
  selfLink:
/api/v1/namespaces/kube-system/configmaps/monitoring-prometheus-alertmanager
  uid: fbd6cd7c-3531-11e9-99e8-06d591293f01

```

Example 5-44 on page 231 shows the Alertmanager ConfigMap with updated Slack configuration including the webhook URL and channel name. Save this file and run the command in Example 5-45 to update the Alertmanager configuration.

Example 5-45 Update Alertmanager configuration command

```
kubectl apply -f monitoring-prometheus-alertmanager.yaml
```

Figure 5-37 on page 233 shows that the Node memory usage alert is sent as a notification on Slack for the operations teams to look at.

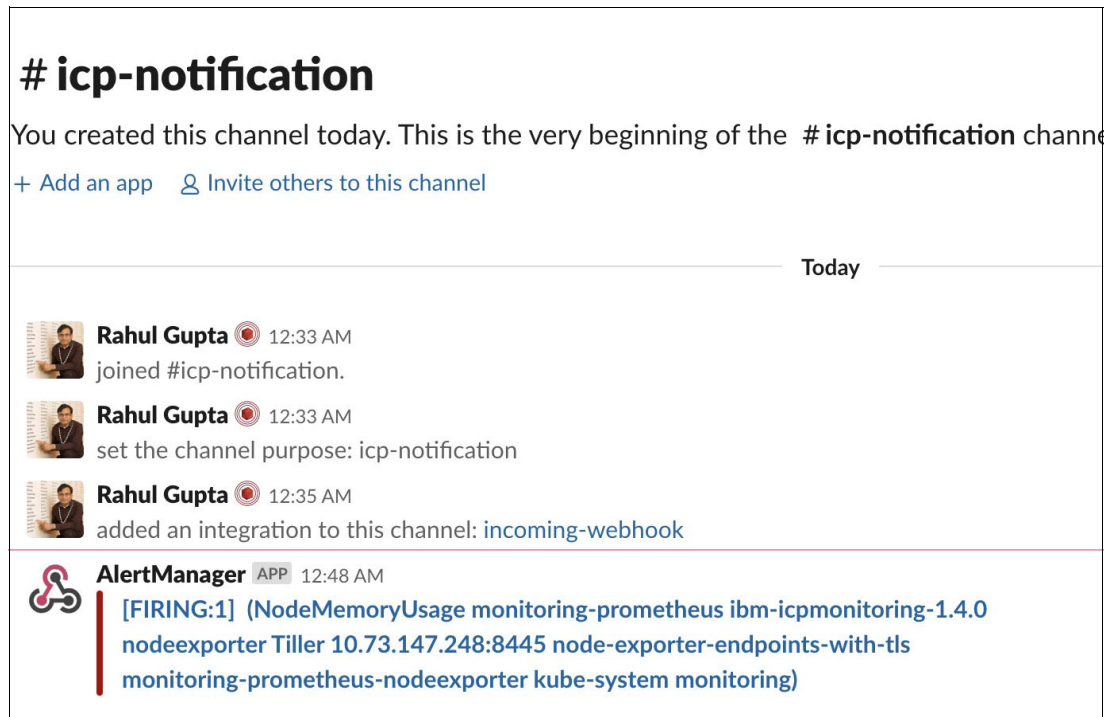


Figure 5-37 Node memory usage notification on Slack

5.3.8 Using Grafana

As discussed in section 5.3.3, “How Grafana works” on page 225, users can use various types of dashboards using the Prometheus datasource. Grafana is already configured to use the Prometheus timeseries datasource.

Users can use existing dashboards like Prometheus stats to see various statistics of Prometheus monitoring system. Figure 5-38 shows the Prometheus statistics dashboard.

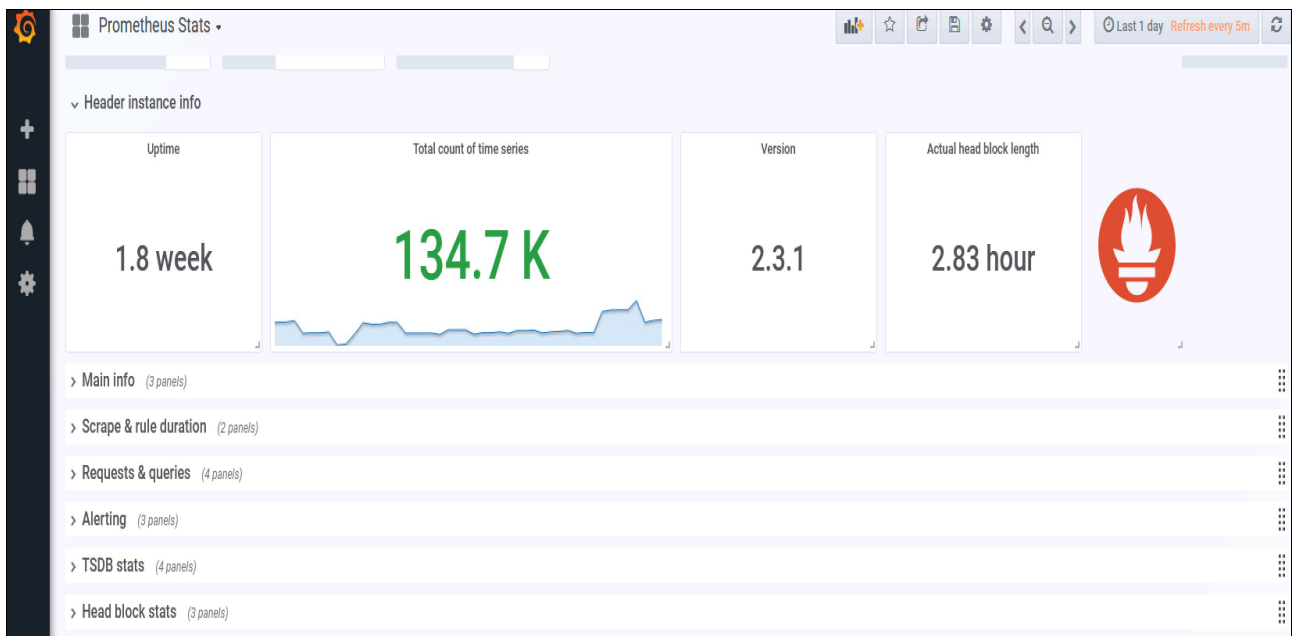
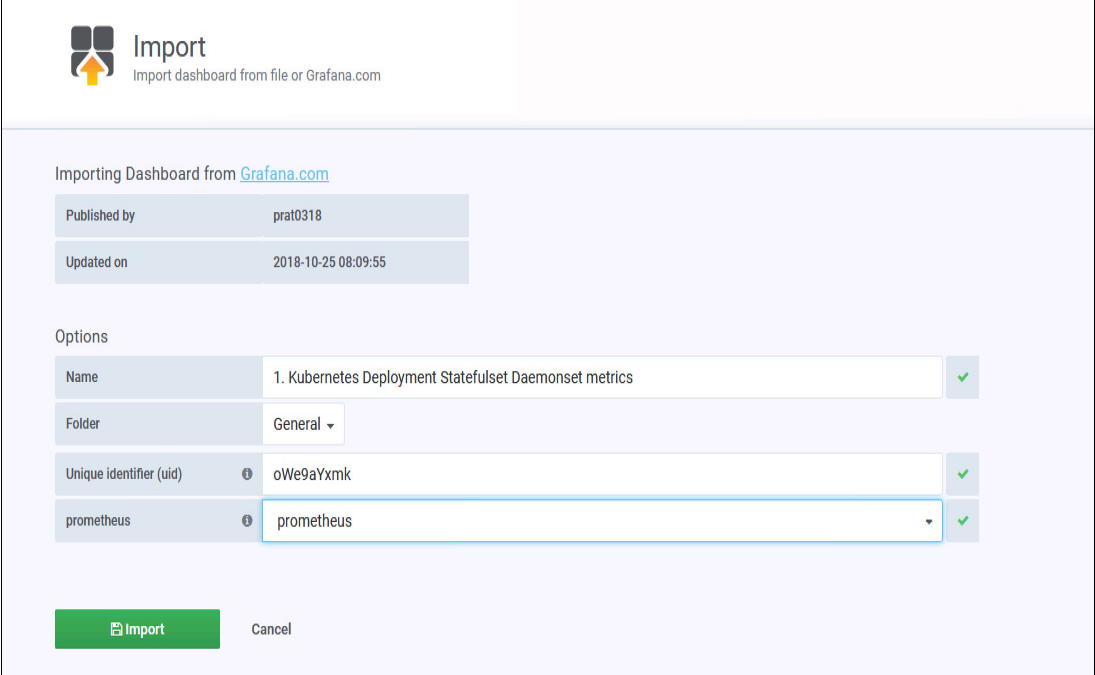


Figure 5-38 Prometheus Statistics dashboard in Grafana

To import dashboards available on Grafana website, click **Import** and add the Grafana dashboard URL or ID. In this case we want to import a dashboard for Kubernetes Deployment, Statefulset, and Daemonset metrics from URL below.

<https://grafana.com/dashboards/8588>

Figure 5-39 shows how to import an external Prometheus dashboard published on grafana.com.



The image shows the Grafana 'Import' interface. At the top, there's a header with the Grafana logo and the word 'Import' in a large font, with a subtitle 'Import dashboard from file or Grafana.com'. Below this, a section titled 'Importing Dashboard from [Grafana.com](#)' contains a table with the following data:

Published by	prat0318
Updated on	2018-10-25 08:09:55

Below the table is an 'Options' section with four rows, each with a label, a value field, and a green checkmark icon:

Name	1. Kubernetes Deployment Statefulset Daemonset metrics	✓
Folder	General ▼	
Unique identifier (uid)	oWe9aYxmk	✓
prometheus	prometheus ▼	✓

At the bottom, there are two buttons: a green 'Import' button and a grey 'Cancel' button.

Figure 5-39 Import and external Prometheus dashboard

Configure the datasource to Prometheus and set unique identifier for the dashboard. You can alternatively import the dashboard by exporting the JSON from grafana.com and importing inside your grafana user interface.

This dashboard should now be available in the list of dashboards. Figure 5-40 shows the Kubernetes Deployment Statefulset Daemonset metrics dashboard.

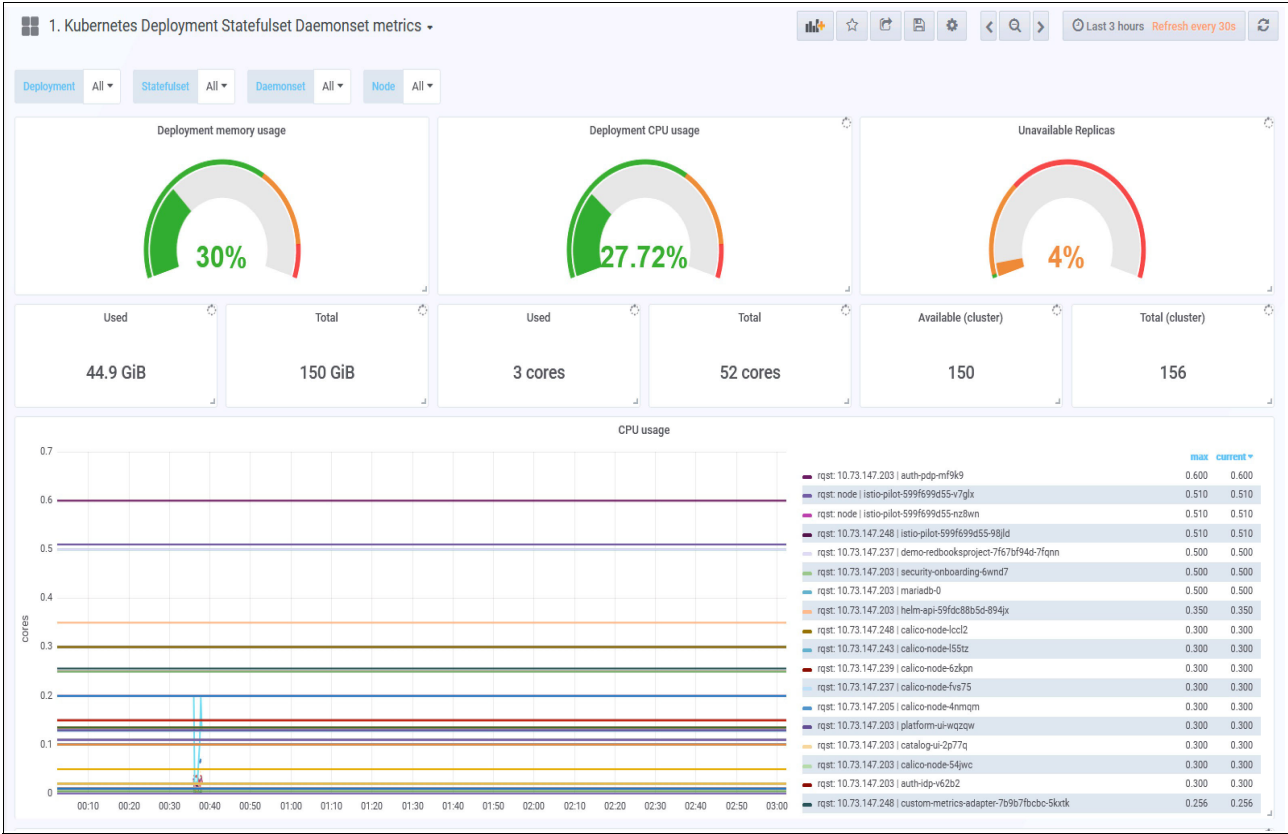


Figure 5-40 Kubernetes Deployment Statefulset Daemonset metrics dashboard



Security

This chapter describes some of the recommended practices for implementing IBM Cloud Private security. This section has the following sections:

- ▶ 6.1, “How IBM Cloud Private handles authentication” on page 238
- ▶ 6.2, “How authorization is handled in IBM Cloud Private” on page 239
- ▶ 6.3, “Isolation on IBM Cloud Private” on page 241
- ▶ 6.4, “The significance of the admission controller in IBM Cloud Private” on page 246
- ▶ 6.5, “Image security” on page 248

6.1 How IBM Cloud Private handles authentication

There are two types of entities (clients) that can authenticate with IBM Cloud Private. These are:

- ▶ Actual humans (users)
- ▶ Pods (service account)

Users are meant to be managed by an external system, such as a LDAP (Lightweight Directory Access Protocol), but pods use a mechanism called the *service account*, which is created and stored in the cluster as a service account resource.

Pods can authenticate by sending the contents of the file `/var/run/secrets/kubernetes.io/serviceaccount/token`, which is mounted into each container's file system through a secret volume. Every pod is associated with a service account, which represents the identity of the app running in the pod. The token file holds the service account's authentication token. Service accounts are resources just like pods, secrets, configmaps, and so on, and are scoped to the individual namespaces. A default service account is automatically created for each namespace.

You can assign a service account to a pod by specifying the account's name in the pod manifest. If you don't assign it explicitly, the pod will use the default service account in the namespace.

IBM Cloud Private supports the following two authentication protocols for users:

- ▶ OIDC (OpenID Connect) based authentication
- ▶ SAML (Security Assertion Markup Language) based federated authentication

6.1.1 OIDC-based authentication

IBM Cloud Private provides an OIDC-based authentication through the WebSphere Liberty Server. This is backed by the Liberty-based OIDC server for providing local and LDAP directory-based authentication.

Lightweight Directory Access Protocol (LDAP) support

IBM Cloud Private can be configured with a single or multiple LDAP servers for authentication and authorization. IBM Cloud Private supports the following LDAP types:

- ▶ IBM Tivoli® Directory Server
- ▶ IBM Lotus® Domino®
- ▶ IBM SecureWay Directory Server
- ▶ Novell eDirectory
- ▶ Sun Java System Directory Server
- ▶ Netscape Directory Server
- ▶ Microsoft Active Directory

With IBM Cloud Private, you can authenticate across multiple LDAPs. You can add multiple directory entries to the LDAP config in the *server.xml* file. Liberty automatically resolves the domain name from the login and authenticates against the targeted LDAP directory. IBM Cloud Private users and user groups are associated with an enterprise directory during the time of the user and user group onboarding via import. When the new LDAP directory entry is created, the domain name also gets added as a new entry. At the time of login, you can specify the domain against which this authentication should be validated.

It is possible to have a mix of directory types, for example Active Directory, IBM Tivoli Directory Server and OpenLDAP. Role-based access control (RBAC) is enforced on the LDAP domain. Cluster administrators have access to all LDAP domains, whereas team administrators are restricted to only those domains they are authorized to.

For more information on configuring LDAP connection with IBM Cloud Private see the following document:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/user_management/configure_ldap.html?view=kc

6.1.2 SAML-based authentication

IBM Cloud Private can be configured to use SAML (Security Assertion Markup Language) based authentication from an enterprise SAML server. The steps to configure SAML-based federated authentication with IBM Cloud Private are discussed in the following document:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/user_management/saml_config.html.

For configuring single sign-on see the following link:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/user_management/saml.html.

6.2 How authorization is handled in IBM Cloud Private

IBM Cloud Private supports role-based access control (RBAC) as the authorization mechanism. RBAC is enforced in IBM Cloud Private through *teams*. A team is an entity that groups users and resources. The resources can be Kubernetes type resources such as namespace, pod, and broker or a non-Kubernetes type, such as the Helm chart, DB instance, and cloud connection. The assignment of the resources to the team happens through the resource CRNs. The responsible services have to expose the resource CRNs through an API so that they become available on the team's Add Resource dialogue.

The Kubernetes resources, such as namespaces, are exposed through the <https://icp-ip:8443/idmgmt/identity/api/v1/k8resources/getK8Resources?resourceType=namespaceAPI>.

It is possible to fetch the resources that are attached to a specific user through their teams by using the <https://icp-ip:8443/idmgmt/identity/api/v1/users/{id}/getTeamResourcesAPI>.

6.2.1 Cloud resource names (CRN) specification

IBM Cloud Private follows the CRN convention:

crn:version:cname:ctype:service-name:region:scope:service-instance:resource-type:resource-instance.

Let us take an example. We will create a team in IBM Cloud Private, onboard LDAP user in the team and assign a role to the user.

1. Create a team in IBM Cloud Private UI name it icp-team as shown in Figure 6-1.

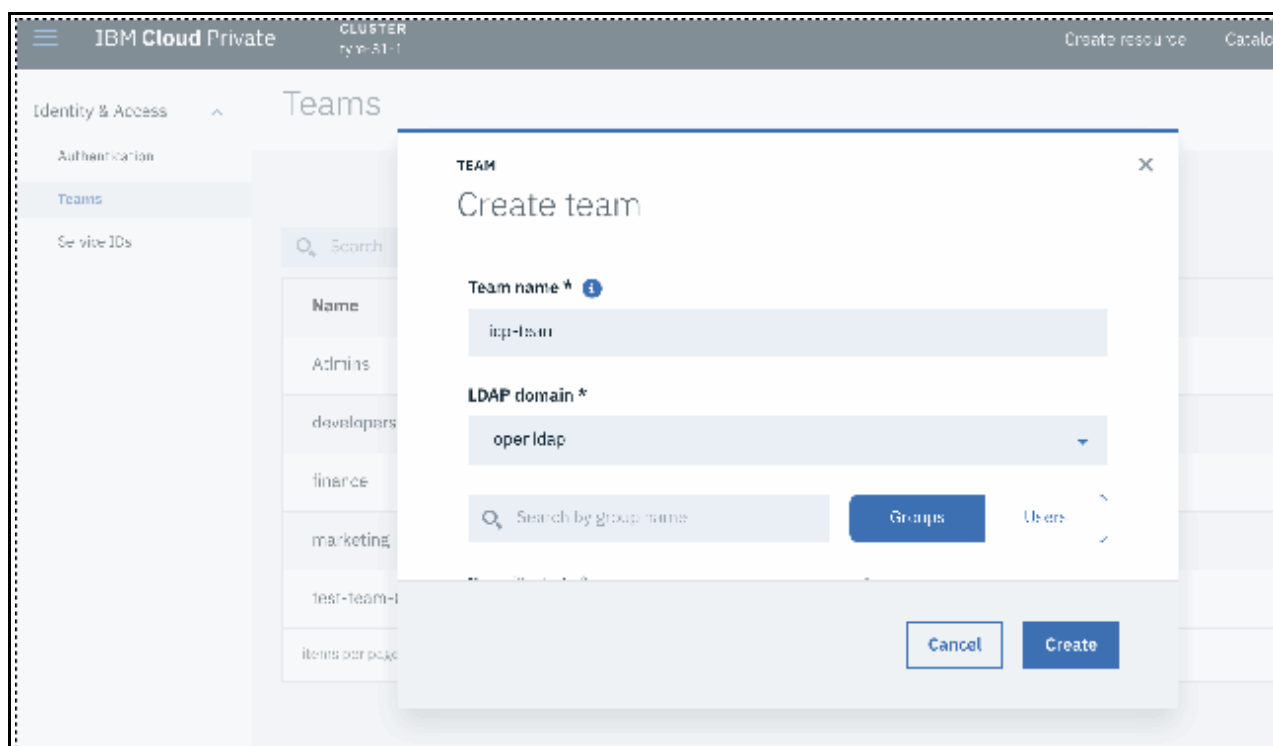


Figure 6-1 Create team icp-team

2. Add user carlos to the team icp-team and assign the Administrator role for this team. See Figure 6-2.

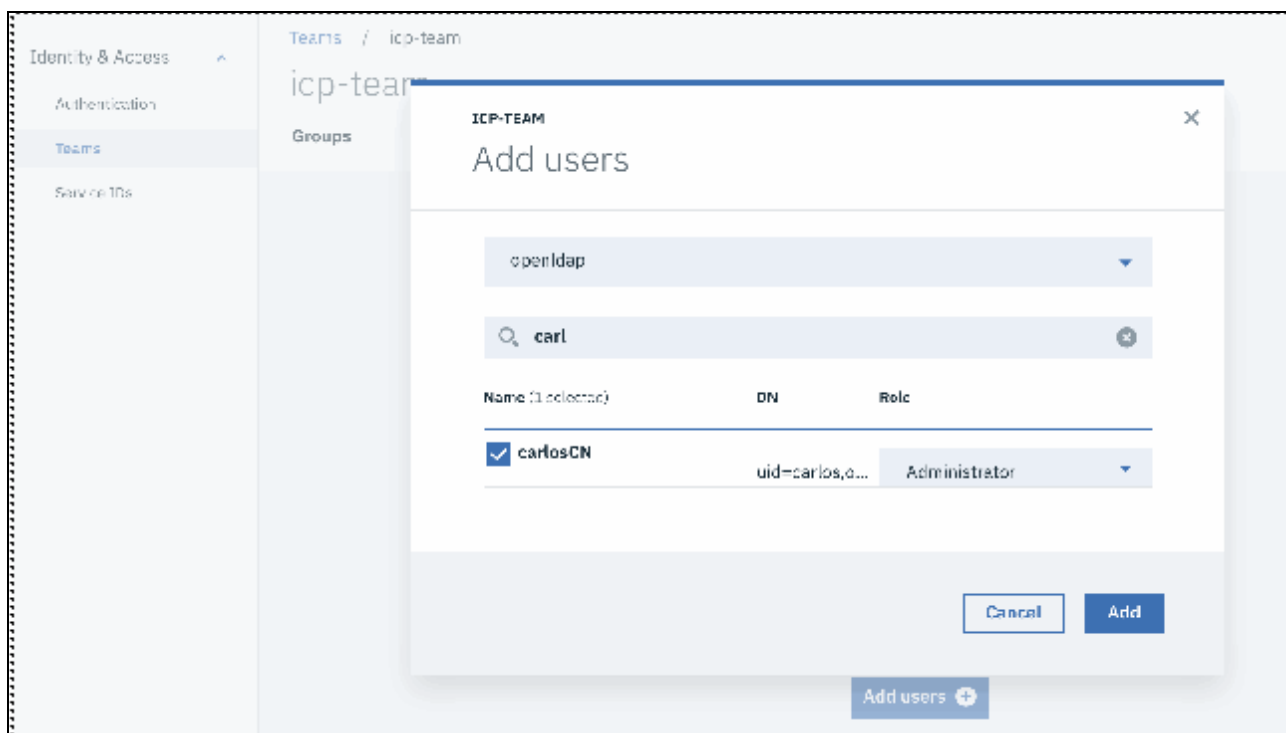


Figure 6-2 Add user carlos to team icp-team

3. Assign namespace development to icp-team as shown in Figure 6-3.

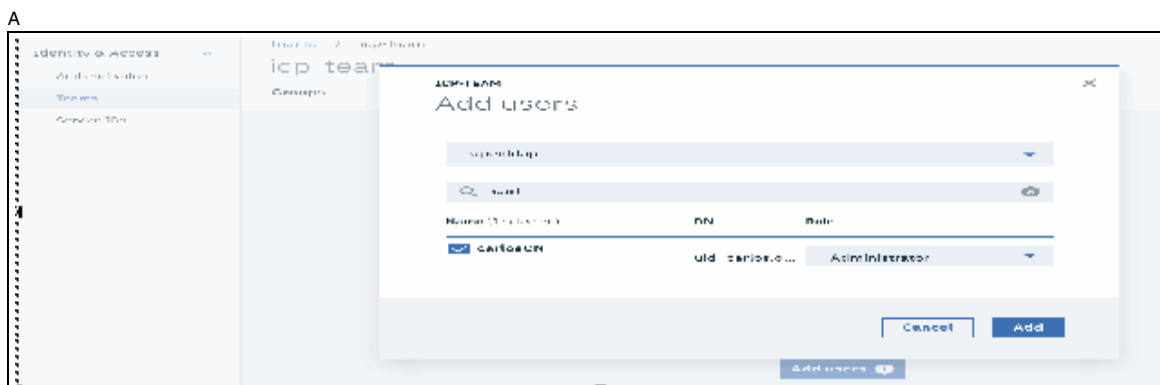


Figure 6-3 Assign namespace development to iicp-team

This will conclude the scenario. For details on which role has permission to perform actions on which resources, see the url below:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/user_management/assign_role.html

Within a team, each user or user group can have only one role. However, a user might have multiple roles within a team when you add a user both individually and also as a member of a team's group. In that case, the user can act based on the highest role that is assigned to the user. For example, if you add a user as an Administrator and also assign the Viewer role to the user's group, the user can act as an Administrator for the team.

6.2.2 Role-based access control (RBAC) for pods

Every pod has an associated service account. Always associate service account with a fine-grained RBAC policy. Only grant this service account the actions and resources that the pod requires to function properly.

6.3 Isolation on IBM Cloud Private

IBM Cloud Private offers multi-tenancy support through user, compute, and network isolation within a cluster. Dedicated physical and logical resources are required for the cluster to achieve workload isolation. Multi-tenancy requires applying various isolation techniques that are described in this topic. User, compute, and network isolation are enforced by confining workload deployments to virtual and physical resources. The enforced isolation also allows the cluster administrator to control the footprint that is allocated to various teams based on their requirements.

The following are some of the key prerequisites to achieve isolation of deployments on cluster nodes.

IBM Cloud Private provides several levels of multi-tenancy. The cluster administrator must analyze workload requirements to determine which levels are required. The following isolation features can be used to satisfy these requirements:

- **Host groups:** As part of preinstall configuration, the cluster administrator can configure groups of nodes to worker host groups and proxy host groups. This operation also involves pre-planning the namespaces, as each host group is mapped to a namespace.

- ▶ **VLAN subnet:** The network infrastructure administrator can plan various subnet ranges for each node or host groups before IBM Cloud Private installation.
- ▶ **Multiple LDAP supports:** Multiple LDAP servers can be configured and the cluster administrator can form teams of users and user groups from various LDAP domains. For the steps to add multiple LDAP registration see https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/user_management/iso_ldap.html.
- ▶ **Namespaces:** The cluster administrator can create namespaces for logical grouping of resources. These namespaces can be created after the IBM Cloud Private installation. If the cluster administrator chooses to have host groups, then the namespace planning is done before installation.
- ▶ **Network ingress controllers:** The cluster administrator must plan the ingress controllers before installation to allow the installer to create ingress controllers for each controller that are mapped to one host group and one namespace.
- ▶ **Users, user groups and teams:** The users and user groups can be on boarded on to IBM Cloud platform and they can be grouped in teams that are mapped to namespaces and other resources.
- ▶ **Network policies:** Team administrators and operators can create network policies to create firewall rules at the namespace scope.
- ▶ **Pod security policies:** The cluster administrator can create policies that either allow or deny container images from running in select namespaces or nodes.

6.3.1 Scenarios

Let us consider the following scenario. In an organization there are two teams, team1, team2 who want to enforce user, compute, and network isolation by confining workload deployments to virtual and physical resources. During the IBM Cloud Private installation we created isolated worker and proxy nodes. Post installation, we onboarded team members from both teams in IBM Cloud Private and assigned them namespaces ns-team1, ns-team2 respectively.

- ▶ Namespace ns-team1 is bound to the worker node workerteam1 and the proxy node proxyteam1.
- ▶ Namespace ns-team2 is bound to the worker node workerteam2 and the proxy node proxyteam2.

Example 6-1 shows that team1 and team2 has dedicated worker and proxy nodes.

Example 6-1 Output of the kubectl get nodes command

```
root@acerate1:~# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
172.16.16.233	Ready	workerteam2	19h	v1.12.4+icp-ee
172.16.16.234	Ready	proxyteam2	19h	v1.12.4+icp-ee
172.16.236.223	Ready	etcd,management,master,proxy,worker	20h	v1.12.4+icp-ee
172.16.237.124	Ready	workerteam1	19h	v1.12.4+icp-ee
172.16.237.225	Ready	proxyteam1	19h	v1.12.4+icp-ee

For both teams users and groups information are onboarded from LDAP into IBM Cloud Private. Deployment that is done from users of team1 will go under the namespace ns-team1 and will be deployed on the worker node workerteam1 and use the proxy node proxyteam1.

Deployment that is done from users of team2 will go under the namespace ns-team2 and will be deployed on the worker node workerteam2 and use the proxy node proxyteam2. Figure 6-4 shows that team1 and team2 from LDAP are onboarded in IBM Cloud Private.

Teams

Q

Search

Name	Groups	Users
team1	1	0
team2	1	0

Figure 6-4 Team1 and team2 from LDAP have been onboarded

So, in the above scenario we were able to achieve the following isolation features:

- ▶ **Host groups:** We are able to isolate proxy node and worker node for each team.
- ▶ **VLAN subnet:** Worker and proxy nodes are in same subnet for each team. Team1 is using the subnet 172.16.236.0/24 and team2 is using the subnet 172.16.16.0/24.
- ▶ **Namespaces:** Both teams have been assigned to different namespaces for logical grouping of resources. This means team1 has been assigned to the namespace ns-team1 and team2 has been assigned to the namespace ns-team2.
- ▶ **Network ingress controllers:** Both teams have isolated proxy nodes, so they will use an ingress controller from their respective proxy nodes.
- ▶ **Users, user groups and teams:** Both teams can onboard any group and users from LDAP in a team, Cluster administrator can create new teams.

Note that the pods deployed from team1 can still talk to the pods of team2. For example both teams deployed the node js sample application from the IBM Cloud Private Helm chart. To stop the communication between the pods of team1 and team2 we should execute the following steps in this order.

See Example 6-2, Example 6-3, Example 6-4 on page 244, Example 6-5 on page 244, and Example 6-6 on page 244. First we start with some prerequisite steps.

Example 6-2 Getting he details of the pod deployed by team1

```

root@aceratel1:~# kubectl get po -n ns-team1 -o wide | awk {' print $1" " $6" " $7'} | column -t
NAME                                     IP      NODE
nodejs-deployment-team1-nodejssample-nodejs-c856dff96-z84gv  10.1.5.5  172.16.237.124

```

Example 6-3 Getting the details of the pod deployed by team2

```

root@aceratel1:~# kubectl get po -n ns-team2 -o wide | awk {' print $1" " $6" " $7'} | column -t
NAME IP      NODE
nodejs-deployment-team2-nodejssample-nodejs-7c764746b9-1mrcc  10.1.219.133  172.16.16.233

```

Example 6-4 Getting the service details of the pod deployed by team1

```
root@aceratel:~# kubectl get svc -n ns-team1 | awk {' print $1" " $5'} | column -t
NAME                                PORT(S)
nodejs-deployment-team1-nodejssample-nodejs  3000:31061/TCP
```

Example 6-5 Getting the service details of the pod deployed by team2

```
root@aceratel:~# kubectl get svc -n ns-team2 | awk {' print $1" " $5'} | column -t
NAME                                PORT(S)
nodejs-deployment-team2-nodejssample-nodejs  3000:30905/TCP
```

Example 6-6 Accessing the pod of team2 from the pod of team1

```
root@aceratel:~# kubectl exec -it nodejs-deployment-team1-nodejssample-nodejs-c856dff96-z84gv -n
ns-team1 -- /bin/bash -c "curl 10.1.219.133:3000"
<!--
  Licensed Materials - Property of IBM
  (C) Copyright IBM Corp. 2018. All Rights Reserved.
  US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP
Schedule Contract with IBM Corp.
-->
<!DOCTYPE html>
<html lang="en">
<head>
....
....
<div class="footer">
    <p>Node.js is a trademark of Joyent, Inc. and is used with its permission. We are not
endorsed by or affiliated with Joyent.</p>
    </div>
  </div>
</body>
```

At this point, we need to create network policies to create firewall rules at the namespace scope. This will stop the communication between pods of team1 and team2. Example 6-7, Example 6-8, Example 6-9 on page 245, Example 6-10 on page 245, and Example 6-11 on page 245 show how to do it.

Example 6-7 Patching the namespace for team1 with label name: ns-team1

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns-team1
  labels:
    name: ns-team1
```

Example 6-8 Patching the namespace for team2 with label name: ns-team2

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns-team2
  labels:
    name: ns-team2
```

Example 6-9 Creating a network policy for team1 to stop communication from any pods except its own pods

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: networkpolicy-team1
  namespace: ns-team1
spec:
  policyTypes:
  - Ingress
  podSelector: {}
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ns-team1
```

Example 6-10 Creating a network policy for team2 to stop communication from any pods except its own pods

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: networkpolicy-team2
  namespace: ns-team2
spec:
  policyTypes:
  - Ingress
  podSelector: {}
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ns-team2
```

Example 6-11 Trying to access the pod of team2 from the pod of team1 or vice versa will fail

```
root@aceratel1:~# kubectl exec -it nodejs-deployment-team1-nodejssample-nodejs-c856dff96-z84gv -n
ns-team1 -- /bin/bash -c "curl 10.1.219.133:3000"
[curl: (7) Failed to connect to 10.1.219.133 port 3000: Connection timed out]
```

If team1 wants to use a different level of security for its pods then team2, they can create its own pod security policy and bind it with namespace. For detailed steps see the following URL:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.1/user_management/create_namespace_pspbind.html.

6.4 The significance of the admission controller in IBM Cloud Private

After authentication and authorization, the next step that IBM Cloud Private performs is the admission control. By the time we have reached this phase, it has already been determined that the request came from an authenticated user and that the user is authorized to perform this request. What we care about right now is whether the request meets the criteria for what we consider to be a valid request and if not, what actions need to be taken. Should we reject the request entirely, or should we alter it to meet our business policies?

Admission control is where an administrator can really start to wrangle the users' workloads. With admission control, you can limit the resources, enforce policies, and enable advanced features.

In the following sections you can find some examples of the admission controller.

6.4.1 Pod security policy

The pod security policies can be used to enforce container image security for the pods in your cluster. A pod security policy is a cluster level resource that controls the security sensitive aspects of a pod's specification and the set of conditions that must be met for a pod to be admitted into the cluster. The pod security policy is applied to the namespace by creating a *ClusterRoleBinding* or *RoleBinding* with the respective pod security policy *ClusterRole* for all service accounts in the namespace.

The pod security policies allow cluster administrators to create pod isolation policies and assign them to namespaces and worker nodes. IBM Cloud Private provides predefined policies that you can apply to your pod by associating them with a namespace during the namespace creation. These predefined pod security policies apply to most of the IBM content charts.

The following list shows the types and descriptions that range from the most restrictive to the least restrictive:

- **ibm-restricted-psp:** This policy requires pods to run with a non-root user ID, and prevents pods from accessing the host.
- **ibm-anyuid-psp:** This policy allows pods to run with any user ID and group ID, but prevents access to the host.
- **ibm-anyuid-hostpath-psp:** This policy allows pods to run with any user ID and group ID and any volume, including the host path.

Attention: This policy allows hostPath volumes. Ensure that this is the level of access that you want to provide.

- **ibm-anyuid-hostaccess-psp:** This policy allows pods to run with any user ID and group ID, any volume, and full access to the host.

Attention: This policy allows full access to the host and network. Ensure that this is the level of access that you want to provide.

- **ibm-privileged-psp:** This policy grants access to all privileged host features and allows a pod to run with any user ID and group ID and any volume.

Attention: This policy is the least restrictive and must be used only for cluster administration. Use with caution.

If you install IBM Cloud Private version 3.1.1, or later as a new installation, the default pod security policy setting is restricted. When it is restricted, the `ibm-restricted-psp` policy is applied by default to all of the existing and newly created namespaces. You can also create your own pod security policy. For more information see the following link:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.1/manage_cluster/enable_pod_security.html.

In the following you can find some of the recommended practices when using the pod security policy:

Running privileged pods separately from un-privileged pods

Unprivileged pods are those that can run with the `ibm-restricted-psp` pod security policy. These containers do not require any elevated privileges and are less likely to affect other containers on the same node. You should separate any pods that require special privileges, especially if the workload is not completely trusted or documented.

Binding pod security policies to namespaces instead of service accounts

Kubernetes does not check for elevated role-based access control (RBAC) permissions when a cluster administrator assigns a service account to a pod. It only checks for elevated permissions when a `RoleBinding` or `ClusterRoleBinding` is created. If a cluster administrator creates several service accounts in a namespace with various levels of privileges, then the namespace is only as secure as the service account that has the most privileges. It is safer and easier for a cluster administrator to examine the security settings of a namespace when a pod security policy is bound to all of the service accounts rather than the individual ones.

Specifying one pod security policy per namespace

The pod security policies are assigned to pods by the pod admission controller based on the user that creates the pod. For pods created by controllers, such as Deployments, the namespace user is the *service account*. If more than one policy matches the pod's declared security context, then any of the policies could match.

Avoid creating pods directly

Pods can be created directly, with the user's credentials. This can circumvent the pod security policy that is bound to the service accounts in the target namespace. A cluster administrator can run a privileged pod in a namespace that is configured as an un-privileged namespace.

6.4.2 ResourceQuota

This admission controller will observe the incoming requests and ensure that they do not violate any of the constraints enumerated in the `ResourceQuota` object in a namespace.

6.4.3 LimitRange

This admission controller will observe the incoming request and ensure that it does not violate any of the constraints enumerated in the LimitRange object in a namespace.

6.4.4 AlwaysPullImages

This admission controller modifies every new pod to force the image pull policy to Always. This is useful in a multitenant cluster so that users can be assured that their private images can be used only by those who have the credentials to pull them. Without this admission controller, once an image has been pulled to a node, any pod from any user can use it simply by knowing the image's name (assuming the pod is scheduled onto the right node), without any authorization check against the image. When this admission controller is enabled, images are always pulled prior to starting containers, which means valid credentials are required.

IBM Cloud Private supports all of the Kubernetes admission controllers. For more details see the following link:

<https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/>

6.5 Image security

IBM Cloud Private component Image Manager runs on top of the Docker registry V2 API. It integrates with the Docker registry to provide a local registry service. The Image Manager uses the cluster's authentication service to authenticate the end user. The Docker command line client is used to push or pull images in your cluster.

Figure 6-5 on page 249 shows the Image Manager architecture.

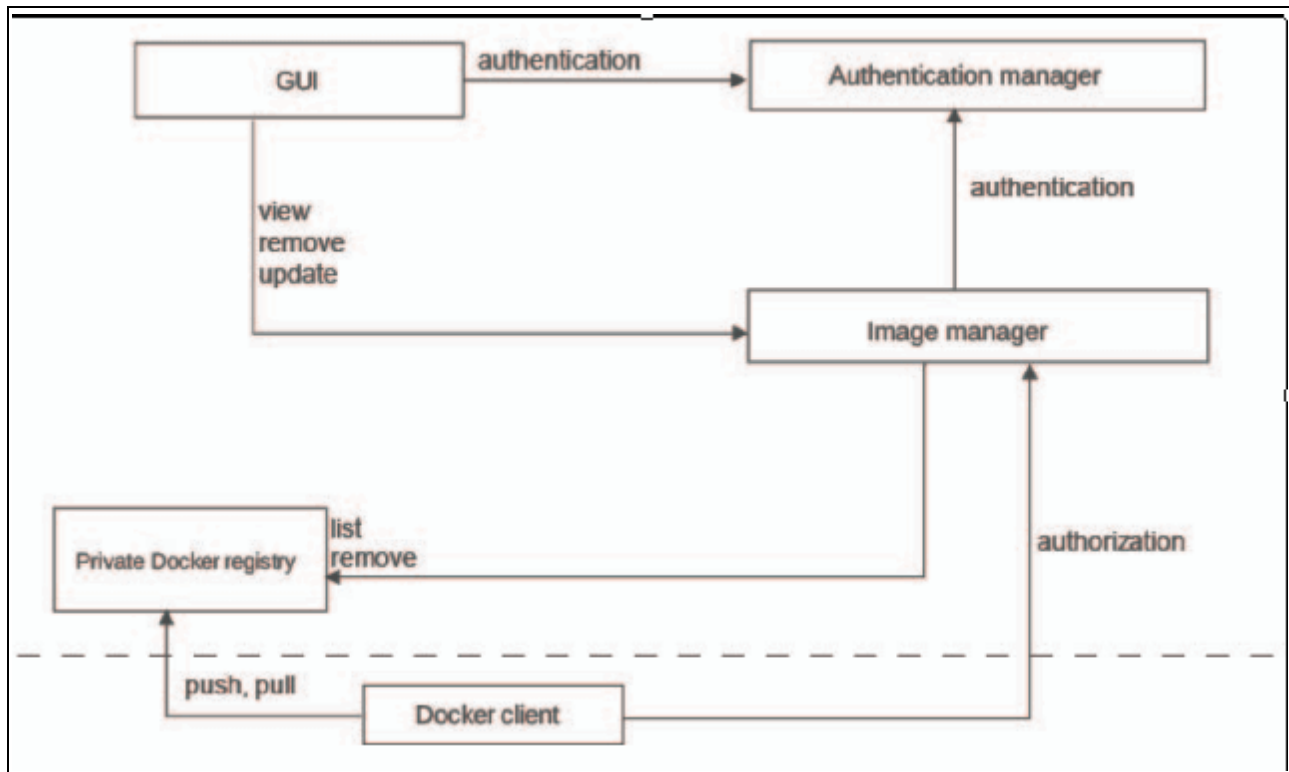


Figure 6-5 Image Manager architecture

6.5.1 Pushing and pulling images

In order to push or pull an image we need to log in to our private image registry.

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

<cluster_CA_domain> is the certificate authority (CA) domain that was set in the config.yaml file during installation. If you did not specify a CA domain name, the default value is mycluster.icp.

You can push or pull the image only if the namespace resource is assigned to a team for which you have the correct role. Administrators and operators can push or pull the image. Editors and viewers can pull images. Unless you specify an imagePullSecret, you can access the image only from the namespace that hosts it.

We can push or pull the image only if the namespace resource is assigned to a team for which you have the correct role. Administrators and operators can push or pull the image. Editors and viewers can pull images. Unless you specify an imagePullSecret, you can access the image only from the namespace that hosts it.

The service account defined in a PodSpec can pull the image from the same namespace under the following conditions:

- ▶ The PodSpec is using default service account.
- ▶ The service account is patched with a valid image pull secret.
- ▶ The PodSpec includes the name of a valid image pull secret.
- ▶ The image scope is changed to global after the image is pushed.

How to change image scope?

There are two ways to change the image scope:

Change image scope using the command line

To change the scope from namespace to global, run the following command:

```
kubect1 get image <image name> -n=namespace -o yaml \  
| sed 's/scope: namespace/scope: global/g' | kubect1 replace -f -
```

To change the scope from global to namespace, run the following command:

```
kubect1 get image <image name> -n=namespace -o yaml \  
| sed 's/scope: global/scope: namespace/g' | kubect1 replace -f -
```

where <image name> value is the name of the image for which scope is changed.

Change the image scope using IBM Cloud Private UI

Perform the following steps to change the image scope using IBM Cloud Private UI:

1. From the navigation menu, click **Container Images**.
2. For the image that you want to update, click **Open** and close the List of options button and select **Change Scope**.
3. Select the scope from the drop-down menu in the Image dialog box.
4. Click **Change Image Scope**.

6.5.2 Enforcing container image security

Using the container image security enforcement feature, IBM Cloud Private can verify the integrity of a container image before it is deployed to an IBM Cloud Private cluster. For each image in a repository, an image policy scope of either the cluster or the namespace is applied. When you deploy an application, IBM container image security enforcement checks whether the namespace that you are deploying to has any policy regulations that must be applied.

If a namespace policy does not exist, then the cluster policy is applied. If the namespace policy and the cluster policy overlap, the cluster scope is ignored. If neither a cluster or namespace scope policy exists, your deployment fails to start.

Pods that are deployed to namespaces, which are reserved for the IBM Cloud Private services, bypass the container image security check.

The following namespaces are reserved for the IBM Cloud Private services:

- ▶ kube-system
- ▶ cert-manager
- ▶ istio-system

Example 6-12 shows a sample image policy.

Example 6-12 Sample image policy.

```
apiVersion: securityenforcement.admission.cloud.ibm.com/v1beta1  
kind: <ClusterImagePolicy_or_ImagePolicy>  
metadata:  
  name: <crd_name>  
spec:  
  repositories:  
    - name: <repository_name>
```



```
policy:
  va:
    enabled: <true_or_false>
```

In this example, `repository_name` specifies the name of repository from which the image will be pulled from. A wildcard (*) character is allowed in the repository name. This wildcard (*) character denotes that the images from all of the repositories are allowed or trusted. To set all your repositories to trusted, set the repository name to (*) and omit the policy subsections.

Repositories by default require a policy check, with the exception of the default `mycluster.icp:8500` repository. An empty or blank repository name value blocks deployment of all the images.

When `va` is set to `enabled: true` (See Example 6-12 on page 250), vulnerability advisor policy is enforced. It works only for the default IBM Cloud Private built-in container registry. With the other image registries this option should be false, otherwise the image will not be pulled.

`ClusterImagePolicy` or `ImagePolicy` for a namespace can be viewed or edited like any other Kubernetes object.

Default image security enforcement

Since the release of IBM Cloud Private Version 3.1.1, IBM container Image Security Enforcement is turned on by default. Any image that does not meet the policy will not be deployed successfully.

Let us check this out with an example: Hello-world image enforcement with `kubect1`. We run the sample hello-world docker image, as shown in Example 6-13.

Example 6-13 Hello-world docker image

```
kubect1 run -it --rm imagepolicy --image=hello-world --restart=Never
```

```
Error from server (InternalError): Internal error occurred: admission webhook
"trust.hooks.securityenforcement.admission.cloud.ibm.com" denied the request:
Deny "docker.io/hello-world", no matching repositories in ClusterImagePolicy and
no ImagePolicies in the "default" namespace
```

The security enforcement hook blocks the running of the image. This is a great security feature enhancement that prevents any unwanted images to be run in the IBM Cloud Private cluster.

Now we will create a whitelist to enable this specific docker image from the Docker Hub. For each repository that you want to enable the pulling of an image, you have to define the name for the repository, where the wildcard (*) is allowed. You can also define the policy of the vulnerability advisor (VA). If you enable the VA enforcement as true, then only those images that have passed the vulnerability scanning can be pulled. Otherwise, they will be denied.

Create the following image policy YAML as shown in Example 6-14.

Example 6-14 Image policy yaml

```
apiVersion: securityenforcement.admission.cloud.ibm.com/v1beta1
kind: ImagePolicy
metadata:
  name: my-cluster-images-whitelist
namespace: default
```

```
spec:
  repositories:
    - name: docker.io/hello-world
      policy:
        va:
          enabled: false
```

Here we create an image policy which applies to the default namespace. We give the exact match of the image and set the VA policy disabled. Save it as `image-policy.yaml`, and apply it with the `kubectl apply -f image-policy.yaml` command.

Now run the same command again. You will see the docker's hello world message as shown in Example 6-15.

Example 6-15 Hello world message displayed

```
kubectl run -it --rm imagepolicy --image=hello-world --restart=Never
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

```
pod "imagepolicy" deleted
```

Perform the following steps for the hello-world image enforcement example with dashboard:

1. Login to dashboard.
2. Go to **Manage Resource Security** → **Image Policies**.
3. Select the **my-cluster-images-whitelist** that was created with the `kubectl` command.
4. Remove the policy as shown in Figure 6-6 on page 253.

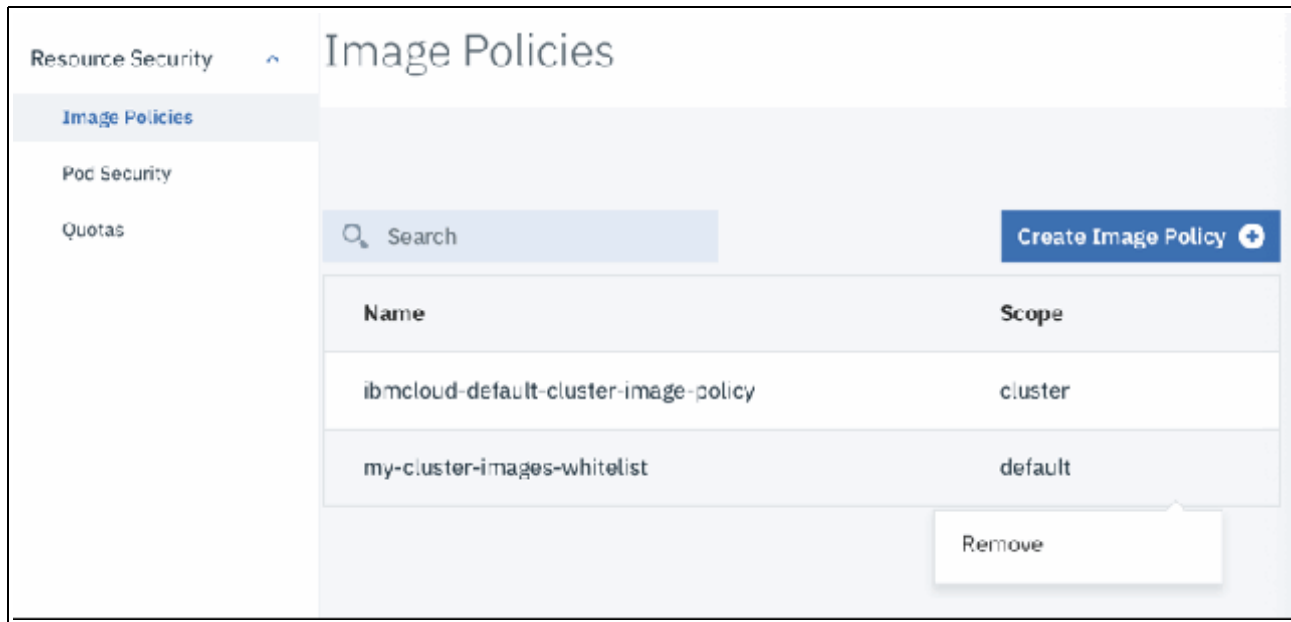


Figure 6-6 Remove the image policy

5. Now the hello-world image will be prevented to run again.
6. Click the **Create Image Policy** button at the up right corner to pop up in Figure 6-7.

The screenshot shows the 'Add Image Policy' form. It has a sidebar on the left with 'Image Policies', 'Pod Security', and 'Quotas'. The main area is titled 'Add Image Policy' and has a 'Cancel' button in the top right corner.

1 Policy Details | Select the type of Image Policy you want to add.

Name *	Scope *
my-white-list	Cluster

2 Registry Policies | Add image registries and configure their policies.

Registry URL	VA scan
docker.io/hello-world	Not enforced

Figure 6-7 Add image policy

7. Give it a name such as "my-white-list".
8. Specify the scope as **Cluster**, which applies the enforcement to the whole cluster. Unlike the above namespace based example, once this policy is created, then any namespace can run the image at the cluster level.
9. Set the VA scan to "not enforced". Currently only the default IBM Private Cloud registry has the VA scanning feature. The rest of the docker registry doesn't have this feature. If you enabled it, none of the images can be run.
10. Click the **Add** button. By adding this, you can run the image at any namespace. As an example see Example 6-16.

Example 6-16 Run the hello-world in kube-public namespace

```
kubectl run -it --namespace=kube-public --rm imagepolicy --image=hello-world
--restart=Never
```

Example 6-17 shows the list of the images that are allowed at the cluster level by default in IBM Cloud Private 3.1.2.

Example 6-17 Default enabled image list in IBM Cloud Private 3.1.2

```
<your icp cluster name>:8500/*
registry.bluemix.net/ibm/*
cp.icr.io/cp/*
docker.io/apache/couchdb*
docker.io/ppc64le/*
docker.io/amd64/busybox*
docker.io/vault:*
docker.io/consul:*
docker.io/python:*

docker.io/centos:*
docker.io/postgres:*
docker.io/hybridcloudibm/*
docker.io/ibmcom/*
docker.io/db2eventstore/*
docker.io/icpdashdb/*
docker.io/store/ibmcorp/*
docker.io/alpine*
docker.io/busybox*
docker.io/duportal/bats:*
docker.io/cassandra:*
docker.io/haproxy:*
docker.io/hazelcast/hazelcast:*
docker.io/library/busybox:*
docker.io/minio/mc:*
docker.io/minio/minio:*
docker.io/nginx:*
docker.io/open-liberty:*
docker.io/openwhisk/*
docker.io/rabbitmq:*
docker.io/radial/busyboxplus:*
docker.io/ubuntu*
docker.io/websphere-liberty:*
docker.io/wurstmeister/kafka:*
docker.io/zookeeper:*
```

```
docker.io/ibmcloudcontainers/strongswan:*
docker.io/opsh2oai/dai-ppc64le:*
docker.io/redis*
docker.io/f5networks/k8s-bigip-ctlr:*
docker.io/rook/rook:*
docker.io/rook/ceph:*
docker.io/couchdb:*
docker.elastic.co/beats/filebeat:*
docker.io/prom/statsd-exporter:*
docker.elastic.co/elasticsearch/elasticsearch:*
docker.elastic.co/kibana/kibana:*
docker.elastic.co/logstash/logstash:*
quay.io/k8scsi/csi-attacher:*
quay.io/k8scsi/driver-registrar:*
quay.io/k8scsi/nfsplugin:*
quay.io/kubernetes-multicluster/federation-v2:*
k8s.gcr.io/hyperkube:*
registry.bluemix.net/armada-master/ibm-worker-recovery:*
```

For any image that is not in the default list, you have to create either a namespace based image policy or a cluster level policy to allow the image to run.



Networking

This chapter provides an overview of the network component in IBM Private Cloud and discusses how communication flows between the pods and the external network and how a pod is exposed to the network.

The chapter has the following sections:

- ▶ 7.1, “Introduction to container networking” on page 258
- ▶ 7.2, “Pod network” on page 259
- ▶ 7.3, “High availability” on page 262
- ▶ 7.4, “Service discovery (kube-dns)” on page 270

7.1 Introduction to container networking¹

A container needs to have at least one network interface. Through this interface, the container communicates with the other containers or endpoints. Kubernetes has its own network on top of the physical network to enable communication between nodes and external systems.

A pod could consist of various containers that are collocated at the same host, share the same network stack, and share other resources (such as volumes). From an application developer point of view, all containers in a Kubernetes cluster are considered on a flat subnet.

Figure 7-1 shows the communication coming from the physical Interface (eth0) going to the Docker0 and then to the virtual network.

Each docker instance creates its own network and enables the pod communication.

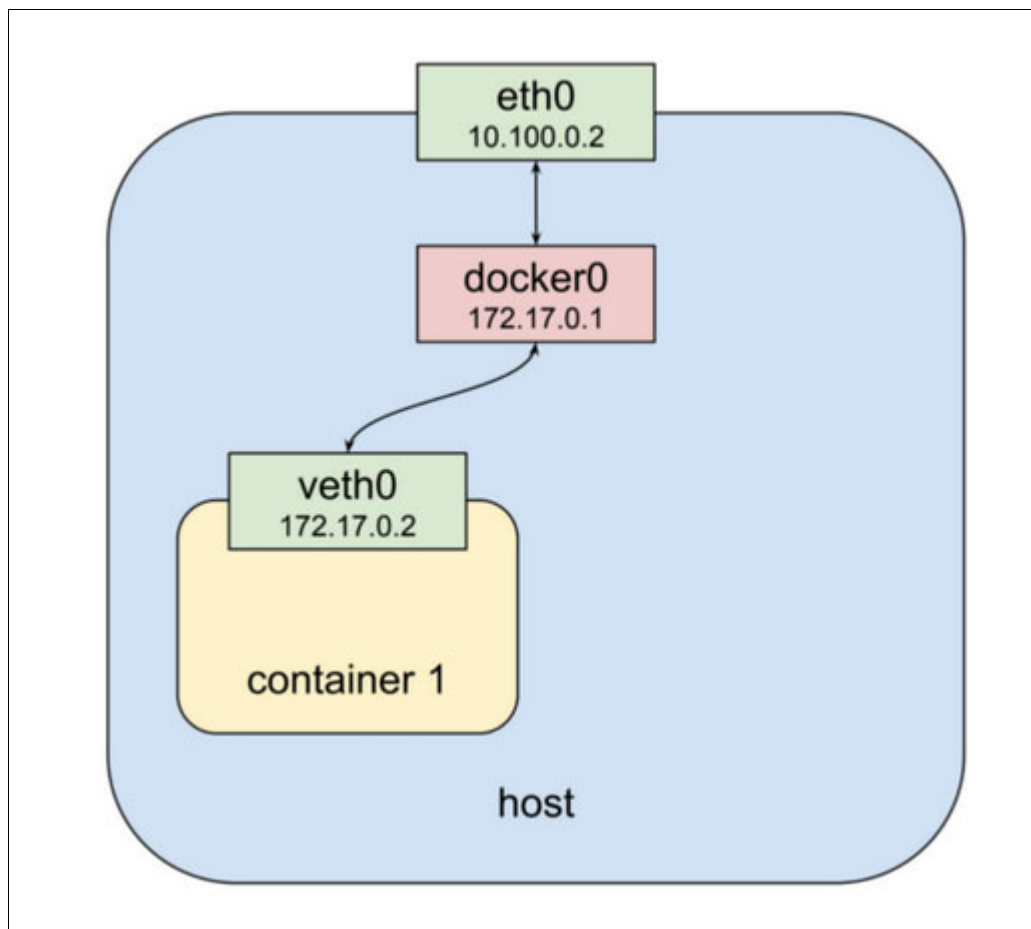


Figure 7-1 Overview of pod network

As shown in Figure 7-1, the communication is bidirectional.

The next sections demonstrate the networking concepts in Kubernetes, such as pod networking, load balancing, and ingress.

¹ Some of the content in this chapter is based on the following GitHub document created by Rachappa Goni, Eswara Kosaraju, Santosh Ananda, and Jeffrey Kwong.

7.2 Pod network

Pod network enables the pods, and all containers that are associated with the pods are network addressable. This basic network is implemented by Kubernetes and allows a pod to communicate with the other pods as though they were on their own dedicated hosts.

It is important to note that all containers in a single pod share the same port space. If container A uses port 80, you cannot have container B inside the same pod that uses port 80 as well. Using the same port would only work if these containers are on different pods. An additional component, called the *pod network namespace*, is provided by the Kubernetes pause container. This component creates and owns the network namespace.

Another component that is present on the pod network is the *container network interface* (CNI). This consists of a plug-in that connects the pod network namespace to the rest of the pods in the Kubernetes cluster. The most widely used CNIs in IBM Cloud Private are the Calico and NSX.

7.2.1 Calico

Calico enables networking and network policy in Kubernetes clusters across the cloud. It combines flexible networking capabilities with run anywhere security enforcement, providing performance similar to a native kernel and enabling real cloud-native scalability. Calico is implemented without encapsulation or overlays, providing high-performance networking. It also provides a Network security policy for Kubernetes pods through its distributed firewall.

When using Calico, policy engine enforces the same policy model at the host networking layers (and at the service mesh if using Istio) helping to protect the infrastructure from workloads from compromised infrastructures.

Because Calico uses the Linux Kernel's forwarding and access control, it provides a high-performance solution without the resources used by encapsulation and decapsulation.

Calico creates a flat layer-3 network, and assigns a fully routable IP address to every pod. To do that, it divides a large network of CIDR (Classless Inter-Domain Routing) into smaller blocks of IP addresses, and assigns one or more of these smaller blocks to the nodes in the cluster. This configuration is specified at the IBM Cloud Private installation time using the **network_cidr** parameter in `config.yaml` in CIDR notation.

By default, Calico creates a BGP (Border Gateway Protocol) mesh between all nodes of the cluster, and broadcasts the routes for container networks to all of the worker nodes. Each node is configured to act as a layer 3 gateway for the subnet assigned to the worker node, and serves the connectivity to pod subnets hosted on the host.

All nodes participate in the BGP mesh, which advertises the local routes that the worker node owns to the rest of the nodes. BGP peers external to the cluster can participate in this mesh as well, but the size of the cluster affects how many BGP advertisements these external peers will receive. Route reflectors can be required when the cluster scales past a certain size.

When routing the pod traffic, Calico uses the system capabilities, such as the node's local route tables and iptables. All pod traffic traverses iptables rules before they are routed to their destination.

Calico maintains its state using an etcd key/value store. By default, in IBM Cloud Private Calico uses the same etcd key/value store as Kubernetes to store the policy and network configuration states.

Calico can be configured to allow pods to communicate with each other with or without IP-in-IP tunneling. IP-in-IP adds an additional header for all packets as part of the encapsulation, but containers can communicate on its overlay network through almost any non-overlapping underlay network. In some environments where the underlay subnet address space is constrained and there is no access to add additional IP Pools, like on some public clouds, Calico can be a good fit.

However, in environments that do not require an overlay, IP-in-IP tunneling should be disabled to remove the packet encapsulation resource use, and enable any physical routing infrastructure to do packet inspection for compliance and audit. In such scenarios, the underlay network should be made aware of the additional pod subnets by adding the underlay network routers to the BGP mesh. See the discussion about this in “Calico components”.

Calico components

The calico network is created by the following 3 components: node agent, CNI, and kube-controller.

Calico/node agent

This entity has three components: felix, bird, and confd.

- ▶ *Felix*’s primary responsibility is to program the host’s iptables and routes to provide the wanted connectivity to and from the pods on that host.
- ▶ *Bird* is an open source BGP agent for Linux that is used to exchange routing information between the hosts. The routes that are programmed by felix are picked up by bird and distributed among the cluster hosts.
- ▶ *Conf*d monitors the etcd data store for changes to the BGP configuration, such as IPAM information and AS number, and accordingly changes the bird configuration files and triggers bird to reload these files on each host. The calico/node creates veth-pairs to connect the Pod network namespace with the host’s default network namespace.

Calico/cni

The CNI plug-in provides the IP address management (IPAM) functionality by provisioning IP addresses for the Pods hosted on the nodes.

Calico/kube-controller

The calico/kube-controller watches Kubernetes NetworkPolicy objects and keeps the Calico data store in sync with the Kubernetes. calico/node runs on each node and uses the information in the Calico etcd data store and program the local iptables accordingly.

Calico network across different network segments

When nodes are on different network segments, they are connected using a router in the underlay infrastructure network. The traffic between two nodes on different subnets happens through the router, which is the gateway for the two subnets. If the router is not aware of the pod subnet, it will not be able to forward the packets between the hosts.

There are two scenarios that can be used to the Calico communication:

- ▶ Calico can be configured to create IP-in-IP tunnel end points on each node for every subnet hosted on the node. Any packet originated by the pod and egressing the node is encapsulated with the IP-in-IP header, and the node IP address is utilized as the source. This way, the infrastructure router does not see the pod IP addresses.

The IP-in-IP tunneling brings in extra resource use in terms of network throughput and latency due to the additional packet resource and processing at each endpoint to encapsulate and decapsulate packets.

On bare metal, the resource use is not significant, because certain network operations can be offloaded to the network interface cards. However, on virtual machines, the resource use can be significant and also affected by the number of CPU cores and network I/O technologies configured and used by the hypervisors. The additional packet encapsulation resources might also be significant when smaller MTU (maximum transmission unit) sizes are used, because it might introduce packet fragmentation; jumbo frames should be enabled whenever possible.

- The second option is to make the infrastructure router aware of the pod network. This can be done by enabling BGP on the router and adding the nodes in the cluster as BGP peers to it. With these actions, the router and the hosts can exchange the route information between each other. The size of the cluster in this scenario might come in to play because in the BGP mesh, every node in the cluster is BGP peering to the router.

7.2.2 NSX-T

NSX-T is a network virtualization and security platform that automates the implementation of network policies, network objects, network isolation, and micro-segmentation.

Figure 7-2 shows an overview of NSX-T.

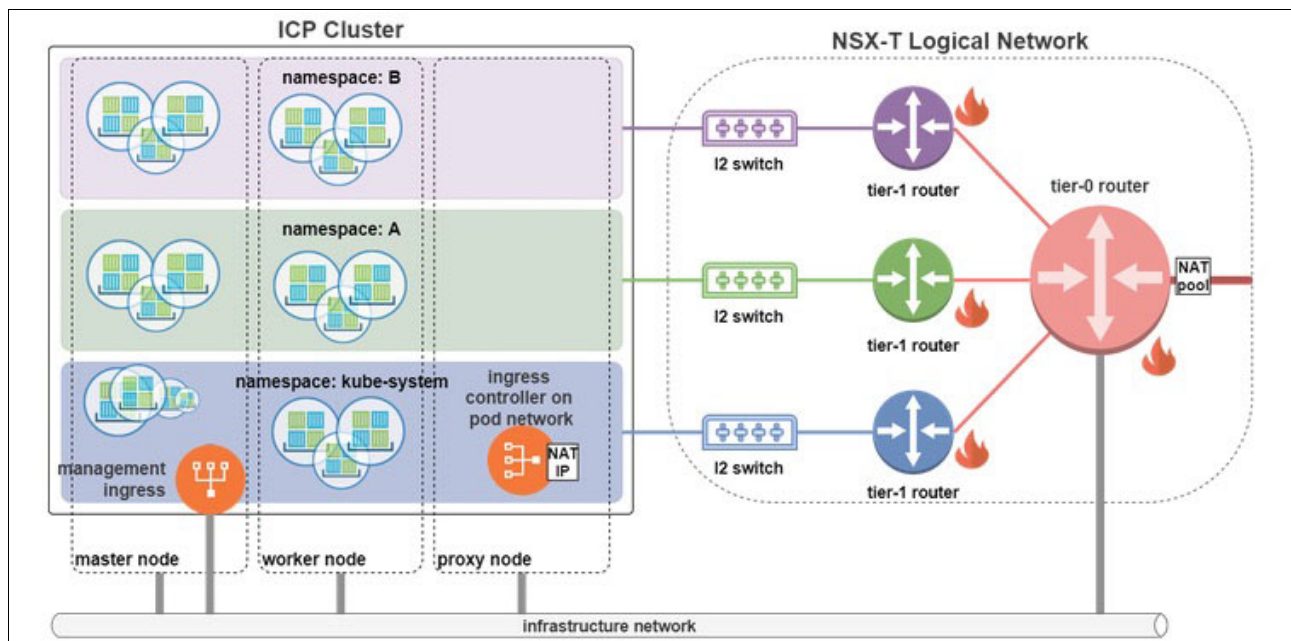


Figure 7-2 NSX-T configuration

NSX-T network virtualization for Kubernetes

The NSX-T network virtualization for Kubernetes consists of L2 and L3 segregation, micro segmentation, and NAT pools.

L2 and L3 segregation

NSX-T creates a separate L2 switch (virtual distributed switch, or VDS) and L3 router (distributed logical router, or DLR) for every namespace. The namespace level router is called a *T1 router*. All T1 routers are connected to the T0 router, which acts like an edge gateway to the IBM Cloud Private cluster, as well as an edge firewall and load balancer. Due to separate L2 switches, all the broadcast traffic is confined to the namespace. In addition, due to the separate L3 router, each namespace can host its own pod IP subnet.

Micro segmentation

NSX-T provides distributed firewall (DFW) for managing the east-west traffic. The Kubernetes network policy gets converted into the NSX-T DFW rules. With L2 segmentation, dedicated L3 subnets for namespaces and network policies, one can achieve micro segmentation within and across a namespace.

NAT pools

Edge appliance is an important component of the NSX-T management cluster. It offers routing, firewall, load balancing, and network address translation, among other features. By creating pods on the NSX-T pod network (and not relying on the host network), all traffic can be traversed through the edge appliance using its firewall, load balancing, and network address translation capabilities.

The edge appliance assigns SNAT (Source Network Address Translation) IPs to the outbound traffic, and DNAT (Destination Network Address Translation) IPs to the inbound traffic from the NAT pool (created as part of the NSX-T deployment). By relying on the network address translation, the cluster node IPs are not exposed on the outbound traffic.

7.3 High availability

For high availability, master and proxy nodes should be deployed redundantly (at different physical locations, if possible) to tolerate hardware failures and network partitions. The following discusses options for network high availability considerations in IBM Cloud Private.

7.3.1 External load balancer

If possible, a highly available external load balancer should be leveraged to spread the traffic among separate master or proxy node instances in the cluster. The external load balancer can either be a DNS URL or an IP address, and specified using `cluster_lb_address` at `config.yaml` during install time. The `cluster_CA_domain` and any TLS certificates should be configured to be a CNAME (Canonical Name Record) or a record pointing at the external load balancer DNS name or IP address. In addition, all nodes in the cluster should be able to resolve this CNAME for internal communication.

When using an external load balancer, the master load balancer should monitor the Kubernetes API server port 8001 for health on all master nodes, and the load balancer needs to be configured to accept connections on the following locations:

- ▶ Forward traffic to 8001 (Kubernetes API)
- ▶ 8443 (platform UI), 9443 (authentication service)
- ▶ 8500 and 8600 (private registry)

When using an external load balancer, each master node can be in different subnets if the round-trip network time between the master nodes is less than 33 ms for `etcd`. Figure 7-3 on page 263 illustrates the load balancer option.

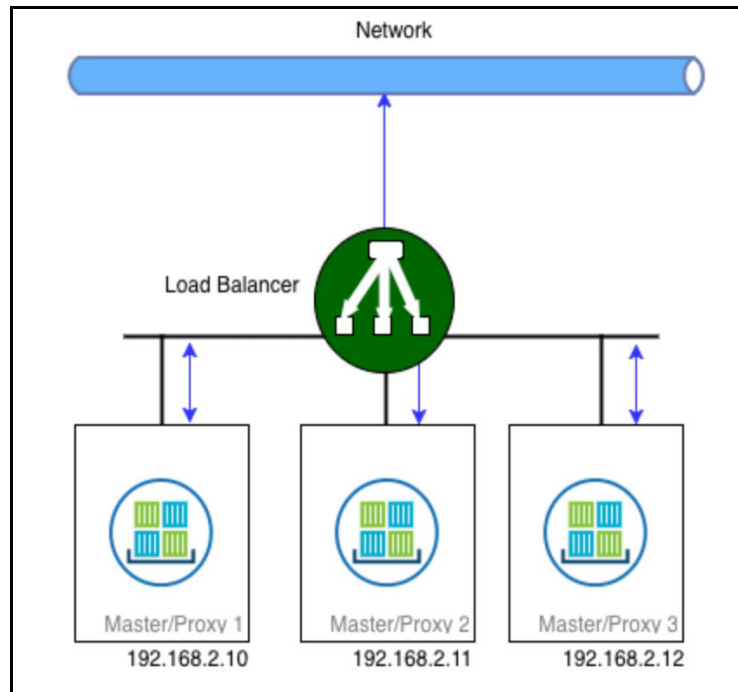


Figure 7-3 Load balancer in an IBM Cloud Private environment

7.3.2 Virtual IP addresses

In case a load balancer is not available, high availability of the master and proxy nodes can be achieved using a virtual IP address, which is in a subnet shared by the master/proxy nodes. IBM Cloud Private supports three types of virtual IP management solutions:

- ▶ etcd (default)
- ▶ ucarp
- ▶ keepalived

This setting is done once as part of the installation of IBM Cloud Private, using the **vip_manager** setting in `config.yaml`. For ucarp and keepalived, the advertisements happen on the management interface, and the virtual IP will be held on the interface provided by `cluster_vip_iface` and `proxy_vip_iface`. *In situations where the virtual IP will be accepting a high load of client traffic, the management network performing the advertisements for master election should be separate from the data network accepting client traffic.*

Note: Considerations when using a virtual IP address are as follows:

- ▶ At any point of time, only one master or proxy node holds the lease for the virtual IP address.
- ▶ Using a virtual IP, traffic is not load balanced among all replicas. Using a virtual IP requires that all candidate nodes use a `cluster_vip_iface` or `proxy_vip_iface` interface on the same subnet.
- ▶ Any long-running and stateful TCP connections from clients will be broken during a failover and must be reestablished.

The etcd solution

Etcd is a distributed key value store used internally by IBM Cloud Private to store state information. It uses a distributed census algorithm called *raft*. The etcd-based VIP manager leverages the distributed key/value store to control which master or proxy node is the instance holding the virtual IP address. The virtual IP address is leased to the leader, so all traffic is routed to that master or proxy node.

The etcd virtual IP manager is implemented as an etcd client that uses a key/value pair. The current master or proxy node holding the virtual IP address acquires a lease to this key/value pair with a TTL of 8 seconds. The other standby master or proxy nodes observe the lease key/value pair.

If the lease expires without being renewed, the standby nodes assume that the first master has failed and attempt to acquire their own lease to the key to be the new master node. The master node that is successful writing the key brings up the virtual IP address. The algorithm uses randomized election timeout to reduce the chance of any racing condition where one or more nodes tries to become the leader of the cluster.

Note: Gratuitous ARP is not used with the etcd virtual IP manager when it fails over. Therefore, any existing client connections to the virtual IP address after it fails over will fail until the client's ARP cache has expired and the MAC address for the new holder of the virtual IP is acquired. However the etcd virtual IP manager avoids the use of multicast as ucarp and keepalived requires.

The ucarp solution

Ucarp is an implementation of the common address redundancy protocol (CARP) ported to Linux. Ucarp allows the master node to “advertise” that it owns a particular IP address using the multicast address 224.0.0.18.

Each node sends out an advertisement message on its network interface that it can have a virtual IP address every few seconds. This is called the *advertise base*. Each master node sends a skew value with that CARP (Common Address Redundancy Protocol) message. This is similar to its priority of holding that IP, which is the *advskew* (advertising skew). Two or more systems both advertising at one second intervals (*advbase*=1), the one with the lower *advskew* will *win*.

Any ties are broken by the node that has the lower IP address. For high availability, moving one address between several nodes in this manner enables you to survive the outage of a host, but you must remember that *this only enables you to be more available and not more scalable*.

A master node will become master if one of the following conditions occurs:

- ▶ No one else advertises for 3 times its own advertisement interval (*advbase*).
- ▶ The `--preempt` option is specified by the user, and it “hears” a master with a longer (advertisement) interval (or the same *advbase* but a higher *advskew*).

An existing master node becomes a backup if on of the following conditions occur:

- ▶ Another master advertises a shorter interval (or the same *advbase*, but a lower *advskew*).
- ▶ Another master advertises the same interval, and has a lower IP address.

After failover, ucarp sends a gratuitous ARP message to all of its neighbors so that they can update their ARP caches with the new master's MAC address.

The keepalived solution

Keepalived provides simple and robust facilities for load balancing and high-availability, originally used for high availability of virtual routers. Keepalived uses VRRP (Virtual Router Redundancy Protocol) as an election protocol to determine which master or proxy node holds the virtual IP. The keepalived virtual IP manager implements a set of checkers to dynamically and adaptively maintain and manage load balanced server pool according to the health.

VRRP is a fundamental brick for failover. The keepalived virtual IP manager implements a set of hooks to the VRRP finite state providing low-level and high-speed protocol interactions.

To ensure stability, the keepalived daemon is split into the following parts:

- ▶ A parent process called as watchdog in charge of the forked children process monitoring.
- ▶ A child process for VRRP.
- ▶ Another child process for health checking.

The keepalived configuration included with IBM Cloud Private uses the multicast address 224.0.0.18 and IP protocol number 112. This must be allowed in the network segment where the master advertisements are made. Keepalived also generates a password for authentication between the master candidates which is the MD5 sum of the virtual IP.

Keepalived by default uses the final octet of the virtual IP address as the virtual router ID (VRID). For example, for a virtual IP address of 192.168.10.50, it uses VRID 50. If there are any other devices using VRRP on the management layer 2 segment that are using this VRID, it might be necessary to change the virtual IP address to avoid conflicts.

7.3.3 Ingress controller

Ingress resources in Kubernetes are used to proxy layer 7 traffic to containers in the cluster. An *ingress* is a collection of rules to allow inbound connections to the Kubernetes cluster services. It can be configured to give Kubernetes services externally reachable URLs, terminate TLS connections, offer name-based virtual hosting, and more.

Note: Ingress controller in an IBM Cloud Private environment is also known as the *proxy node*.

Ingress resources require an ingress controller component to be running as a Layer 7 proxy service inside the cluster. In IBM Cloud Private, an nginx-based ingress controller is provided by default that is deployed on the proxy or master (in case master acts as proxy) nodes. The default ingress controller watches Kubernetes ingress objects on all namespaces through the Kubernetes API and dynamically programs the nginx proxy rules for upstream services based on the ingress resource. By default, the ingress controller is bootstrapped with some load balancing policies, such as load-balancing algorithms and a back-end weight scheme.

More than one ingress controller can also be deployed if isolation between namespaces is required. The ingress controller itself is a container deployment that can be scaled out and is exposed on a host port on the proxy nodes. The ingress controller can proxy all of the pod and service IP mesh running in the cluster.

IBM Cloud Private installation defines some node roles dedicated to running the shared IBM Cloud Private ingress controller called proxy nodes. These nodes serve as a layer 7 reverse proxy for the workload running in the cluster. In situations where an external load balancer can be used, this is the suggested configuration, because it can be difficult to secure and scale proxy nodes, and using a load balancer avoids additional network hops through proxy nodes to the pods running the actual application.

If you are planning to use an external load balancer, set up the cluster to label the master nodes as proxy nodes using the `hosts` file before installation. This marks the master nodes with the additional proxy label, as shown in Example 7-1, and the shared ingress controller will be started on the master nodes. This ingress controller can generally be ignored for “northbound” traffic, or used for lightweight applications exposed “southbound”, such as additional administrative consoles for some applications that are running in the cluster.

Example 7-1 The config.yaml file

```
[master]
172.21.13.110
172.21.13.111
172.21.13.112

[proxy]
172.21.13.110
172.21.13.111
172.21.13.112
```

If an ingress controller and ingress resources are required to aggregate several services that use the built-in ingress resources, a good practice is to install additional isolated ingress controllers using the included Helm chart for the namespace, and expose these individually through the external load balancer.

Single service ingress

It is possible to expose a single service through ingress. In Example 7-2, a Node.js server was created with service name `mynode-ibm-nodejs-sample` on port 3000.

Example 7-2 Ingress controller configuration

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: myingressexample
spec:
  backend:
    serviceName: test-app-using-nodejs
    servicePort: 3000
```

In this case, all traffic on the ingress controller’s address and port (80 or 443) will be forwarded to this service.

Simple fanout

With the simple fanout approach you can define multiple HTTP services at different paths and provide a single proxy that routes to the correct endpoints in the back end. When there is a highly available load balancer managing the traffic, this type of ingress resource will be helpful in reducing the number of load balancers to a minimum.

In the Example 7-3 on page 267, “/” is the rewrite target for two services: `employee-api` on port 4191 and `manager-api` on port 9090. The context root for both of these services is `/`; the ingress will rewrite the path `/hr/employee/*` and `/hr/manager/*` to `/` when proxying the requests to the back ends.

Example 7-3 Example of rewriting

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: api
spec:
  rules:
  - host: icpdemo.mydomain.com
    http:
      paths:
      - backend:
          serviceName: employee-api
          servicePort: 4191
        path: /hr/employee/*
      - backend:
          serviceName: manager-api
          servicePort: 9090
        path: /hr/manager/*
```

Example 7-3 demonstrates that it is possible to expose multiple services and rewrite the URI.

Name-based virtual hosting

Name-based virtual hosting provides the capability to host multiple applications using the same ingress controller address. This kind of ingress routes HTTP requests to different services based on the Host header.

In Example 7-4 and Example 7-5 on page 268, two Node.js servers are deployed. The console for the first service can be accessed using the host name `myserver.mydomain.com` and the second using `superserver.mydomain.com`. In DNS, `myserver.mydomain.com` and `superserver.mydomain.com` can either be an A record for the proxy node virtual IP 10.0.0.1, or a CNAME for the load balancer forwarding traffic to where the ingress controller is listening.

Example 7-4 First service

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    name: myserver
spec:
  rules:
  - host: myserver.mydomain.com
    http:
      paths:
      - backend:
          serviceName: nodejs-myserver
          servicePort: 3000
```

Deploying the second service is shown in Example 7-5.

Example 7-5 Second service

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    name: mysuperserver
spec:
  rules:
  - host: superserver.mydomain.com
    http:
      paths:
      - backend:
          serviceName: nodejs-superserver
          servicePort: 3000
```

It is usually a good practice to provide some value for the host, because the default is *, which forwards all requests to the back end.

Transport Layer Security (TLS)

An ingress service can be secured using a TLS private key and certificate. The TLS private key and certificate should be defined in a secret with key names `tls.key` and `tls.crt`. The ingress assumes TLS termination and traffic is proxied only on port 443. Example 7-6 shows how to create this.

Example 7-6 TLS configuration

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
    ingress.kubernetes.io/rewrite-target: /
  name: api
spec:
  rules:
  - host: myserver.mydomain.com
    http:
      paths:
      - backend:
          serviceName: main-sales-api
          servicePort: 4191
        path: /api/sales/*
      - backend:
          serviceName: backorder-api
          servicePort: 9090
        path: /api/backorder/*
  tls:
  - hosts:
    - myserver.mydomain.com
    secretName: api-tls-secret
```

In Example 7-6, the TLS termination is added to the `myserver.mydomain.com` ingress resource.

The certificate's subject name or subject alternative names (SANs) should match the host value in the ingress resource and should be unexpired. In addition, the full certificate chain (including any intermediate and root certificates) should be trusted by the client. Otherwise, the application gives a security warning during the TLS handshake. In the example, the `tls.crt` subject name contains either `api.example.com` or is a wildcard certificate for `*.mydomain.com`. The DNS entry for `myserver.mydomain.com` is an A record for the proxy nodes' virtual IP address.

The secret `api-tls-secret` is created in the same namespace as the ingress resource using the command:

```
kubectl create secret tls api-tls-secret --key=/path/to/tls.key
--cert=/path/to/tls.crt
```

The secret can also declaratively be created in a YAML file if the TLS key and certificate payloads are base-64 encoded, as shown in Example 7-7.

Example 7-7 TLS configuration

```
apiVersion: v1
type: Opaque
kind: Secret
metadata:
  name: api-tls-secret
data:
  tls.crt: <base64-encoded cert>
  tls.key: <base64-encoded key>
```

Shared ingress controller

By default in IBM Cloud Private, a global ingress controller is installed and deployed on all proxy nodes. This provides the capability to define the ingress resources for applications across all namespaces. The global ingress controller runs in the `kube-system` namespace; if a `NetworkPolicy` is used to isolate namespace traffic, another one needs to be created to allow traffic from the ingress controller to any proxied back-end services in other namespaces.

Advantages:

- ▶ A common ingress controller reduces compute resources required to host applications.
- ▶ Ready for use.

Disadvantages:

- ▶ All client traffic passes through a shared ingress controller. One service's client traffic can affect the other.
- ▶ Limited ability to isolate northbound ingress resource traffic from southbound ingress traffic, such as a public-facing API versus an operations dashboard running in the same cluster would share the same ingress controller.
- ▶ If an attacker were to gain access to the ingress controller they would be able to observe unencrypted traffic for all proxied services.
- ▶ Need to maintain different ingress resource documents for different stages. For example, the need to maintain multiple copies of the same ingress resource YAML file with different namespace fields.
- ▶ The ingress controller needs access to read ingress, service, and pod resources in every namespace in the Kubernetes API to implement the ingress rules.

Isolated ingress controllers per namespace

An ingress controller can be installed as a Helm chart in an isolated namespace and perform ingress for services in the namespace. In this deployment type, the ingress controller is given a role that can only access ingress and resources in the namespace.

Advantages:

- ▶ Delineation of ingress resources for various stages of development, production.
- ▶ Performance for each namespace can be scaled individually.
- ▶ Traffic is isolated; when combined with isolated worker nodes on separate VLANs, true Layer 2 isolation can be achieved as the upstream traffic does not leave the VLAN.
- ▶ Continuous integration and continuous delivery teams can use the same ingress resource document to deploy (assuming that the dev namespace is different from the production namespace) across various stages.

Disadvantages:

- ▶ Additional ingress controllers must be deployed, using extra resources.
- ▶ Ingress controllers in separate namespaces might require either a dedicated node or a dedicated external load balancer.

7.4 Service discovery (kube-dns)

Kubernetes expects that a service should be running within the pod network mesh that performs name resolution and should act as the primary name server within the cluster. In IBM Cloud Private, this is implemented using CoreDNS running on the master nodes, which resolves names for all services running in Kubernetes. In addition, the service forwards name lookups against upstream name servers on behalf of containers. The DNS service itself runs as a ClusterIP service that is backed by one or more containers for high availability. See Figure 7-4.

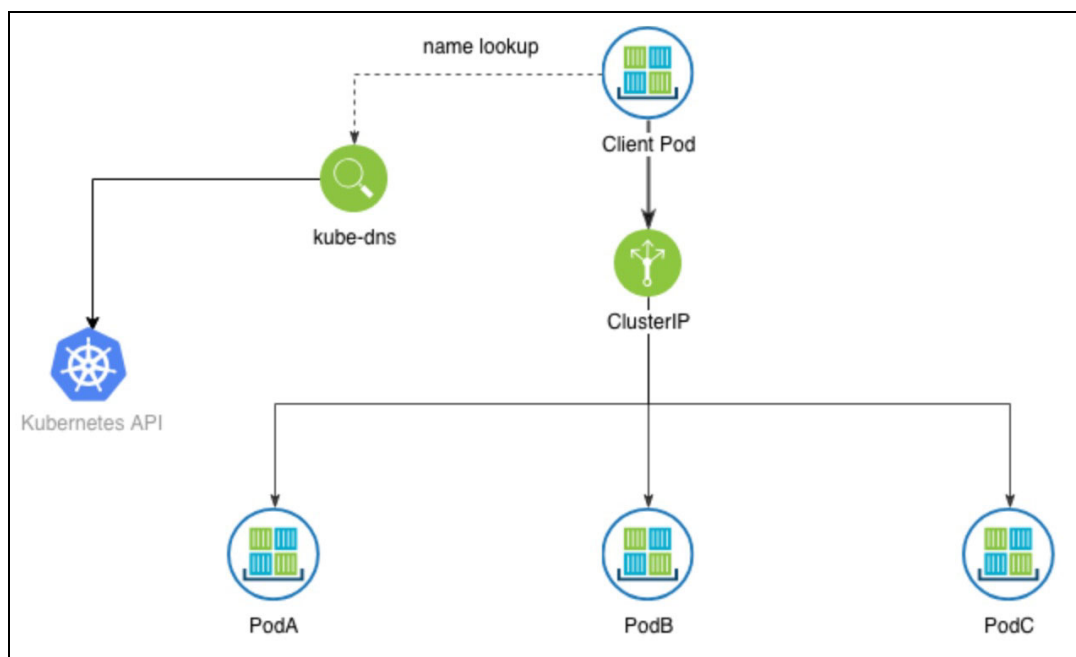


Figure 7-4 Kube-dns

Kubernetes service names are resolved to *ClusterIPs* representing one or more pods matching a label selector. The cluster is assigned a cluster domain that is specified at installation time that uses `cluster_domain` (this is `cluster.local` by default) to distinguish between names local to the cluster and external names.

Each Kubernetes cluster is logically separated into namespaces, and each namespace acts as a subdomain for name resolution. Upon examining a container's `/etc/resolv.conf`, observe that the name server line points at an IP address internal to the cluster, and the search suffixes are generated in a particular order, as shown in Example 7-8.

Example 7-8 The `/etc/resolv.conf`

```
# cat /etc/resolv.conf
Name server <kube-dns ClusterIP>
search <namespace>.svc.<cluster_domain> svc.<cluster_domain> <cluster_domain>
<additional ...>
options ndots:5
```

The `<additional ...>` is a list of search suffixes obtained from the worker node's `/etc/resolv.conf` file. By default, a short host name like `account-service` has `<namespace>.svc.<cluster_domain>` appended to it, so a pod matching the label selector that is running in the same namespace as the running pod will be selected. A pod can look up the ClusterIP of a pod in a different namespace by appending the namespace to the host name.

For example, `account-service.prod` will target `account-service` running in the `prod` namespace, because the search suffix `svc.<cluster_domain>` is appended to the end. Figure 7-5 shows how the naming segmentation works.

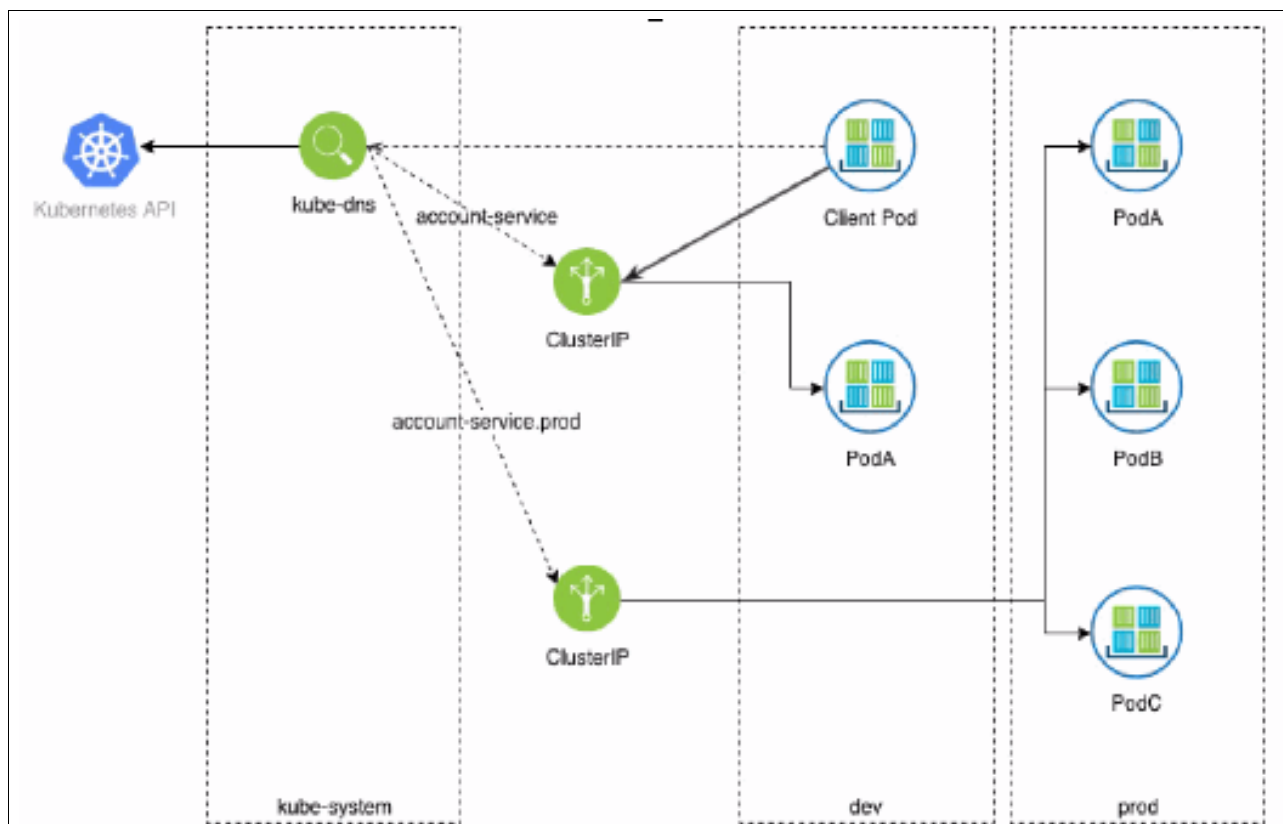


Figure 7-5 KubeDNS with namespace segmentation

Note the last line in `/etc/resolv.conf`, options `ndots:5`, which indicates to the container's system resolver that any host names being resolved that have fewer than five dots in the name should have the search domain suffixes appended to it. This might affect performance because lookups of external network resources with fewer than 5 dots in the name results in lookups for every entry in the search line.

For example, a lookup of `www.ibm.com` results in lookups of `www.ibm.com.<namespace>.svc.<cluster_domain>`, `www.ibm.com.svc.<cluster_domain>`, `www.ibm.com.<cluster_domain>`, and so on before finally trying `www.ibm.com`. To resolve this issue, adding an additional period (".") to the end of the fully qualified domain names used in the application configuration will prevent the system resolver from cycling through the list of suffixes in the name lookups (for example `www.ibm.com.`).

7.4.1 Headless services

In some cases, it is desirable *not* to create a ClusterIP service at all; this can be achieved by specifying a service type of `None`. This will create A records for each pod matching the label selector in the DNS, but no ClusterIP. This is typically used with `StatefulSets` where each of the pods needs to have a resolvable name for communication between all pods in the set (for example, a clustered database, such as MongoDB). When a service without a ClusterIP is created, each pod's A record will be in the following format:

```
<pod-name>.<service-name>.<namespace>.svc.<cluster-domain>
```

7.4.2 External services

It is possible to have Kubernetes proxy endpoints outside of the cluster by creating a Service resource with no label selector, and by either creating `Endpoints` resources manually containing the IPs outside of the cluster to proxy, or creating a Service resource with the type `ExternalName` containing an external DNS name, which creates a CNAME record in the cluster's DNS. By using these functions, the cluster DNS can be leveraged as service discovery for services both inside and outside of the cluster.



Troubleshooting

This chapter provides information about how to fix some common issues with IBM Cloud Private. It shows you how to collect log information and open a request with the IBM Support team.

This chapter has the following sections:

- ▶ 8.1, “Common errors during the IBM Cloud Private installation” on page 274
- ▶ 8.2, “Network configuration errors” on page 277
- ▶ 8.3, “Common errors when installing a Helm chart” on page 281
- ▶ 8.4, “Common errors when running applications” on page 286
- ▶ 8.5, “Opening a support case” on page 287

8.1 Common errors during the IBM Cloud Private installation

This section gives you some tips on how to troubleshoot IBM Cloud Private installation problems.

8.1.1 Customizing the config.yaml file

While installing IBM Cloud Private, you need to customize the config.yaml file located at /<installation_directory>/cluster/config.yaml.

The list of required parameters to be configured on a config.yaml file is available at https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/installing/install_containers.html see *step #3 “Customize your cluster”*.

The first parameter that you need to configure in the config.yaml file is the admin user and password. The admin password policy has been changed in IBM Cloud Private version 3.1.2 and now requires, by default, *at least 32 characters*. If the password does not match the requirements, the installation log will show an error as shown in Example 8-1.

Example 8-1 Password problem

```
TASK [Checking if setting password or not] *****
fatal: [localhost]: FAILED! => changed=false
  msg: 'The password is not set. You must specify a password that meets the
  following criteria: '^([a-zA-Z0-9\~]{32,})$''
NO MORE HOSTS LEFT *****
```

To fix the previous issue, go to /<installation_directory>/cluster/config.yaml and define a password that matches the standard that has at least 32 alphanumeric characters.

If you want to change the policy, you can use the regular expression that best fits your company policy as described at the following link:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/installing/install_containers.html

If your server has more than one Ethernet adapter and you are installing the IBM Cloud Private Enterprise Edition, you also need to configure the following parameters:

- ▶ cluster_lb_address: <external address>
- ▶ proxy_lb_address: <external address>

The external address is the IP address of the adapter from which the incoming traffic will be coming to the server.

For the full options of the cluster.yaml file configuration, see the following URL:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/installing/config_yaml.html

8.1.2 Customizing the /cluster/hosts file

The hosts file in the cluster directory is used to define the cluster architecture and configures how the workers, masters, and management servers are distributed.

When configuring persistence storage, you need to group them in the hosts file. See the full specification of the hosts file configuration at the following link:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/installing/hosts.html

If the hosts file configuration is not done, the error described in Example 8-2 will be displayed.

Example 8-2 The hosts file configuration error

```
fatal: [...]: UNREACHABLE! => changed=false
msg: |-
    Failed to connect to the host via ssh: ssh: Could not resolve hostname ...:
Name does not resolve
unreachable: true
```

Tip: When planning for the installation of IBM Cloud Private, it is highly advised that you define all of the functions that you want your server to run, because some of the customizations on the hosts file require IBM Cloud Private to be uninstalled and installed again.

If you plan to use storage for persistent data, you might need to add the host group in the hosts file for that particular type of storage. Read the documentation about the storage system you are using and make sure that the prerequisites are met before running the IBM Cloud Private installation. For more information, see Chapter 4, “Managing persistence in IBM Cloud Private” on page 115.

8.1.3 SSH key error

During the installation preparation, it is required to generate and exchange the SSH key between the nodes. In addition, you need to copy the SSH key to the IBM Cloud Private cluster installation folder (`/<installation_directory>/cluster`). If this step is not performed, you will receive the error message in Example 8-3 during the installation.

Example 8-3 SSH key error

```
fatal: [9.46.67.246]: UNREACHABLE! => changed=false
msg: |-
    Failed to connect to the host via ssh: Load key "/installer/cluster/ssh_key":
invalid format
    Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).
unreachable: true
```

To correct the error, copy the `~/.ssh/id_rsa` to `/<installation_directory>/cluster/ssh_key`

This procedure is described in “*Step 2: Set up the installation environment item 9*” at the following link:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/installing/install_containers.html

8.1.4 Missing the IBM Cloud Private binary files in the installation folder

To have the IBM Cloud Private installed you need to copy the binary files to the `/<installation_directory>/cluster/images` folder.

If this step is not performed, the error shown in Example 8-4 is displayed during the installation.

Example 8-4 Missing binary files error

```
TASK [icp-registry-image : Aborting installation process] *****
fatal: [9.46.67.246]: FAILED! => changed=false
  msg: Unable to find offline package under images directory
```

This procedure is described in step 2: *Set up the installation environment item 10* at the following link:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/installing/install_containers.html

8.1.5 Missing the minimum system requirements

Missing the minimum system requirements could cause random errors during the installation. If the installation completes, it might present an error during a new Helm chart deployment, or when trying to access an existing chart or running a function in your IBM Cloud Private environment.

To avoid these kinds of errors, it is required that the system matches at least the minimum system requirements. You can see the system requirements at this link:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/supported_system_config/hardware_reqs.html

Also, it is suggested that you need to evaluate the sizing of the cluster before the installation. See this link for information about how to size your cluster:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/installing/plan_capacity.html

8.1.6 Perform the system cleanup when the installation fails

If the installation fails, you need to uninstall IBM Cloud Private with the **uninstall** command before trying a new installation.

Run the **uninstall** command:

```
sudo docker run --net=host -t -e LICENSE=accept -v "$(pwd)":/installer/cluster
ibmcom/icp-inception-amd64:3.1.2-ee uninstall
```

After running the uninstaller, you need to run the commands described in Example 8-5 to make sure that the system is clean and ready for a new installation.

Example 8-5 Making sure the system is ready for a new installation

```
sudo systemctl stop kubelet docker
sudo systemctl start docker
sudo docker rm $(sudo docker ps -qa)
if sudo mount | grep /var/lib/kubelet/pods; then sudo umount $(sudo mount | grep
/var/lib/kubelet/pods | awk '{print $3}'); fi
sudo rm -rf /opt/cni /opt/ibm/cfc /opt/kubernetes
sudo rm -rf /etc/cfc /etc/cni /etc/docker/certs.d
sudo rm -rf /var/lib/etcd/* /var/lib/etcd-wal/*
sudo rm -rf /var/lib/mysql/*
```

```
sudo rm -rf /var/lib/kubelet/* /var/lib/icp/* /var/lib/calico
echo done
sudo rm -rf {{ .icpInstallDir }}/cluster/cfc-certs {{ .icpInstallDir
}}/cluster/cfc-components {{ .icpInstallDir }}/cluster/cfc-keys {{ .icpInstallDir
}}/cluster/.addon {{ .icpInstallDir }}/cluster/.misc ; echo done
```

After completing the previous steps, perform the installation again.

8.2 Network configuration errors

This section describes how to troubleshoot the IBM Cloud Private networking components, such as Calico and IPsec.

8.2.1 Calico troubleshooting

Calico network issues might show up during or after an IBM Cloud Private installation. During the installation, the installer runs checks to ensure seamless pod-to-pod connectivity in the cluster. However, if there are issues, the following information might help to identify the possible causes and resolve the issues.

Problems during the installation

To avoid Calico network issues during the installation, ensure that the following settings are correctly configured.

The **calico_ipip_enabled** parameter must be set to true if all the nodes in the cluster do not belong to the same subnet. This parameter must be set to true if the nodes are deployed in a cloud environment such as OpenStack, where source and destination checks prevent the IP traffic from unknown IP ranges, even if all the nodes belong to the same subnet. This configuration enables encapsulation of pod to pod traffic over the underlying network infrastructure.

The **calico_ip_autodetection_method** parameter must be set, so that Calico uses the correct interface on the node. If there are multiple interfaces, aliases, logical interfaces, bridge interfaces, or any other type of interfaces on the nodes, use either the following settings to ensure that the auto-detect mechanism chooses the correct interface.

- ▶ **calico_ip_autodetection_method**: can-reach= (This is the default setting.)
- ▶ **calico_ip_autodetection_method**: interface=
- ▶ The **calico_tunnel_mtu** parameter must be set based on the MTU of the interface that is configured to be used by Calico.
- ▶ If the **calico_ipip_enabled** parameter is set to true, 20 bytes are used for IP-IP tunnel header. It is required to set the **calico_tunnel_mtu** parameter to be at least 20 bytes less than the actual MTU of the interface.
- ▶ If IPsec is enabled, 40 bytes are needed for the IPsec packet header. Because when you enable IPsec, **calico_ipip_enabled** to true is set, you also need the 20 bytes for the IP-IP tunnel header. Therefore, you must set the **calico_tunnel_mtu** parameter to be at least 60 bytes less than the actual MTU of the interface.
- ▶ The network CIDR (Classless Inter-Domain Routing), existing host network, and the service cluster IP range must not be in conflict with each other.

Problems after the installation of IBM Cloud Private

After the cluster is installed, it could present IP connectivity issues across the pods. Service name resolution issues are a symptom of pods not being able to reach the DNS service. These problems are not always related to Calico networks.

In these situations, gather the following information from the cluster for support:

1. Get the node list:

```
kubectl get nodes -o wide
```

2. Get the logs:

Collect logs from the `calico-node-*` pod running on the node which is experiencing the mesh problem. Example 8-6 shows how to get the logs from `calico-node-*` running on node 10.10.25.71.

Example 8-6 Getting the logs

```
# kubectl get pods -o wide | grep calico-node
```

```
calico-node-amd64-2cbjh 2/2 Running 0 7h 10.10.25.70    10.10.25.70
calico-node-amd64-48lf9 2/2 Running 0 7h 10.10.25.71    10.10.25.71
calico-node-amd64-75667 2/2 Running 0 7h 10.10.25.7     10.10.25.7
```

3. Retrieve the logs from the `calico-node` container in the pod:

```
# kubectl logs calico-node-amd64-48lf9 -c calico-node-amd64
```

4. Get the routing table and interface details to complete this. Run the commands on master node(s) + nodes on which pods are experiencing the connectivity issues.

```
route -n
ifconfig -a
```

5. Get Calico node list running the command on the master node:

```
calicoctl get nodes
```

6. Get all the pods and endpoints on Calico mesh by running the command on the IBM Cloud Private master node:

```
calicoctl get workload endpoints
```

7. Get calico node status and diagnostics. Run the commands on the IBM Cloud Private master node and the nodes on which pods are experiencing connectivity problem:

```
calicoctl node status
calicoctl node diags
```

8. Provide `config.yaml` and host files from boot node.

Configuring calicoctl

Perform the following steps:

1. Log in to the node. Find the `calico-ctl` docker image and copy the `calicoctl` to node as shown in Example 8-7.

Example 8-7 Copy the calicoctl to node

```
# docker images | grep "icp-inception"
```

```
ibmcom-amd64/icp-inception    3.1.0-ee    c816bd4546f9
2 days ago                    746MB
```

```
# docker run -v /usr/local/bin:/data -t --rm -e LICENSE=accept
ibmcom-amd64/icp-inception:3.1.0-ee cp /usr/local/bin/calicoctl /data
# ls /usr/local/bin/calicoctl
/usr/local/bin/calicoctl
```

2. Configure **calicoctl** to authenticate to the etcd cluster. Copy the etcd cert, key, and ca files to a node from boot node's cluster directory:
 - cert file: cluster/cfc-certs/etcd/client.pem
 - key file: cluster/cfc-certs/etcd/client-key.pem
 - ca file: cluster/cfc-certs/etcd/ca.pem
3. Create a calicoctl.cfg file at /etc/calico/calicoctl.cfg, with the following contents, as shown in Example 8-8.

Example 8-8 The calicoctl.cfg file

```
apiVersion: projectcalico.org/v3
kind: CalicoAPIConfig
metadata:
spec:
  datastoreType: "etcdv3"
  etcdEndpoints: "https://<master node IP>:4001"
  etcdKeyFile: <File path of client-key.pem>
  etcdCertFile: <File path of client.pem>
  etcdCACertFile: <file path of ca.pem>
```

4. Change the value between <... > by the actual name.

8.2.2 IPsec troubleshooting

To configure IPsec on IBM Cloud Private, every node in the cluster you must have least two network interfaces. The first one is a management interface. The second interface provides secure networking for the pods. Specify the IP address of the management interface in cluster/hosts and the other interface name (data plane interface) in the Calico and IPsec configurations in cluster/config.yaml.

Calico networks must be enabled in IP-in-IP mode. Calico tunnel MTU must be set correctly.

The IPsec package used for encryption must be installed on all the nodes in the cluster. The IPsec package used for RHEL is libreswan. On Ubuntu and SLES, it is strongswan.

Note: All nodes in the cluster must run the same operating system.

Configuration

When performing the configuration to use IPsec, ensure that the following Calico configurations are provided in the config.yaml file (Example 8-9).

Example 8-9 The config.yaml file

```
network_type: calico
calico_ipip_enabled: true
calico_tunnel_mtu: 1390
calico_ip_autodetection_method: interface=eth0
```

- ▶ `calico_ipip_enabled` must be true. IPIP tunnelling must be enabled for IPsec.
- ▶ `calico_tunnel_mtu` must be at least 60 bytes less than the interface MTU. If the `eth0` interface mtu is 1450 bytes, the `calico_tunnel_mtu` must be set to at most 1390 bytes.
- ▶ `calico_ip_autodetection_method` must be configured to choose the data plane interface.

Example 8-10 Check the IPsec configuration

```
ipsec_mesh:
  enable: true
  interface: eth0
  subnets: [10.24.10.0/24]
  exclude_ips: [10.24.10.1/32, 10.24.10.2, 10.24.10.192/28]
  cipher_suite: aes128gcm16!
```

- ▶ interface must be the same interface that was set in the `calico_ip_autodetection_method` parameter.
- ▶ subnets are the address ranges. The packets destined for such subnet ranges are encrypted. The IP address of the data plane interface must fall in one of the provided subnet ranges.
- ▶ `exclude_ips` are the IP addresses that are excluded from the IPsec subnet. Traffic to these IP addresses is not encrypted.
- ▶ `cipher_suite`: `aes128gcm16!` is the list of Encapsulating Security Payload (ESP) encryption/authentication algorithms to be used. The default cipher suite that is used is `aes128gcm16!`. Ensure that this module is available and loaded in the operating system on all the hosts. It is also possible to change it to another cipher suite.

For the RHEL installation, perform the following steps:

1. Check the `libreswan` configuration:

```
cat /etc/ipsec.conf  
cat /etc/ipsec.d/ipsec-libreswan.conf
```
2. Check the status of the `ipsec` process:

```
ipsec status
```
3. If the `ipsec status` does not display the established connections, check `/var/log/messages` for errors related to IPsec. Enable the `libreswan` logging by enabling `plutodebug` in the `/etc/ipsec.conf` file, as shown in Example 8-11.

```
# /etc/ipsec.conf - libreswan IPsec configuration file

config setup
    ...
    ...
    plutodebug = all          # <<<<<<<<<<<<<
```


To access the pod description, you can check the pod status and pod information:

```
kubectl describe pods <pod> -n <namespace>
```

See the sample output in Example 8-13.

Example 8-13 Pod description

```
Name:          mydatapower-ibm-datapower-dev-d95f656dd-rjk5x
Namespace:     default
Priority:       0
PriorityClassName: <none>
Node:          <none>
Labels:        app=mydatapower-ibm-datapower-dev
               chart=ibm-datapower-dev-2.0.4
               heritage=Tiller
               pod-template-hash=d95f656dd
               release=mydatapower
Annotations:   kubernetes.io/psp: ibm-privileged-psp
               productID:
IBMDatapowerGatewayVirtualEdition_2018.4.1.2.306098_Developers
               productName: IBM DataPower Gateway Virtual Edition for
Developers
               productVersion: 2018.4.1.2.306098
               prometheus.io/module: dpStatusMIB
               prometheus.io/path: /snmp
               prometheus.io/port: 63512
               prometheus.io/scrape: true
               prometheus.io/target: 127.0.0.1:1161
Status:        Pending
IP:
Controlled By: ReplicaSet/mydatapower-ibm-datapower-dev-d95f656dd
Containers:
  ibm-datapower-dev:
    Image:      ibmcom/datapower:2018.4.1.2.306098
    Port:       8443/TCP
    Host Port:  0/TCP
    Command:
      sh
      -c
      exec /start.sh --log-format json-icp

Limits:
  cpu:    8
  memory: 64Gi
Requests:
  cpu:    4
  memory: 8Gi
Liveness: http-get http://:service/ delay=120s timeout=5s period=10s
#success=1 #failure=3
Readiness: http-get http://:service/ delay=120s timeout=5s period=10s
#success=1 #failure=3
Environment:
  DATAPOWER_ACCEPT_LICENSE: true
  DATAPOWER_INTERACTIVE:    true
  DATAPOWER_LOG_COLOR:      false
  DATAPOWER_WORKER_THREADS: 4
```



```

Mounts:
  /drouter/config from mydatapower-ibm-datapower-dev-config-volume (rw)
  /var/run/secrets/kubernetes.io/serviceaccount from default-token-j82nq (ro)
Conditions:
  Type           Status
  PodScheduled   False
Volumes:
  mydatapower-ibm-datapower-dev-config-volume:
    Type:          ConfigMap (a volume populated by a ConfigMap)
    Name:          mydatapower-ibm-datapower-dev-config
    Optional:      false
  default-token-j82nq:
    Type:          Secret (a volume populated by a Secret)
    SecretName:    default-token-j82nq
    Optional:      false
QoS Class:       Burstable
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/memory-pressure:NoSchedule
                  node.kubernetes.io/not-ready:NoExecute for 300s
                  node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type           Reason             Age             From             Message
  ----           -
  Warning        FailedScheduling    52s (x2 over 52s)  default-scheduler  0/1 nodes are
available: 1 Insufficient cpu, 1 Insufficient memory.

```

In Example 8-13 on page 282, it is possible that the pod was not able to run due to insufficient CPU or memory. This would cause the error. To fix this issue, make sure that the server (worker) has sufficient memory and CPU to run the pod.

8.3.2 No CPU available

When looking at the pod description sometimes the message that there is not enough CPU to run the pod is displayed.

To fix the issue, add more CPU and restart the docker and Kubernetes to get the pod running.

To determine the amount of CPU being used, run the following command:

```
kubectl describe node <node>
```

The output of the command will be like that shown in Example 8-14.

Example 8-14 Check the amount of CPU being used

```

Name:          9.46.73.206
Roles:         etcd,management,master,proxy,worker
Labels:        beta.kubernetes.io/arch=amd64
               beta.kubernetes.io/os=linux
               etcd=true
               kubernetes.io/hostname=9.46.73.206
               management=true
               master=true
               node-role.kubernetes.io/etcd=true
               node-role.kubernetes.io/management=true
               node-role.kubernetes.io/master=true

```

```

node-role.kubernetes.io/proxy=true
node-role.kubernetes.io/worker=true
proxy=true
role=master
Annotations: node.alpha.kubernetes.io/ttl: 0
              volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp: Wed, 20 Feb 2019 14:58:20 -0800
Taints: <none>
Unschedulable: false
Conditions:
  Type           Status  LastHeartbeatTime           LastTransitionTime
  Reason                                     Message
  ----
  OutOfDisk      False   Mon, 25 Feb 2019 14:49:08 -0800   Wed, 20 Feb 2019
14:58:20 -0800   KubeletHasSufficientDisk       kubelet has sufficient disk space
available
  MemoryPressure False   Mon, 25 Feb 2019 14:49:08 -0800   Wed, 20 Feb 2019
14:58:20 -0800   KubeletHasSufficientMemory     kubelet has sufficient memory
available
  DiskPressure   False   Mon, 25 Feb 2019 14:49:08 -0800   Wed, 20 Feb 2019
14:58:20 -0800   KubeletHasNoDiskPressure       kubelet has no disk pressure
  PIDPressure    False   Mon, 25 Feb 2019 14:49:08 -0800   Wed, 20 Feb 2019
14:58:20 -0800   KubeletHasSufficientPID        kubelet has sufficient PID available
  Ready          True    Mon, 25 Feb 2019 14:49:08 -0800   Wed, 20 Feb 2019
15:49:34 -0800   KubeletReady                   kubelet is posting ready status
Addresses:
  InternalIP: 9.46.73.206
  Hostname:   9.46.73.206
Capacity:
  cpu:                8
  ephemeral-storage: 244194820Ki
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:              16265924Ki
  pods:               80
Allocatable:
  cpu:                7600m
  ephemeral-storage: 241995268Ki
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:              15114948Ki
  pods:               80
System Info:
  Machine ID:          cbb00030e5204543a0474ffff17ec26f
  System UUID:         79E65241-2145-4307-995A-B3A5C6401F48
  Boot ID:             c8e1b505-e5cf-4da4-ab04-63eb5ad2d360
  Kernel Version:      3.10.0-957.el7.x86_64
  OS Image:            Red Hat Enterprise Linux Server 7.6 (Maipo)
  Operating System:    linux
  Architecture:        amd64
  Container Runtime Version: docker://18.3.1
  Kubelet Version:      v1.12.4+icp-ee
  Kube-Proxy Version:   v1.12.4+icp-ee
  Non-terminated Pods:  (58 in total)

```

Namespace	Name	
CPU Requests	CPU Limits	Memory Requests Memory Limits
-----	-----	-----
cert-manager	ibm-cert-manager-cert-manager-7dbc9c8db6-5d84q	
0 (0%)	0 (0%)	0 (0%) 0 (0%)
kube-system	audit-logging-fluentd-ds-dzswg	
0 (0%)	0 (0%)	0 (0%) 0 (0%)
kube-system	auth-apikeys-sc4k8	
200m (2%)	1 (13%)	300Mi (2%) 1Gi (6%)
kube-system	auth-idp-j457s	
300m (3%)	3200m (42%)	768Mi (5%) 3584Mi (24%)
kube-system	auth-pap-thf7x kube-system	
unified-router-n5v2f		20m (0%) 0
(0%)	64Mi (0%)	0 (0%)
kube-system	web-terminal-6488cfff5d-mgzgw	
10m (0%)	100m (1%)	64Mi (0%) 512Mi (3%)
Allocated resources:		
(Total limits may be over 100 percent, i.e., overcommitted.)		
Resource	Requests	Limits
-----	-----	-----
cpu	6049m (79%)	10506m (138%)
memory	19087040Ki (126%)	23525056Ki (155%)
Events:	<none>	

To solve this issue, you need to add more CPUs to the instance or remove some of the unused pods.

Attention: Be careful if removing a pod on a kube-system, because this action could impact the whole system.

After adding more CPUs, check the node description again.

8.3.3 The required port is in use

When deploying a Helm chart, you might see the message shown in Example 8-15 on the pod description.

Example 8-15 Required port is in use

Type	Reason	Age	From	Message
----	-----	----	----	-----
Warning	FailedScheduling	32s (x2 over 32s)	default-scheduler	0/1 nodes are available: 1 node(s) didn't have free ports for the requested pod ports.

In this case, you can check the full list of ports that have conflicts with the following command:

```
kubectl describe pods <pod>
```

To fix the problem, remove the deployment and change the port that is being used so that there are no conflicts.

8.3.4 Deployment fails due to a missing permission

When deploying a pod with a missing permission, the error message described in Figure 8-2 is displayed.

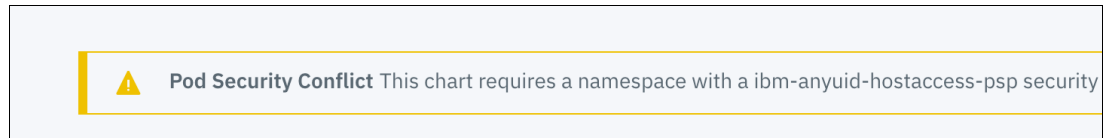


Figure 8-2 Pod security missing

To fix the problem, you need to grant the pod security to the namespace. Run the command:

```
kubectl -n appsales create rolebinding ibm-anyuid-clusterrole-rolebinding  
--clusterrole=ibm-anyuid-clusterrole --group=system:serviceaccounts:appsales
```

After the command completion, try to deploy the pod again.

See the following URL on this troubleshooting:

[https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/user_management/psp_ad
dbind_ns.html](https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/user_management/psp_ad
dbind_ns.html)

8.4 Common errors when running applications

The following sections describe some of the common errors when running applications on IBM Cloud Private, and their solutions.

8.4.1 Getting the 504 or 500 errors when trying to access the application

After deploying a pod or during the execution of the pod, you might get the error message 504 or 500 when trying to access the application from a browser as shown in Figure 8-3.

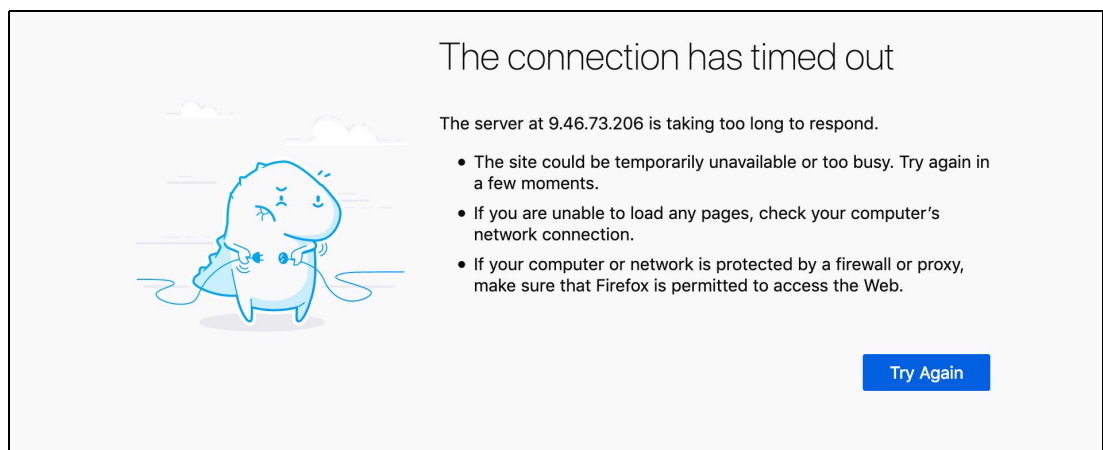


Figure 8-3 Connection time out error

There are some common cases where this error is displayed, such as the pod entering in CrashLoopBack or not starting. Those errors discussed in the next sections.

Pod in CrashLoopBack

When a pod enters in a CrashLoopBack, it means that the pod is trying to start, crashes, and then tries to restart again. At the server console, run the **kubect1 get pods --all-namespaces** command to observe the output, as shown in Example 8-16.

Example 8-16 The kubect1 get pods --all-namespaces command

kubernetes get pods --all-namespaces					
NAMESPACE	NAME	READY	STATUS	RESTARTS	
AGE					
[..]					
my-server-demo	myapp-web-55-84kkm	0/1	CrashLoopBackOff	3774	9h

in Example 8-16, you can see that the pod has been restarted 3774 times in the last 9 hours. Usually this error happens when the pod is starting and failing in a loop.

To try to understand where the error is, you can run the following commands:

```
kubect1 logs <pod>
kubect1 describe <pod>
```

With the output of both commands you can determine where the error is and how to solve it.

Pod not starting

When there is an issue with starting the pod, you can run the **kubect1 get pods -n <namespace>** command. If you see the status of the pod as ErrImagePull or ImagePullBackOff, this means that there is a problem with the deployment. Possible problems include pointing to an image that does not exist, having an incorrect image tag, or not giving Kubernetes permission to run the image.

The details about the error are observed when running the description of the pod (the **kubect1 describe <pod>** command).

8.5 Opening a support case

When you try the troubleshooting methods that are discussed in this chapter and they do not fix the issue, you can open a request to IBM support team. *You need to be an IBM Customer with a valid product ID and license for this.*

Note: if you are using the IBM Cloud Private Community Edition it is possible to get support through the Slack channel and community forum or ask the Watson Chatbot. See the following addresses:

The slack channel is at <https://slack-invite-ibm-cloud-tech.mybluemix.net>.

The stack overflow is at <https://stackoverflow.com/search?q=ibm-cloud-private>.

IBM Watson Chatbot is at <https://ibm.biz/icpsupport>.

Follow the directions at <https://ibm.biz/icpsupport> for opening a support ticket and sending the related data to IBM Support:

When opening the ticket, it is suggested that you include the following information:

- ▶ Title: High level description of the issue
- ▶ Product version
- ▶ Platform (architecture): Specify whether it is a PowerPC®, x86, or other
- ▶ Operating system (OS)
- ▶ Virtualization platform: Where it is installed (VMWARE, Azure, other)
- ▶ High availability (HA) environment or not
- ▶ Problem area
- ▶ Severity
- ▶ Detailed error description
- ▶ Business impact: Is this a PoC, development environment, or production cluster?

Collect the following general troubleshooting data, along with any other data for your problem area:

- ▶ hosts file (located at /<installation_directory>/cluster): This provides IBM the server topology details
- ▶ config.yaml file (located at /<installation_directory>/cluster): This provides details about customization, including the load balancer details.
- ▶ Run the following command and attach the output to the case:

```
sudo docker run --net=host -t -e LICENSE=accept -v "$(pwd)":/installer/cluster  
ibmcom/icp-inception-<architecture>:<version> healthcheck -v
```

Tip: You can find the cheat sheet in “Cheat sheet for production environment” on page 361 useful when troubleshooting the IBM Cloud Private problems.



Service mesh implementation using Istio

This chapter discusses the service mesh implementation using Istio and has the following sections:

- ▶ 9.1, “Overview” on page 290
- ▶ 9.2, “Role of the service mesh” on page 290
- ▶ 9.3, “Istio architecture” on page 292
- ▶ 9.4, “Installation of Istio and enabling the application for Istio” on page 295
- ▶ 9.5, “Service resiliency” on page 301
- ▶ 9.6, “Achieving E2E security for microservices using Istio” on page 311

9.1 Overview

When talking to many enterprise customers who are at the beginning of their application modernization journey, the question that often pops up is: “What about a service mesh?”. This usually prompts many follow-up questions:

- ▶ How should we implement traffic control (firewall rules) for services?
- ▶ How to control ingress and egress from the Kubernetes cluster?
- ▶ What service registry to use?
- ▶ How to implement load balancing between the microservices?
- ▶ How to integrate the microservices on Kubernetes with existing ESB?
- ▶ Can the whole traffic be encrypted?

There are a lot of articles on the internet about different service registry and discovery solutions, and about microservices and cloud native programming frameworks. Many of them relate to a specific product or technology, but it is not obvious how these solutions relate to Kubernetes or to what extent they are applicable in hybrid, multi-cloud, enterprise-centric environments. Things that work fine for an enthusiastic cloud startup developer might not be easy for an enterprise developer who often operates in a highly regulated environment.

In the following section we briefly summarize the role of the service mesh and discuss which functions are fulfilled by Kubernetes itself and which ones need to be augmented by additional solution – in case of IBM Cloud Private the product of choice is *Istio*.

9.2 Role of the service mesh

In traditional applications, the communication pattern was usually built into application code and service endpoint configuration was usually static. This approach does not work in dynamic cloud-native environments. Therefore, there is a requirement for an additional infrastructure layer that helps manage communication between complex applications consisting of a large number of distinct services. Such a layer is usually called a *service mesh*.

The service mesh provides several important capabilities like:

- ▶ Service discovery
- ▶ Load balancing
- ▶ Encryption
- ▶ Observability and traceability
- ▶ Authentication and authorization
- ▶ Support for the circuit breaker pattern

The following sections describe each of these functionalities in more detail.

9.2.1 Service registry

Modern, cloud-native applications are often highly distributed and use a large number of components or microservices that are loosely coupled. In a dynamic, cloud environments placement of any service instance can change at any time, so there is a need for an information repository which will hold current information about which services are available and how they can be reached. This repository is called *service registry*. There are several popular implementations of service registry, for example Eureka developed by Netflix, Consul from Hashicorp, and Apache Zookeeper. None of these solutions are not Kubernetes specific, and their functions to some extent overlap with what Kubernetes natively provides.

IBM Cloud Private uses Kubernetes technology as its foundation. In Kubernetes, services are of primary importance, and Kubernetes provides implicit service registry using DNS. When a controller alters Kubernetes resources, for example starting or stopping some pods, it also updates the related service entries. Binding between pod instances and services that expose them is dynamic, using label selectors.

Istio leverages Kubernetes service resources, and does not implement a separate service registry.

9.2.2 Service discovery

The process of locating the service instance is called *service discovery*. There are 2 types of service discoveries:

Client-side discovery Application that requests a service network location gets all service instances from the service registry, and decides which one to contact. This approach is implemented, for example, by the Netflix Ribbon library.

Server-side discovery Application that sends a request to a proxy that routes a request to one of the available instances. This approach is used in that Kubernetes environment and in Istio.

9.2.3 Load balancing

When there are multiple instances of a target service available, the incoming traffic should be load balanced between them. Kubernetes natively implements this functionality, but Istio greatly enhances the available configuration options.

9.2.4 Traffic encryption

In Kubernetes internal data traffic can be either be all plain or all encrypted using IPsec. Istio allows dynamically to encrypt traffic to and from specific services based on policies and it does not require any changes to the application code.

9.2.5 Observability and traceability

This capability is not implemented in standard Kubernetes, and can be provided by the CNI network implementation. However project Calico used by IBM Cloud Private does not provide this capability. To trace traffic between applications, the applications must embed some distributed tracing libraries, such as Zipkin. Istio implements this capability, allowing all traffic to be traced and visualized, without any modification to the application code.

9.2.6 Access control

Kubernetes, by default, defines network policies that govern which pods can communicate but implementation of network policies is done at the Container Network Interface (CNI) network level. Project Calico used by IBM Cloud Private allows for defining firewall rules between pods based on IP and port combinations. Istio enhances access control up to L7 (layer 7).

9.2.7 Circuit breaker pattern support

By default, Kubernetes does not provide this capability. It can be embedded in application code using, for example, Hystrix from Netflix OSS, but can be also implemented at the proxy level like in Istio or Consul.

9.3 Istio architecture

An Istio service mesh is logically split into a data plane and a control plane¹.

The data plane is composed of a set of intelligent proxies (Envoy) deployed as sidecars. These proxies mediate and control all network communication between microservices along with Mixer, a general-purpose policy and telemetry hub.

The control plane manages and configures the proxies to route traffic. Additionally, the control plane configures Mixers to enforce policies and collect telemetry.

9.3.1 Components

The following section describes the components of the planes.

Figure 9-1 on page 293 shows the different components that make up each plane.

¹ Excerpt from <https://istio.io/docs/concepts/what-is-istio/>

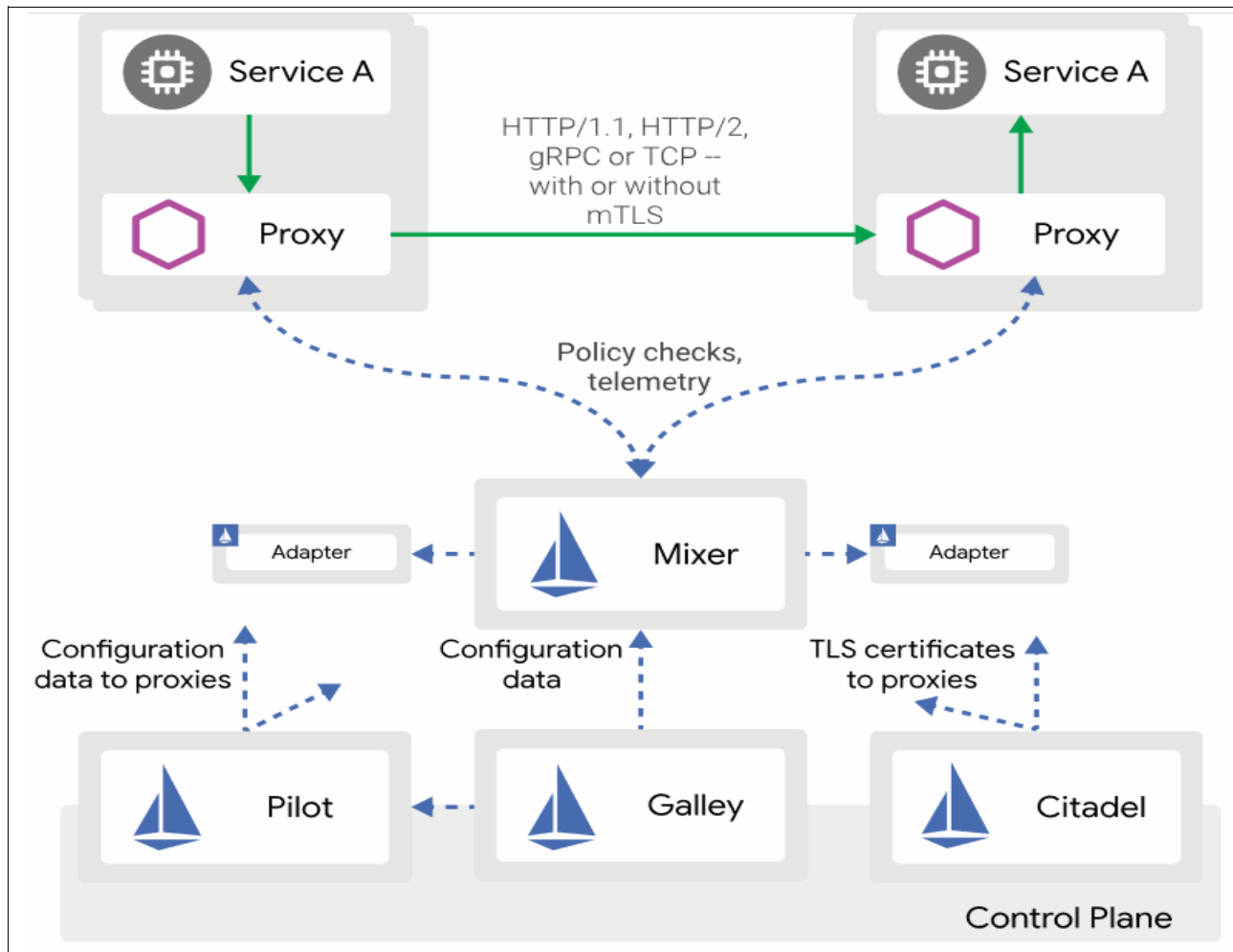


Figure 9-1 Istio architecture²

Envoy

Istio uses an extended version of the Envoy proxy. Envoy is a high-performance proxy developed in C++ to mediate all inbound and outbound traffic for all services in the service mesh. Istio leverages Envoy's many built-in features, for example:

- ▶ Dynamic service discovery
- ▶ Load balancing
- ▶ TLS termination
- ▶ HTTP/2 and gRPC proxies
- ▶ Circuit breakers
- ▶ Health checks
- ▶ Staged rollouts with percentage-based traffic split
- ▶ Fault injection
- ▶ Rich metrics

² Image taken from Istio documentation at <https://istio.io/docs/concepts/what-is-istio/>

Envoy is deployed as a sidecar to the relevant service in the same Kubernetes pod. This deployment allows Istio to extract a wealth of signals about traffic behavior as attributes. Istio can, in turn, use these attributes in Mixer to enforce policy decisions, and send them to monitoring systems to provide information about the behavior of the entire mesh.

The sidecar proxy model also allows you to add Istio capabilities to an existing deployment with no need to rearchitect or rewrite code. You can read more about why we chose this approach in our Design Goals.

Mixer

Mixer is a platform-independent component. Mixer enforces access control and usage policies across the service mesh, and collects telemetry data from the Envoy proxy and other services. The proxy extracts request level attributes, and sends them to Mixer for evaluation. You can find more information about this attribute extraction and policy evaluation in Mixer Configuration documentation.

Mixer includes a flexible plug-in model. This model enables Istio to interface with a variety of host environments and infrastructure backends. Thus, Istio abstracts the Envoy proxy and Istio-managed services from these details.

Pilot

Pilot provides service discovery for the Envoy sidecars, traffic management capabilities for intelligent routing (e.g., A/B tests, canary deployments, etc.), and resiliency (timeouts, retries, circuit breakers, etc.).

Pilot converts high level routing rules that control traffic behavior into Envoy-specific configurations, and propagates them to the sidecars at run time. Pilot abstracts platform-specific service discovery mechanisms and synthesizes them into a standard format that any sidecar conforming with the Envoy data plane APIs can use. With this loose coupling, Istio can run on multiple environments, such as Kubernetes, Consul, or Nomad, while maintaining the same operator interface for traffic management.

Citadel

Citadel provides strong service-to-service and end-user authentication with built-in identity and credential management. You can use Citadel to upgrade unencrypted traffic in the service mesh. Using Citadel, operators can enforce policies based on service identity rather than on network controls. Starting from release 0.5, you can use Istio's authorization feature to control who can access your services.

Galley

Galley validates user authored Istio API configuration on behalf of the other Istio control plane components. Over time, Galley will take over responsibility as the top-level configuration ingestion, processing and distribution component of Istio. It will be responsible for insulating the rest of the Istio components from the details of obtaining user configuration from the underlying platform (for example Kubernetes).

9.3.2 Istio functions

Istio has the following major functions.

Connect

Istio provides traffic management for services. The traffic management function includes:

- ▶ **Intelligent routing:** The ability to perform traffic splitting and traffic steering over multiple versions of the service
- ▶ **Resiliency:** The capability to increase micro services application performance and fault tolerance by performing resiliency tests, error and fault isolation and failed service ejection.

Secure

Istio implements a Role-based Access Control (RBAC) which allow a specific determination on which service can connect to which other services. Istio uses Secure Production Identity Framework for Everyone (SPIFFE) to identify the ServiceAccount of a micro service uniquely and use that to make sure communication are allowed.

Control

Istio provides a set of Policies that allows control to be enforced based on data collected. This capability is performed by Mixer.

Observe

While enforcing the policy, Istio also collects data that is generated by the Envoy proxies. The data can be collected as metrics into Prometheus or as tracing data that can be viewed through Jaeger and Kiali.

This chapter describes mainly the resiliency (9.5, “Service resiliency” on page 301) and security (9.6, “Achieving E2E security for microservices using Istio” on page 311) implementation using Istio, while intelligent routing is discussed in “*Chapter 4 Manage your service mesh with Istio*” of the IBM Redbooks *IBM Cloud Private Application Developer's Guide*, SG24-8441.

9.4 Installation of Istio and enabling the application for Istio

Istio can be enabled during IBM Cloud Private installation time, or it can be installed after the cluster is set up. The IBM Knowledge Center for Cloud Private provides good guidance. You can see

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/manage_cluster/istio.html for more details.

Here we will introduce another approach by installing the Istio using the command-line interface, where you have more control on the options.

Tip: This approach can be used as a base in the airgap environment where you must use a local chart.

In addition, we will also demonstrate the sample bookInfo application deployed in IBM Cloud Private, where you must pay extra attention due to the enhanced security management features starting from IBM Cloud Private version 3.1.1.

9.4.1 Install Istio with the helm command

Before the installation, we assume that you have downloaded the **cloudctl**, **kubectl**, and **helm** tools for your platform from the IBM Cloud Private dashboard.

1. Log in to IBM Cloud Private:

Run the command **cloudctl -a https://Your_ICP_Host:8443/**. Log in with the admin ID and password. Upon successful login, the tool sets up the **kubectl** and the Helm client.

2. Next you will set up the Helm repository. Run the command in Example 9-1 to add a Helm repository that points to the management repository from IBM Cloud Private.

Example 9-1 Command to add a Helm repository

```
helm repo add --ca-file ~/.helm/ca.pem --key-file ~/.helm/key.pem --cert-file  
~/.helm/cert.pem icp https://Your_ICP_Host:8443/mgmt-repo/charts
```

Note that the ca-file, the key-file, and the cert-file are created automatically when you perform the first step.

Refresh the repository with the **helm repo update** command.

3. Create the secret for Grafana and Kiali. We enable Grafana and Kiali for this deployment. The secret for the console login is required. Create the files as in Example 9-2 and in Example 9-3 on page 296.

Example 9-2 Secret object for Kiali

```
apiVersion: v1  
kind: Secretometadata:  
  name: kiali  
  namespace: istio-system  
  labels:  
    app: kiali  
type: Opaque  
data:  
  username: YWRtaW4K  
  passphrase: YWRtaW4K
```

Example 9-3 Secret object for Grafana

```
apiVersion: v1  
kind: Secret  
metadata:  
  name: grafana  
  namespace: istio-system  
  labels:  
    app: grafana  
type: Opaque  
data:  
  username: YWRtaW4K  
  passphrase: YWRtaW4K
```

Notice that the username and passphrase are base64 encoded. You can get the encoded value by running **echo yourtext | base64**.

4. Run the commands in Example 9-4 to apply the objects.

Example 9-4 Commands to apply the secret objects

```
kubectl apply -f grafana.yaml
kubectl apply -f kiali.yaml
```

5. Now you need to customize your settings. Create a YAML file as shown in Example 9-5 to override the default settings of the istio Helm chart. Save it as vars.yaml.

You can see the values.yaml file in the chart. The default chart tarball can be downloaded with the following command:

```
curl -k -LO https://<Your
Cluster>:8443/mgmt-repo/requiredAssets/ibm-istio-1.0.5.tgz
```

Example 9-5 Override the default settings of the Istio Helm chart

```
grafana:
  enabled: true
tracing:
  enabled: true
kiali:
  enabled: true
```

Here we enable the grafana, tracing, and kiali, which are disabled by default.

6. Next you will deploy the Istio chart by running the command in Example 9-6 on page 297.

Example 9-6 Deploy the Istio Helm chart

```
helm.icp install --name istio --namespace istio-system -f vars.yaml icp/ibm-istio
--tls
```

Note that the CustomResourceDefinitions (CRD) no longer needs to be created separately before the deployment.

7. To validate the installation, run the command shown in Example 9-7.

Example 9-7 Validation

```
kubectl -n istio-system get pods -o wide
```

NAME		READY	STATUS	RESTARTS	AGE	IP
grafana-cbc8c66bb-bqd1l			1/1	Running	0	103m
172.20.72.180	10.93.221.105	<none>				
istio-citadel-7cc85b9986-vk7nn			1/1	Running	0	103m
172.20.72.138	10.93.221.105	<none>				
istio-egressgateway-79895bb8f7-k6zfw			1/1	Running	0	103m
172.20.121.208	10.93.221.68	<none>				
istio-galley-77554979fc-j2qcg			1/1	Running	0	103m
172.20.72.189	10.93.221.105	<none>				
istio-ingressgateway-56758bf968-4gfmt			1/1	Running	0	103m
172.20.61.77	10.171.37.135	<none>				
istio-ingressgateway-56758bf968-zjzjz			1/1	Running	0	65m
172.20.121.209	10.93.221.68	<none>				
istio-pilot-599f699d55-479ct			2/2	Running	0	103m
172.20.72.185	10.93.221.105	<none>				
istio-policy-f8fcb8496-smck			2/2	Running	0	103m
172.20.72.184	10.93.221.105	<none>				

istio-sidecar-injector-864d889459-zz1q2	1/1	Running	0	103m
172.20.72.190 10.93.221.105 <none>				
istio-statsd-prom-bridge-75cc7c6c45-xq72c	1/1	Running	0	103m
172.20.72.131 10.93.221.105 <none>				
istio-telemetry-665689b445-vfvqb	2/2	Running	0	103m
172.20.72.150 10.93.221.105 <none>				
istio-tracing-694d9bf7b4-8tlhs	1/1	Running	0	103m
172.20.72.186 10.93.221.105 <none>				
kiali-749cfd5f6-5kgjw	1/1	Running	0	103m
172.20.72.183 10.93.221.105 <none>				
prometheus-77c5cc6dbd-h8bxv	1/1	Running	0	103m
172.20.72.187 10.93.221.105 <none>				

All the pods under the namespace istio-system are running. Notice that other than the ingressgateway and egressgateway that run on the proxy node, the rest of the services all run on the management node.

9.4.2 Enable application for Istio

With the release of IBM Cloud Private Version 3.1.1, many security enhancements are turned on by default. This section documents what extra configurations are required in order for your application to be managed by Istio.

The bookinfo application is a sample application that was developed by istio.io to demonstrate the various Istio features. We will deploy the application into a dedicated namespace, istio-exp, instead of the default namespace, which you will more likely face in a real project.

1. Create the namespace as follows:
 - a. You can create the namespace with the Dashboard console. Go to **Menu —Manage → Namespaces**, then click **Create Namespace**. This action displays a dialog, as shown in Figure 9-2 on page 299.

Figure 9-2 Create a namespace

- b. Name the namespace `istio-exp`, then select **ibm-privileged-psp** as the Pod Security Policy.
- c. Click **Create**.

Note: You can also create these using the `kubectl create ns istio-exp` `kubectl` command.

- d. Create the file named as `rolebinding.yaml` with the content shown in Example 9-8.

Example 9-8 Rolebinding to bind service account to predefined cluster role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: ibm-privileged-clusterrole-rolebinding
  namespace: istio-exp
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ibm-privileged-clusterrole
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts:istio-exp
```

- e. Then run the command `kubectl apply -f rolebinding.yaml`.

When Istio injects the sidecar into the pod through initContainer, it needs the privileged right. Therefore, we assign the `ibm-privileged-clusterrole` to the service account of the namespace. If not, you might see the error message as Example 9-9.

Example 9-9 Error message when proper PSP is nor assigned

```
message: 'pods "details-v1-876bf485f-m84f8" is forbidden: unable to validate
against
  any pod security policy:
[spec.initContainers[0].securityContext.capabilities.add:
  Invalid value: "NET_ADMIN": capability may not be added]'
```

2. Next you will create the image policy.
 - a. The images for the bookInfo application are not in the whitelist. To run the application, create the file, `image.policy.yaml`, as in Example 9-10.

Example 9-10 Image policy for the bookInfo app

```
apiVersion: securityenforcement.admission.cloud.ibm.com/v1beta1
kind: ImagePolicy
metadata:
  name: book-info-images-whitelist
  namespace: istio-exp
spec:
  repositories:
    - name: docker.io/istio/*
```

- b. Then apply the policy with `kubectl apply -f image.policy.yaml`.
3. Now you will label the namespace for istio injection. Run the command `kubectl label namespace istio-exp istio-injection=enabled` to tag the target namespace with the flag of `istio-injection` to enable the automatic sidecar injection for the namespace.
4. Deploy the application by running the command:


```
kubectl -n istio-exp apply -f
istio/istio-1.0.6/samples/bookinfo/platform/kube/bookinfo.yaml
```
5. Validate that the pods are running as shown in Example 9-11.

Example 9-11 Validate that the pods are running

kubectl -n istio-exp get pods

NAME	READY	STATUS	RESTARTS	AGE
details-v1-876bf485f-k58pb	2/2	Running	0	45m
productpage-v1-8d69b45c-6thb9	2/2	Running	0	45m
ratings-v1-7c9949d479-sbt8p	2/2	Running	0	45m
reviews-v1-85b7d84c56-pntvg	2/2	Running	0	45m
reviews-v2-cbd94c99b-hbpzz	2/2	Running	0	45m
reviews-v3-748456d47b-qtcs5	2/2	Running	0	45m

Notice the 2/2 of the output. The sidecar injection adds the additional istio-proxy container and makes it into two containers.

9.4.3 Uninstallation

To uninstall Istio, run the command `helm delete istio --tls --purge` to delete and purge the release.

You will also need to delete the CustomResourceDefinition objects that are left over in this version. To delete these object run the command `kubectl delete -f istio/ibm-mgmt-istio/templates/crds.yaml`.

9.5 Service resiliency

In a distributed system, dealing with unexpected failures is one of the hardest problems to solve. For example what happens when an instance of microservice is unhealthy? Apart from detecting it, we need a mechanism to auto-correct it. With an appropriate liveness/readiness probe in POD specification we can detect if the pod is working correctly, and Kubernetes will restart it if pod is not functioning properly.

But to achieve service resiliency we need to address the following challenges.

- ▶ How to handle services which are working, but taking too long to respond due to certain environment issue?
- ▶ How to handle services that respond after certain number of retries and within a certain amount of time?
- ▶ How to stop the incoming traffic for sometime (if the service has a problem), wait for the service to “heal” itself and when it is working resume the inbound traffic automatically.
- ▶ How to set a timeout for a request landed on Kubernetes so that in a definite time frame a response is given to the client?
- ▶ How to auto-remove a pod, which is unhealthy for quite sometime?
- ▶ How to do load balancing for pods to increase the throughput and lower the latency?

Kubernetes does not fulfill the previous challenges as ready-for-use functionality; we need service mesh for it. Service mesh provides critical capabilities including service discovery, load balancing, encryption, observability, traceability, authentication, and authorization and support for the circuit breaker pattern. There are different solutions for service mesh, such as Istio, Linkerd, Conduit, and so on. We will use the Istio service mesh to discuss how to achieve service resiliency.

Istio comes with several capabilities for implementing resilience within applications. Actual enforcement of these capabilities happens in the *sidecar*. These capabilities will be discussed in the following sections within the context of a microservices example.

To understand the Istio capabilities we will use the example published on the following IBM Redbooks GitHub url:

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide/tree/master/CH8-Istio>.

The details about microservices setup are given in the readme file.

9.5.1 Retry

In the two following scenarios, the retry capability of Istio can be used:

- ▶ Transient, intermittent errors can come due to environment issues for example network, storage.
- ▶ The service or pod might have gone down only briefly.

With Istio's retry capability, it can make a few retries before having to truly deal with the error.

Example 9-12 the virtual service definition for the catalog service.

Example 9-12 Virtual service definition for the catalog service

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: catalog
spec:
  hosts:
  - catalog
  http:
  - route:
    - destination:
        host: catalog
      retries:
        attempts: 3
        perTryTimeout: 2s
```

1. Create a virtual service for the catalog. See Example 9-13.

Example 9-13 Create virtual service for catalog

```
root@scamp1:~/istio_lab# kubectl create -f catalog_retry.yaml
```

```
virtualservice.networking.istio.io/catalog created
```

2. Create a destination rule for all three microservices. See Example 9-14.

Example 9-14 Create a destination rule for all three microservices

```
root@scamp1:~/istio_lab# istioctl create -f destination_rule.yaml
```

```
Created config destination-rule/default/user at revision 2544618
Created config destination-rule/default/catalog at revision 2544619
Created config destination-rule/default/product at revision 2544621
```

3. Get cluster IP for the user microservice. See Example 9-15.

Example 9-15 Get cluster IP for user microservice

```
root@scamp1:~/istio_lab# kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
catalog	ClusterIP	10.0.0.31	<none>	8000/TCP	23h
kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	23h
product	ClusterIP	10.0.0.221	<none>	8000/TCP	23h
user	ClusterIP	10.0.0.55	<none>	8000/TCP	23h

4. Launch the user microservice to check the responses. See Example 9-16.

Example 9-16 Launch the user microservice to check the responses

```
root@scamp1:~/istio_lab# ./execute.sh 10.0.0.55
```

```
user==>catalog:v1==>product:Able to fetch infromation from product service
user==>catalog:v3==>product:Able to fetch infromation from product service
user==>catalog:v2==>product::Able to fetch infromation from product service
user==>catalog:v3==>product:Able to fetch infromation from product service
user==>catalog:v1==>product:Able to fetch infromation from product service
user==>catalog:v3==>product:Able to fetch infromation from product service
user==>catalog:v1==>product:Able to fetch infromation from product service
user==>catalog:v3==>product:Able to fetch infromation from product service
user==>catalog:v2==>product::Able to fetch infromation from product service
user==>catalog:v3==>product:Able to fetch infromation from product service
user==>catalog:v1==>product:Able to fetch infromation from product service
user==>catalog:v2==>product::Able to fetch infromation from product service
user==>catalog:v2==>product::Able to fetch infromation from product service
user==>catalog:v2==>product::Able to fetch infromation from product service
user==>catalog:v1==>product:Able to fetch infromation from product service
user==>catalog:v1==>product:Able to fetch infromation from product service
user==>catalog:v1==>product:Able to fetch infromation from product service
```

5. As we can see from the output in Example 9-16, the user microservice is randomly calling different versions of the catalog microservice. Now add a bug in catalog:v2 microservice and check how it behaves. See Example 9-17.

Example 9-17 Add a bug in the catalog:v2 microservice to make it unhealthy?

```
root@scamp1:~/istio_lab# kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
catalog-v1-5bf8c759b9-vbmvs	2/2	Running	0	1m
catalog-v2-547b5f6769-6qgzq	2/2	Running	0	1m
catalog-v3-569bd6c7d9-p9sgr	2/2	Running	0	1m
product-v1-747cf9f795-c4z5l	2/2	Running	0	1m
user-v1-6b5c74b477-cqr6b	2/2	Running	0	1m

```
root@scamp1:~/istio_lab# kubectl exec -it catalog-v2-547b5f6769-6qgzq -- curl
localhost:8000/unhealthy
```

```
Defaulting container name to catalog.
service got unhealthy
```

We will launch the user microservice to determine how it responds. See Example 9-18.

Example 9-18 Start the user microservice

```
root@scamp1:~/istio_lab# ./execute.sh 10.0.0.55
```

```
user==>catalog:v1==>product:Able to fetch infromation from product service
catalog:v2 service not available
catalog:v2 service not available
user==>catalog:v3==>product:Able to fetch infromation from product service
user==>catalog:v1==>product:Able to fetch infromation from product service
catalog:v2 service not available
user==>catalog:v3==>product:Able to fetch infromation from product service
```

```
user=>catalog:v1=>product:Able to fetch information from product service
user=>catalog:v1=>product:Able to fetch information from product service
user=>catalog:v1=>product:Able to fetch information from product service
catalog:v2 service not available
catalog:v2 service not available
user=>catalog:v3=>product:Able to fetch information from product service
user=>catalog:v1=>product:Able to fetch information from product service
catalog:v2 service not available
```

As we can see in the output, catalog:v2 microservice is not responding and all calls made to it from the user microservice fail.

6. We will add retry logic for virtualservice of the catalog and check how it behaves. See Example 9-19.

Example 9-19 Add retry logic in virtualservice of the catalog

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: catalog
spec:
  hosts:
  - catalog
  http:
  - route:
    - destination:
        host: catalog
    retries:
      attempts: 3
      perTryTimeout: 2s
```

-
7. The previous virtualservice definition for the catalog will make sure that each call to the catalog service will do 3 attempts of calling the catalog microservices before failure. Next, create a virtual service for the catalog using the definition in Example 9-20.

Example 9-20 Create a virtual service for catalog

```
root@scamp1:~/istio_lab# istioctl create -f catalog_retry.yaml
```

```
Created config virtual-service/default/catalog at revision 2553113
```

8. Then, examine the output of starting the user microservice, as shown in Example 9-21.

Example 9-21 Launching the user microservice

```
root@scamp1:~/istio_lab# ./execute.sh 10.0.0.55
```

```
user=>catalog:v1=>product:Able to fetch information from product service
user=>catalog:v3=>product:Able to fetch information from product service
user=>catalog:v1=>product:Able to fetch information from product service
user=>catalog:v1=>product:Able to fetch information from product service
user=>catalog:v3=>product:Able to fetch information from product service
user=>catalog:v1=>product:Able to fetch information from product service
user=>catalog:v1=>product:Able to fetch information from product service
user=>catalog:v3=>product:Able to fetch information from product service
user=>catalog:v3=>product:Able to fetch information from product service
user=>catalog:v3=>product:Able to fetch information from product service
```

```
user==>catalog:v1==>product:Able to fetch information from product service
user==>catalog:v1==>product:Able to fetch information from product service
user==>catalog:v3==>product:Able to fetch information from product service
user==>catalog:v3==>product:Able to fetch information from product service
```

As we can see in Example 9-21 on page 304, the user microservice no longer starts catalog:v2. As it tries to launch, the call fails. The user microservice retries 2 more times to start catalog:v2, and then launches catalog:v1 or catalog:v3, which succeeds.

9.5.2 Timeout

Calls to services over a network can result in unexpected behavior. We can only guess why the service has failed? Is it just slow? Is it unavailable? Waiting without any timeouts uses resources unnecessarily, causes other systems to wait, and is usually a contributor to cascading failures. *Your network traffic should always have timeouts in place, and you can achieve this goal with the timeout capability of Istio.* It can wait for only a few seconds before giving up and failing.

1. We will delete any virtualservice, destinationrule definitions on setup. Add 5 second delay in response of the catalog:v2 microservice, as shown in Example 9-22.

Example 9-22 Add 5 second delay in response of the catalog:v2 microservice

```
root@scamp1:~/istio_lab# kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
catalog-v1-5bf8c759b9-vbmvs	2/2	Running	0	1h
catalog-v2-547b5f6769-6qgzq	2/2	Running	0	1h
catalog-v3-569bd6c7d9-p9sgr	2/2	Running	0	1h
product-v1-747cf9f795-c4z5l	2/2	Running	0	1h
user-v1-6b5c74b477-cqr6b	2/2	Running	0	1h

```
root@scamp1:~/istio_lab# kubectl exec -it catalog-v2-547b5f6769-6qgzq -- curl
```

```
localhost:8000/timeout
```

```
Defaulting container name to catalog.
```

```
delayed added in microservice
```

2. Try to start the user microservice in Example 9-23.

Example 9-23 Launch user microservice

```
root@scamp1:~/istio_lab# ./execute.sh 10.0.0.55
```

```
user==>catalog:v1==>product:Able to fetch information from product service
user==>catalog:v3==>product:Able to fetch information from product service
user==>catalog:v3==>product:Able to fetch information from product service
user==>catalog:v1==>product:Able to fetch information from product service
user==>catalog:v3==>product:Able to fetch information from product service
user==>catalog:v2==>product::Able to fetch information from product service
user==>catalog:v1==>product:Able to fetch information from product service
user==>catalog:v3==>product:Able to fetch information from product service
user==>catalog:v1==>product:Able to fetch information from product service
```

As the script is run, you will notice that whenever a call goes to catalog:v2 microservice, it takes more than 5 seconds to respond, which is unacceptable for service resiliency.

3. Now we will create a VirtualService for the catalog and add 1 second as the timeout value. This is shown in Example 9-24.

Example 9-24 Create a VirtualService for the catalog and add 1 second as timeout

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: catalog
spec:
  hosts:
  - catalog
  http:
  - route:
    - destination:
        host: catalog
      timeout: 1.000s
```

4. Create a virtual service for the catalog as shown in Example 9-25.

Example 9-25 Create a virtual service for the catalog

```
root@scamp1:~/istio_lab# istioctl create -f catalog_timeout.yaml
Created config virtual-service/default/catalog at revision 2561104
```

5. We will now start the user microservice as shown in Example 9-26.

Example 9-26 Launch the user microservice

```
root@scamp1:~/istio_lab# ./execute.sh 10.0.0.55
user==>catalog:v3==>product:Able to fetch information from product service
upstream request timeout
upstream request timeout
user==>catalog:v1==>product:Able to fetch information from product service
user==>catalog:v1==>product:Able to fetch information from product service
user==>catalog:v3==>product:Able to fetch information from product service
upstream request timeout
user==>catalog:v1==>product:Able to fetch information from product service
upstream request timeout
user==>catalog:v3==>product:Able to fetch information from product service
user==>catalog:v3==>product:Able to fetch information from product service
user==>catalog:v3==>product:Able to fetch information from product service
user==>catalog:v1==>product:Able to fetch information from product service
upstream request timeout
user==>catalog:v1==>product:Able to fetch information from product service
user==>catalog:v1==>product:Able to fetch information from product service
upstream request timeout
user==>catalog:v1==>product:Able to fetch information from product service
```

As we can see in Example 9-26, whenever a call is made to catalog:v2, it timeouts in 1 sec according to the virtual service definition for the catalog.

9.5.3 Load balancer

All HTTP traffic bound to a service is automatically rerouted through Envoy. Envoy distributes the traffic across instances in the load balancing pool. Istio currently allows three load balancing modes: *round robin*, *random* and *weighted least request*.

9.5.4 Simple circuit breaker

Circuit breaking is an important pattern for creating resilient microservice applications. Circuit breaking allows you to write applications that limit the impact of failures, latency spikes, and other undesirable effects of network peculiarities.

We will show an example where we will not define circuit breaker rule and see how it works. See Example 9-27.

Example 9-27 Add timeout of 5 seconds for catalog:v2 microservice

```
root@scamp1:~/istio_lab/cb# kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
catalog-v1-5bf8c759b9-fjzp8	2/2	Running	0	2m
catalog-v2-547b5f6769-h4rkr	2/2	Running	0	2m
catalog-v3-569bd6c7d9-714c4	2/2	Running	0	2m
product-v1-747cf9f795-zwgt4	2/2	Running	0	2m
user-v1-6b5c74b477-r84tn	2/2	Running	0	2m

```
root@scamp1:~/istio_lab/cb# kubectl exec -it catalog-v2-547b5f6769-h4rkr -- curl localhost:8000/timeout
```

delayed added in microservice

1. We have added 5 seconds in the catalog:v2 microservice, it will take a minimum of 5 seconds to respond. ClusterIP for the catalog microservice in the setup is 10.0.0.31. See Example 9-28.

Example 9-28 Create a destinationrule for the catalog microservice

```
root@scamp1:~/istio_lab/cb# cat destination_rule.yaml
```

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: catalog
spec:
  host: catalog
  subsets:
  - labels:
      version: v1
    name: version-v1
  - labels:
      version: v2
    name: version-v2
  - labels:
      version: v3
    name: version-v3
```

```
root@scamp1:~/istio_lab/cb# kubectl create -f destination_rule.yaml
```

```
destinationrule.networking.istio.io/catalog created
```

2. Next, create a virtual service and destination rule for the catalog microservice. See Example 9-29.

Example 9-29 Create a virtual service for the catalog microservice

```
root@scamp1:~/istio_lab/cb# cat virtual_service_cb.yaml
```

```
apiVersion: networking.istio.io/v1alpha3
```

```
kind: VirtualService
```

```
metadata:
```

```
  name: catalog
```

```
spec:
```

```
  hosts:
```

```
  - catalog
```

```
  http:
```

```
  - route:
```

```
    - destination:
```

```
      host: catalog
```

```
      subset: version-v1
```

```
      weight: 25
```

```
    - destination:
```

```
      host: catalog
```

```
      subset: version-v2
```

```
      weight: 50
```

```
    - destination:
```

```
      host: catalog
```

```
      subset: version-v3
```

```
      weight: 25
```

```
root@scamp1:~/istio_lab/cb# kubectl create -f virtual_service_cb.yaml
```

```
virtualservice.networking.istio.io/catalog created
```

3. Now, try to check how the catalog microservice responds to requests. Use the `seige` tool to make a request to the catalog microservice. In Example 9-30 you can see that 5 clients are sending 8 concurrent requests to the catalog.

Example 9-30 Five clients are sending eight concurrent requests to the catalog

```
root@scamp1:~/istio_lab/cb# siege -r 8 -c 5 -v 10.0.0.31:8000
```

```
** SIEGE 4.0.4
```

```
** Preparing 5 concurrent users for battle.
```

```
The server is now under siege...
```

```
HTTP/1.1 200      0.02 secs:      74 bytes ==> GET  /
HTTP/1.1 200      0.03 secs:      74 bytes ==> GET  /
HTTP/1.1 200      5.05 secs:      75 bytes ==> GET  /
HTTP/1.1 200      5.06 secs:      75 bytes ==> GET  /
HTTP/1.1 200      5.04 secs:      75 bytes ==> GET  /
HTTP/1.1 200      0.03 secs:      74 bytes ==> GET  /
HTTP/1.1 200      0.03 secs:      74 bytes ==> GET  /
HTTP/1.1 200      0.02 secs:      74 bytes ==> GET  /
HTTP/1.1 200      0.01 secs:      74 bytes ==> GET  /
```

```

HTTP/1.1 200    5.09 secs:    75 bytes ==> GET /
HTTP/1.1 200    0.02 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.03 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.02 secs:    74 bytes ==> GET /
HTTP/1.1 200    5.14 secs:    75 bytes ==> GET /
HTTP/1.1 200    0.02 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.02 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.03 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.01 secs:    74 bytes ==> GET /
HTTP/1.1 200    5.03 secs:    75 bytes ==> GET /
HTTP/1.1 200    5.05 secs:    75 bytes ==> GET /
HTTP/1.1 200    5.01 secs:    75 bytes ==> GET /

```

.

.

```

Transactions:      40 hits
Availability:      100.00 %
Elapsed time:      25.17 secs
Data transferred:  0.00 MB
Response time:     2.03 secs
Transaction rate:  1.59 trans/sec
Throughput:        0.00 MB/sec
Concurrency:       3.22
Successful transactions: 40
Failed transactions: 0
Longest transaction: 5.14

```

As we can see in Example 9-30 on page 308, all requests to our system were successful, but it took some time to complete the test, as the v2 instance of catalog was a slow performer. But suppose that in a production system this 5 second delay was caused by too many concurrent requests to the same instance. This might cause cascade failures in your system.

4. Now, add a circuit breaker that will open whenever there is more than one request that is handled by the catalog microservice. See Example 9-31.

Example 9-31 Add a circuit breaker

```

root@scamp1:~/istio_lab/cb# cat destination_rule_cb.yaml

```

```

apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: catalog
spec:
  host: catalog
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
      tcp:
        maxConnections: 1
    outlierDetection:
      baseEjectionTime: 120.000s
      consecutiveErrors: 1
      interval: 1.000s
      maxEjectionPercent: 100

```

```

subsets:
- labels:
    version: v1
    name: version-v1
- labels:
    version: v2
    name: version-v2
- labels:
    version: v3
    name: version-v3

```

kubectl apply -f destination_rule_cb.yaml

Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
destinationrule.networking.istio.io/catalog configured

5. Because the circuit breaker for the catalog is in place, make a request to the catalog microservice. Example 9-32 shows 5 clients sending 8 concurrent requests to the catalog microservice.

Example 9-32 Make a request to the catalog microservice

root@scamp1:~/istio_lab/cb# siege -r 8 -c 5 -v 10.0.0.31:8000

```

** SIEGE 4.0.4
** Preparing 5 concurrent users for battle.
The server is now under siege...
HTTP/1.1 200    0.05 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.05 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.05 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.05 secs:    74 bytes ==> GET /
HTTP/1.1 503    0.02 secs:    57 bytes ==> GET /
HTTP/1.1 503    0.04 secs:    57 bytes ==> GET /
HTTP/1.1 200    0.04 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.05 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.04 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.02 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.01 secs:    74 bytes ==> GET /
HTTP/1.1 503    0.02 secs:    57 bytes ==> GET /
HTTP/1.1 503    0.00 secs:    57 bytes ==> GET /
HTTP/1.1 200    0.01 secs:    74 bytes ==> GET /
HTTP/1.1 503    0.01 secs:    57 bytes ==> GET /
HTTP/1.1 200    0.04 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.04 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.03 secs:    74 bytes ==> GET /
HTTP/1.1 503    0.02 secs:    57 bytes ==> GET /
HTTP/1.1 200    0.01 secs:    74 bytes ==> GET /
HTTP/1.1 200    5.05 secs:    75 bytes ==> GET /
HTTP/1.1 503    0.01 secs:    57 bytes ==> GET /
HTTP/1.1 200    0.01 secs:    74 bytes ==> GET /
HTTP/1.1 200    0.02 secs:    74 bytes ==> GET /

```

In Example 9-32, you can see the 503 errors being displayed. As the circuit breaker is being opened, whenever Istio detects more than one pending request being handled by the catalog microservice, it opens circuit breaker.

9.5.5 Pool ejection

Pool ejection is a strategy that works when you have a pool of pods to serve the requests. If a request that was forwarded to a pod fails, Istio will eject this pod from the pool for a sleep window. After the sleep window is over, it will be added to the pool again. This strategy makes sure we have functioning pods participating in the pool of instances. For more information, see the following link:

<https://istio.io/docs/reference/config/istio.networking.v1alpha3/#OutlierDetection>

9.6 Achieving E2E security for microservices using Istio

To achieve E2E security for microservices using Istio the following items should be taken into consideration.

9.6.1 Inbound traffic

For Inbound traffic in a Kubernetes environment, Kubernetes ingress resource is used to specify services that should be exposed outside the cluster. In an Istio service mesh, a better approach (which also works in both Kubernetes and other environments) is to use a different configuration model, namely Istio gateway. A gateway allows Istio features such as monitoring and route rules to be applied to the traffic entering the cluster. The following example shows inbound traffic.

1. Find the ingress port and ingress host IP for the Kubernetes cluster as shown in Example 9-33.

Example 9-33 Find the ingress port and ingress host IP for the Kubernetes cluster

```
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o
jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
```

```
export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service
istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')
```

```
root@scamp1:~# echo $INGRESS_PORT
31380
```

```
root@scamp1:~# echo $SECURE_INGRESS_PORT
31390
```

```
root@scamp1:~# export INGRESS_HOST=$(kubectl get po -l istio=ingressgateway -n
istio-system -o 'jsonpath={.items[0].status.hostIP}')
```

```
root@scamp1:~# echo $INGRESS_HOST
XX.XX.XX.XX
```

2. Create an ingress gateway and attach it to the user virtual service, as shown in Example 9-34.

Example 9-34 Create an ingress gateway and attach it the user virtual service

```
root@scamp1:~/istio_lab# cat gateway.yaml
```

```
apiVersion: networking.istio.io/v1alpha3
```

```

kind: Gateway
metadata:
  name: app-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: app
spec:
  hosts:
  - "*"
  gateways:
  - app-gateway
  http:
  - match:
    - uri:
        exact: /
    route:
    - destination:
        host: user
        port:
          number: 8000

```

```

root@scamp1:~/istio_lab# kubectl create -f gateway.yaml
gateway.networking.istio.io/app-gateway created
virtualservice.networking.istio.io/app created

```

We have now created a virtual service configuration for the user service containing one route rule that allows traffic for path /. The gateways list specifies that only requests through your app-gateway are allowed. All other external requests will be rejected with a 404 response. Note that in this configuration, internal requests from other services in the mesh are not subject to these rules, but instead will default to round-robin routing. To apply these or other rules to internal calls, you can add the special value mesh to the list of the gateways.

3. Try to access the user service using the **curl** command shown in Example 9-35.

Example 9-35 Access the user service

```

root@scamp1:~/istio_lab# curl $INGRESS_HOST:$INGRESS_PORT

user=>catalog:v1=>product:Able to fetch information from product service

```

If you want to expose HTTPs endpoint for inbound traffic, see the following link:

<https://istio.io/docs/tasks/traffic-management/secure-ingress/>

9.6.2 Outbound traffic

By default, Istio-enabled services are unable to access URLs outside of the cluster, because the pod uses iptables to transparently redirect all outbound traffic to the sidecar proxy, which only handles intra-cluster destinations.

This section describes how to configure Istio to expose external services to Istio-enabled clients. You will learn how to enable access to external services by defining ServiceEntry configurations, or alternatively, to bypass the Istio proxy for a specific range of IPs.

Configuring the external service

Using Istio ServiceEntry configurations, you can access any publicly accessible service from within your Istio cluster. Example 9-36 shows that you can access `httpbin.org` and `www.google.com`.

Example 9-36 Access `httpbin.org` and `www.google.com`

```
cat <<EOF | kubectl apply -f -
```

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: google
spec:
  hosts:
  - www.google.com
  ports:
  - number: 443
    name: https
    protocol: HTTPS
  resolution: DNS
  location: MESH_EXTERNAL
```

```
---
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: google
spec:
  hosts:
  - www.google.com
  tls:
  - match:
    - port: 443
      sni_hosts:
      - www.google.com
    route:
    - destination:
        host: www.google.com
        port:
          number: 443
      weight: 100
```

```
EOF
```

In Example 9-36 on page 313 we have created a serviceentry and a virtualservice to allow access to an external HTTP service. Note that for TLS protocols, including HTTPS, the TLS virtualservice is required in addition to the serviceentry.

1. Create a ServiceEntry to allow access to an external HTTP service Example 9-37.

Example 9-37 Create a ServiceEntry

```
root@scamp1:~/istio_lab# kubectl apply -f -
```

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: httpbin-ext
spec:
  hosts:
  - httpbin.org
  ports:
  - number: 80
    name: http
    protocol: HTTP
  resolution: DNS
  location: MESH_EXTERNAL
EOF
```

2. Make requests to the external services. Make an HTTP outbound request to httpbin.org from the user pod, as shown in Example 9-38.

Example 9-38 HTTP outbound request to httpbin.org from the user to the pod

```
root@scamp1:~/istio_lab# export SOURCE_POD=$(kubectl get pod -l app=user -o
jsonpath={.items..metadata.name})
```

```
root@scamp1:~/istio_lab# kubectl exec -it $SOURCE_POD -c user curl
http://httpbin.org/headers
```

```
{
  "headers": {
    "Accept": "*/*",
    "Cache-Control": "max-stale=0",
    "Host": "httpbin.org",
    "If-Modified-Since": "Thu, 14 Mar 2019 06:00:44 GMT",
    "User-Agent": "curl/7.38.0",
    "X-B3-Sampled": "1",
    "X-B3-Spanid": "51c9d4d21b2e4f7c",
    "X-B3-Traceid": "51c9d4d21b2e4f7c",
    "X-Bluecoat-Via": "6f5b02aba0abb15e",
    "X-Envoy-Decorator-Operation": "httpbin.org:80/*",
    "X-Istio-Attributes":
"CikKGGRIc3RpbmF0aW9uLnN1cnZpY2UubmFtZRIEgtodHRwYm1uLm9yZwoqCh1kZXN0aW5hdG1vbi5zZ
XJ2aWN1Lm5hbWVzcGFjZRIJEgdkZWZhdWx0CiQKE2R1c3RpbmF0aW9uLnN1cnZpY2USDRILaHR0cGJpbi5
vcmcKPQoKc291cmN1LnVpZBIvEi1rdWJlcm5ldGVzOi8vdXN1ci12MS02YjVjNzRiNDc3LXd4dGN6LmRlZ
mF1bHQKKQoYZGVzdGluYXRpb24uc2Vydm1jZS5ob3N0Eg0SC2h0dHBiaW4ub3Jn"
  }
}
```

Similarly, we can access <https://google.com> from any pod on the Kubernetes setup. We will be able to access it because we have created a relevant serviceentry and virtualservice for the same.

9.6.3 Mutual TLS authentication

Istio tunnels service-to-service communication through the client-side and server-side Envoy proxies. For a client to call a server, the steps followed are:

1. Istio reroutes the outbound traffic from a client to the client's local sidecar Envoy.
2. The client-side Envoy starts a mutual TLS handshake with the server-side Envoy. During the handshake, the client-side Envoy also performs a secure naming check to verify that the service account presented in the server certificate is authorized to run the target service.
3. The client-side Envoy and the server-side Envoy establish a mutual TLS connection, and Istio forwards the traffic from the client-side Envoy to the server-side Envoy.
4. After the authorization, the server-side Envoy forwards the traffic to the server service through local TCP connections.

We will use an example for mutual TLS. See Example 9-39.

Example 9-39 Create mutual TLS mesh-wide policy

```
root@scamp1:~# cat <<EOF | kubectl apply -f -

> apiVersion: "authentication.istio.io/v1alpha1"
> kind: "MeshPolicy"
> metadata:
>   name: "default"
> spec:
>   peers:
>     - mtls: {}
> EOF
meshpolicy.authentication.istio.io/default created
```

This policy specifies that all workloads in the service mesh will only accept encrypted requests using TLS. As you can see, this authentication policy has the kind: *MeshPolicy*. The name of the policy must be default and it contains no targets specification (as it is intended to apply to all services in the mesh). At this point, only the receiving side is configured to use the mutual TLS. If you run the `curl` command between the Istio services (for example those with the sidecars), all requests will fail with a 503 error code as the client side is still using plain-text. See Example 9-40.

Example 9-40 Try to access the catalog microservice from the user microservice

```
root@scamp1:~# kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
catalog	ClusterIP	10.0.0.31	<none>	8000/TCP	4d
kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	4d
product	ClusterIP	10.0.0.221	<none>	8000/TCP	4d
sleep	ClusterIP	10.0.0.249	<none>	80/TCP	5h
user	ClusterIP	10.0.0.55	<none>	8000/TCP	4d

```
root@scamp1:~# kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
catalog-v1-5bf8c759b9-pcphd	2/2	Running	0	1h
catalog-v2-547b5f6769-7sdkd	2/2	Running	0	1h
catalog-v3-569bd6c7d9-sgz8h	2/2	Running	0	1h
product-v1-747cf9f795-z5ps2	2/2	Running	0	1h
sleep-64cbb8bf78-2w85v	2/2	Running	0	1h
user-v1-6b5c74b477-svtj7	2/2	Running	0	1h

```
root@scamp1:~# kubectl exec -it user-v1-6b5c74b477-svtj7 -- curl 10.0.0.31:8000
```

Defaulting container name to user.
upstream connect error or disconnect/reset before headers

To configure the client side, you need to set destination rules to use mutual TLS. It is possible to use multiple destination rules, one for each applicable service (or namespace). However, it is more convenient to use a rule with the * wildcard to match all services so that the configuration is on par with the mesh-wide authentication policy. See Example 9-41.

Example 9-41 Set the destination rule to use the mutual TLS

```
root@scamp1:~# cat <<EOF | kubectl apply -f -
> apiVersion: "networking.istio.io/v1alpha3"
> kind: "DestinationRule"
> metadata:
>   name: "default"
>   namespace: "default"
> spec:
>   host: "*.local"
>   trafficPolicy:
>     tls:
>       mode: ISTIO_MUTUAL
> EOF
destinationrule.networking.istio.io/default created
```

Now, try to make a call to the catalog microservice from the user microservice. See Example 9-42.

Example 9-42 Make a call to the catalog microservice from the user microservice

```
root@scamp1:~# kubectl exec -it user-v1-6b5c74b477-svtj7 -- curl 10.0.0.31:8000
```

Defaulting container name to user.
user==>catalog:v2==>product::Able to fetch information from product service

9.6.4 White or black listing

Istio supports attribute-based whitelists and blacklists. Using Istio you can control access to a service based on any attributes that are available within Mixer.

White listing

The whitelist is a deny everything rule, except for the approved invocation paths:

1. In the following example, you white list calls from the user microservice to the catalog microservice. See Example 9-43.

Example 9-43 White listing calls from user microservice to catalog microservice

```
root@scamp1:~/istio_lab# cat catalog_whitelist.yaml
```

```
apiVersion: "config.istio.io/v1alpha2"
kind: listchecker
metadata:
  name: catalogwhitelist
spec:
  overrides: ["user"]
  blacklist: false
---
apiVersion: "config.istio.io/v1alpha2"
kind: listentry
metadata:
  name: catalogsource
spec:
  value: source.labels["app"]
---
apiVersion: "config.istio.io/v1alpha2"
kind: rule
metadata:
  name: checkfromuser
spec:
  match: destination.labels["app"] == "catalog"
  actions:
    - handler: catalogwhitelist.listchecker
      instances:
        - catalogsource.listentry
```

```
root@scamp1:~/istio_lab# istioctl replace -f catalog_whitelist.yaml
```

```
Updated config listchecker/default/catalogwhitelist to revision 3251839
Updated config listentry/default/catalogsource to revision 3251840
Updated config rule/default/checkfromuser to revision 3251841
```

2. We will try to make calls from the user microservice to the catalog microservice. See Example 9-44.

Example 9-44 Make calls from the user microservice to the catalog microservice

```
root@scamp1:~/istio_lab# kubectl get svc
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
catalog	ClusterIP	10.0.0.31	<none>	8000/TCP	4d
kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	4d

product	ClusterIP	10.0.0.221	<none>	8000/TCP	4d
sleep	ClusterIP	10.0.0.249	<none>	80/TCP	6h
user	ClusterIP	10.0.0.55	<none>	8000/TCP	4d

```
root@scamp1:~/istio_lab# kubectl get po
```

NAME	READY	STATUS	RESTARTS	AGE
catalog-v1-5bf8c759b9-pcphd	2/2	Running	0	1h
catalog-v2-547b5f6769-7sdkd	2/2	Running	0	1h
catalog-v3-569bd6c7d9-sgz8h	2/2	Running	0	1h
product-v1-747cf9f795-z5ps2	2/2	Running	0	1h
sleep-64cbb8bf78-2w85v	2/2	Running	0	1h
user-v1-6b5c74b477-svtj7	2/2	Running	0	1h

```
root@scamp1:~/istio_lab# kubectl exec -it user-v1-6b5c74b477-svtj7 -- curl
10.0.0.31:8000
```

Defaulting container name to user.
user==>catalog:v3==>product:Able to fetch information from product service

As we can see in Example 9-44 on page 317, the user microservice call can be made to the catalog microservice.

- Now we will try to make a call from the product microservice to the catalog microservice. See Example 9-45.

Example 9-45 Make a call from the product microservice to the catalog microservice

```
root@scamp1:~/istio_lab# kubectl exec -it product-v1-747cf9f795-z5ps2 -- curl
10.0.0.31:8000
```

Defaulting container name to product.
NOT_FOUND:catalogwhitelist.listchecker.default:product is not whitelisted

Example 9-45 shows that the call from the product microservice to the catalog microservice has failed, as we had only white listed the user microservice. This is the wanted result.

Black listing

The black list is explicit denials of particular invocation paths.

- In Example 9-46 we black list the user microservice in the catalog microservice, so that it cannot make calls to the catalog microservice.

Example 9-46 Black listing the user microservice to the catalog microservice

```
root@scamp1:~/istio_lab# cat catalog_blacklist.yaml
```

```
apiVersion: "config.istio.io/v1alpha2"
kind: denier
metadata:
  name: denyuserhandler
spec:
  status:
    code: 7
    message: Not allowed
---
apiVersion: "config.istio.io/v1alpha2"
```

```

kind: checknothing
metadata:
  name: denyuserrequests
spec:
---
apiVersion: "config.istio.io/v1alpha2"
kind: rule
metadata:
  name: denycustomer
spec:
  match: destination.labels["app"] == "catalog" && source.labels["app"]=="user"
  actions:
    - handler: denyuserhandler.denier
      instances: [ denyuserrequests.checknothing ]

```

2. We will make a call from the user microservice to the catalog microservice as shown in Example 9-47.

Example 9-47 Make a call from the user microservice to the catalog microservice

```

root@scamp1:~/istio_lab# kubectl exec -it user-v1-6b5c74b477-svtj7 -- curl
10.0.0.31:8000

```

```

Defaulting container name to user
PERMISSION_DENIED:denyuserhandler.denier.default:Not allowed

```

As we had blacklisted user microservice to the catalog microservice so call has failed in Example 9-47.

3. Make a call from the product microservice to the user microservice. See Example 9-48.

Example 9-48 Make a call from the product microservice to the user microservice

```

root@scamp1:~/istio_lab# kubectl exec -it product-v1-747cf9f795-z5ps2 -- curl
10.0.0.31:8000

```

```

Defaulting container name to product.
user==>catalog:v2==>product::Able to fetch infromation from product service

```

The previous call has succeeded because we did not blacklist the product microservice.

9.6.5 Istio authorization

Istio's authorization feature, also known as role-based access control (RBAC), provides namespace-level, service-level, and method-level access control for services in an Istio mesh. It features:

- ▶ Role-based semantics, which are simple and easy to use.
- ▶ Service-to-service and end-user-to-service authorization.
- ▶ Flexibility through custom properties support, for example conditions, in roles and role-bindings.
- ▶ High performance, as Istio authorization is enforced natively on Envoy.

See the Istio authorization architecture in Figure 9-3.

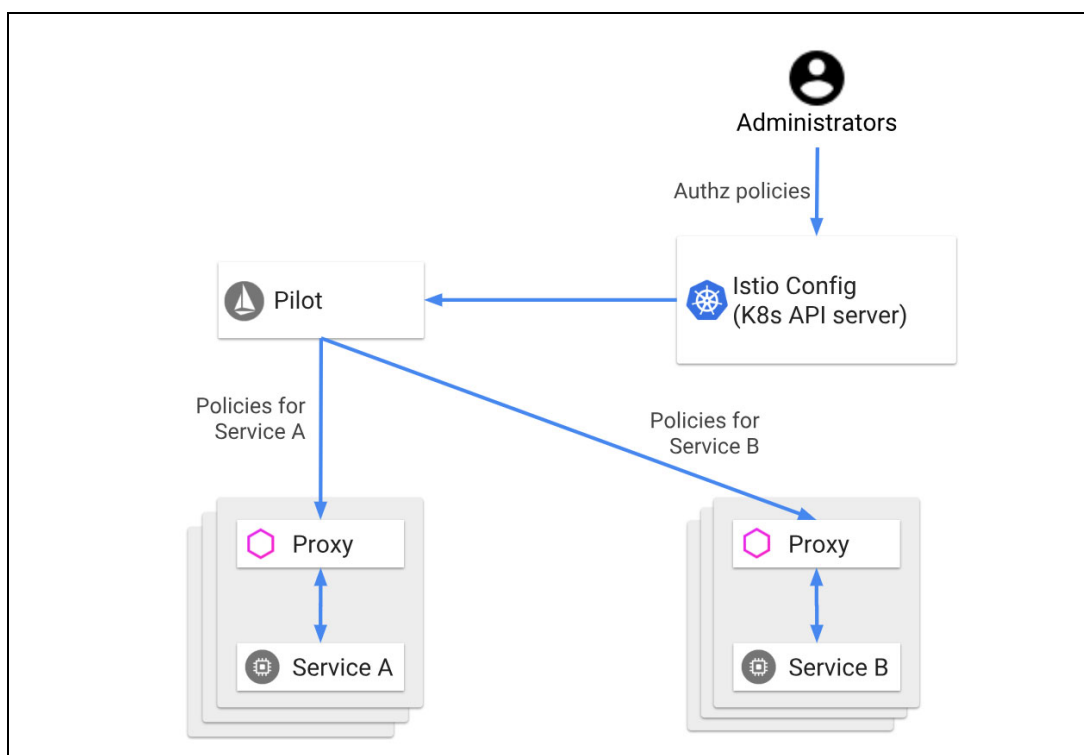


Figure 9-3 Istio authorization architecture³

Figure 9-3 shows the basic Istio authorization architecture. Operators specify Istio authorization policies by using .yaml files. When Istio is deployed, it saves the policies in the Istio Config Store. Pilot watches for changes to Istio authorization policies. It fetches the updated authorization policies if it sees any changes. Pilot distributes Istio authorization policies to the Envoy proxies that are colocated with the service instances.

Each Envoy proxy runs an authorization engine that authorizes requests at run time. When a request comes to the proxy, the authorization engine evaluates the request context against the current authorization policies, and returns the authorization result, ALLOW or DENY.

1. The first thing to do is enable Istio authorization by using the RbacConfig object. See Example 9-49.

Example 9-49 Enable Istio authorization

```
root@scamp1:~/istio_lab# cat << EOF | kubectl apply -f -
```

```
> apiVersion: "rbac.istio.io/v1alpha1"
> kind: RbacConfig
> metadata:
>   name: default
> spec:
>   mode: 'ON_WITH_INCLUSION'
>   inclusion:
>     namespaces: ["default"]
> EOF
rbacconfig.rbac.istio.io/default created
```

³ Image taken from <https://istio.io/docs/concepts/security/>

2. Try to make a call to the user microservice. See Example 9-50.

Example 9-50 Make a call to the user microservice

```
root@scamp1:~/istio_lab# curl 10.0.0.55:8000
```

```
RBAC: access denied
```

By default, Istio uses a deny by default strategy, meaning that nothing is permitted until you explicitly define access control policy to grant access to any service.

3. Now, grant access to any user to any service of our mesh (user, catalog, or product) only if the communication goes through the GET method. See Example 9-51.

Example 9-51 Grant access to any user to any service

```
root@scamp1:~/istio_lab# cat << EOF | kubectl apply -f -
```

```
> apiVersion: "rbac.istio.io/v1alpha1"
> kind: ServiceRole
> metadata:
>   name: service-viewer
> spec:
>   rules:
>     - services: ["*"]
>       methods: ["GET"]
>       constraints:
>         - key: "destination.labels[app]"
>           values: ["user", "catalog", "product"]
> ---
> apiVersion: "rbac.istio.io/v1alpha1"
> kind: ServiceRoleBinding
> metadata:
>   name: bind-service-viewer
>   namespace: default
> spec:
>   subjects:
>     - user: "*"
>   roleRef:
>     kind: ServiceRole
>     name: "service-viewer"
> EOF
```

4. Now if you make a GET call to the user microservice, it should succeed as seen in Example 9-52.

Example 9-52 Make a GET call to the user microservice

```
root@scamp1:~/istio_lab# curl 10.0.0.55:8000
```

```
user==>catalog:v2==>product::Able to fetch information from product service
```

For more information about RBAC, see the following link:

<https://istio.io/docs/tasks/security/role-based-access-control/>



Part 3

Cloud Foundry related topics

The last part of this book covers Cloud Foundry related topics.



IBM Cloud Private Cloud Foundry and common systems administration tasks

In this chapter we introduce Cloud Foundry on IBM Cloud Private and the related technologies. We also discuss installation and common systems administration tasks for IBM Cloud Private Cloud Foundry.

This chapter has the following sections:

- ▶ 10.1, “Introduction” on page 326
- ▶ 10.2, “Installation and extensions” on page 327
- ▶ 10.3, “High availability installation” on page 333
- ▶ 10.4, “Backup and restore strategy” on page 335
- ▶ 10.5, “Storage and persistent volumes” on page 336
- ▶ 10.6, “Sizing and licensing” on page 337
- ▶ 10.7, “Networking” on page 338
- ▶ 10.8, “Security” on page 338
- ▶ 10.9, “Monitoring and logging” on page 340
- ▶ 10.10, “Integrating external services” on page 342
- ▶ 10.11, “Applications and buildpacks” on page 343
- ▶ 10.12, “iFix and releases” on page 344

10.1 Introduction

Cloud Foundry continues to evolve as an application service platform. It allows developers to focus on code and not application infrastructure. Its ease of use allows new developers to publish applications from day one, without having to worry about hosting, routing, scaling, and high availability. Cloud Foundry provides all these operational capabilities in a way that's easy for developers to consume, while being scalable and easily maintained by operations.

Cloud Foundry allows development in several languages, and has provisions to allow for containers or binaries to be incorporated as applications. This gives an enterprise endless flexibility, while maintaining ease of operation at scale.

Operationally, networking, applications, routing, and services are ritualized, so the physical infrastructure requirements are related to configuring an appropriate IaaS. IBM provides the stemcell, which guarantees that all platform VMs are cut from the same binaries, and also allows for a single stemcell patch to easily be propagated to all systems in the platform. This helps to lower the operational workload usually spent patching and maintaining systems.

Cloud Foundry takes this concept one step further to the garden containers. By basing each application container on the cflinuxfs base, security and container improvements are propagated to each application. This means that operations does not need to worry about container security on a per application basis. It frees operations to work on maintenance at a higher scale, and development can focus on coding and not application hosting foundations.

10.1.1 IaaS flavors

IBM Cloud Private Cloud Foundry is available on VMWare, OpenStack, AWS and on IBM Cloud Foundry (technical preview) itself.

10.1.2 Technology BOSH versus Kubernetes

One is not necessarily better than the other, but both can be used to serve the needs of your teams. BOSH (<https://bosh.io/docs/>) works well if you have a large virtual machine infrastructure such as VMware or OpenStack, where an IaaS team has been managing and dealing with these types of resources for many years. Full stack works seamlessly in these scenarios deploying, using native infrastructure APIs and then managing and monitoring the virtual systems.

Kubernetes abstracts the IaaS from the platform, so there is no need to worry about IaaS information or dealing with the systems. Kubernetes allows the focus to be shifted to platform management and development pursuits. It deploys quickly and scales fast. It consumes Kubernetes resources, no matter what underlying IaaS is being used.

This lowers the entry bar to Cloud Foundry when Kubernetes is already deployed, as well as provides a base for local services to be provisioned on demand for Cloud Foundry. This does preclude that Kubernetes has been configured, and it will be sitting on an IaaS, so this management piece is pushed down the stack.

In most cases you have both the IaaS and the Kubernetes. If you're not running Cloud Foundry on Kubernetes, you will likely still use IBM's Cloud Private platform for shared monitoring, logging, and data services.

Frequently asked questions:

- *Does IBM Cloud Private Cloud Foundry Enterprise Environment install IBM Cloud Private?*

IBM Cloud Private is a pre-requirement for the installation of IBM Cloud Private CFEE. IBM Cloud Private CFEE gets installed on top of an IBM Cloud Private installation.

- *Do I need IBM Cloud Private to install IBM Cloud Private Cloud Foundry?*

IBM Cloud Private Cloud Foundry (not the CFEE flavor) doesn't need IBM Cloud Private to get installed. IBM Cloud Private is only needed for the IBM Cloud Private CFEE flavor. The IBM Cloud Private Cloud Foundry can be installed on different platforms (AWS, OpenStack and VMWare) and it relies on BOSH CPI to request resources on these platforms.

10.2 Installation and extensions

In this section we introduce the installation process and the notions of extensions which allows the deployer to customize the installation.

The installation is done in two steps, first the creation of the installer container and second Cloud Foundry deployment itself using the Cloud Foundry deployment tool.

The customer must download the installation kit from (link). The installation kit, if needed, must be copied to a Linux base server that has network connectivity with the targeted environment.

The installer host has different requirements depending if the target environment is Cloud Foundry Enterprise Environment (CFEE) or Cloud Foundry Full Stack. The installer container is created either by using the provided scripts or by deploying the provided Cloud Pak in IBM Cloud Private. Once the installer container is created, the installation is similar on all supported platforms. The installer container is used only for deployment and upgrades. Once deployment or upgrade of Cloud Foundry is done, the installer container is not needed anymore.

10.2.1 Installation of the installer container in a Cloud Foundry Full Stack environment

For Cloud Foundry Full Stack, the installer host is the server where the installation kit is downloaded and run. In case of an airgap environment, the deployer needs to copy the installation kit to a server that has access to the target environment. One prerequisite is to have Docker installed on that server. The deployer decompresses the installation kit and then runs the `import_image.sh` script which reads the installer kit directory.

It then uploads all images into Docker and executes `launch.sh` with some parameters to create the *installation container*. One of the parameters is the directory where the installation data should reside. It is useful to have a backup of the data directory and this eases the subsequent upgrade of the Cloud Foundry Full Stack as this directory contains all configuration parameters for the installation and upgrades. Once the installer container is installed, the environment is ready to get deployed.

Figure 10-1 shows the installation flow for a Cloud Foundry Full Stack environment.

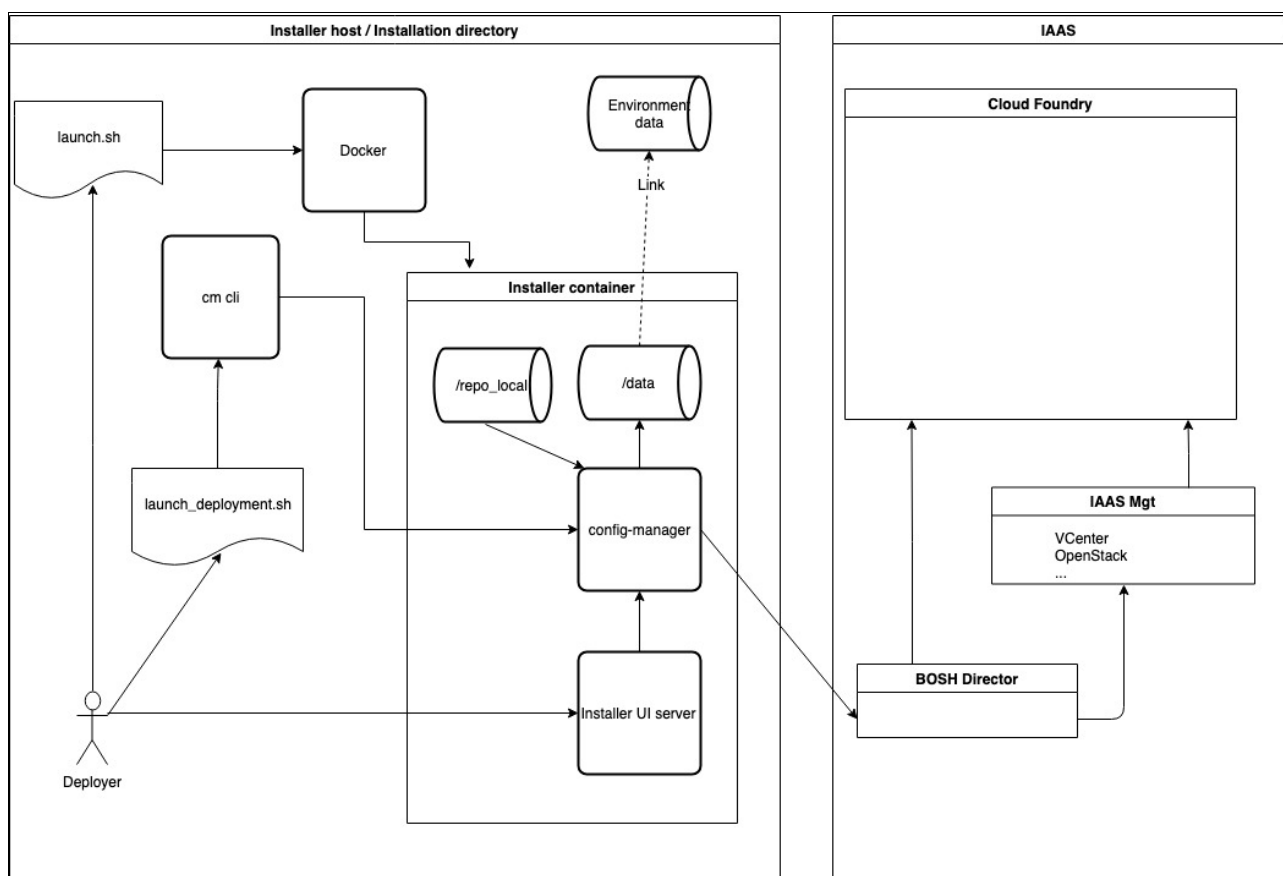


Figure 10-1 Installation flow for a Cloud Foundry Full Stack environment

10.2.2 Installation of the installer container in a CFEE environment

For CFEE the installation is similar to the process for the Cloud Foundry Full Stack environment. The deployer downloads and copies the installation kit to a server which has access to IBM Cloud Private. After that the deployer launches the **load_cloudpak.sh**, which loads all artifacts to IBM Cloud Private. Browsing the IBM Cloud Private catalog, the deployer selects the **ibm-cfee-installer**. Check the listed prerequisite (for example, persistent volume) and deploy it.

A pod is created with the “installer container” as it is for the Cloud Foundry Full Stack solution. The deployer uses the *Cloud Foundry deployment tool* to start the deployment of Cloud Foundry. It is useful to have the installer persistent volume backed up to ease the Cloud Foundry upgrades in case of data lost.

Figure 10-2 shows the installation flow for a Cloud Foundry Enterprise Environment.

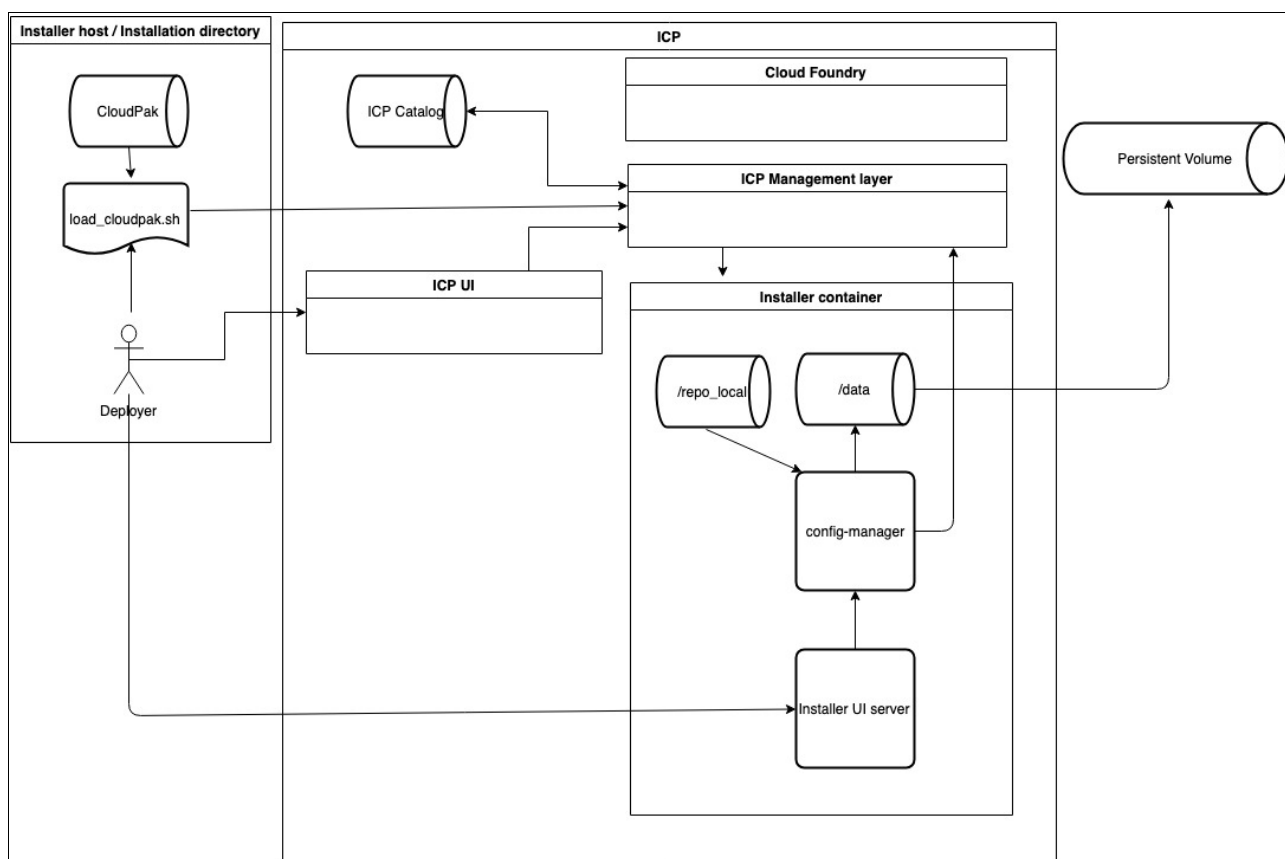


Figure 10-2 Installation flow for a Cloud Foundry Enterprise Environment

10.2.3 Config-manager role

The installer container runs the “config-manager” server. Its role is to orchestrate the deployment of the different extensions that compose the Cloud Foundry deployment. Each extension is wrapped as a Docker volume image and it provides the orchestration template, scripts, and assets to execute its own deployment. The main extensions are the “cfp-kubernetes-cf” for CFEE and “cfp-bosh-template” for Cloud Foundry Full Stack platforms.

The “config-manager” server reads the orchestration template from the extension-manifest.yml of the extension and starts its execution. The deployer can interact with the “config-manager” through the “cm cli” which is installed on the installation directory.

When the installer container is installed, either the installation can be executed using the script `launch_deployment.sh` for Cloud Foundry Full Stack, or through the deployment tool for Cloud Foundry Full Stack and CFEE. For Cloud Foundry Full Stack the deployment tool is accessible at the url `http://<installer_host_ip>:8180`. For CFEE, the deployer can launch the installer deployment tool from the IBM Cloud Private management console.

Figure 10-3 and Figure 10-4 show some screen captures of the Cloud Foundry deployment tool.

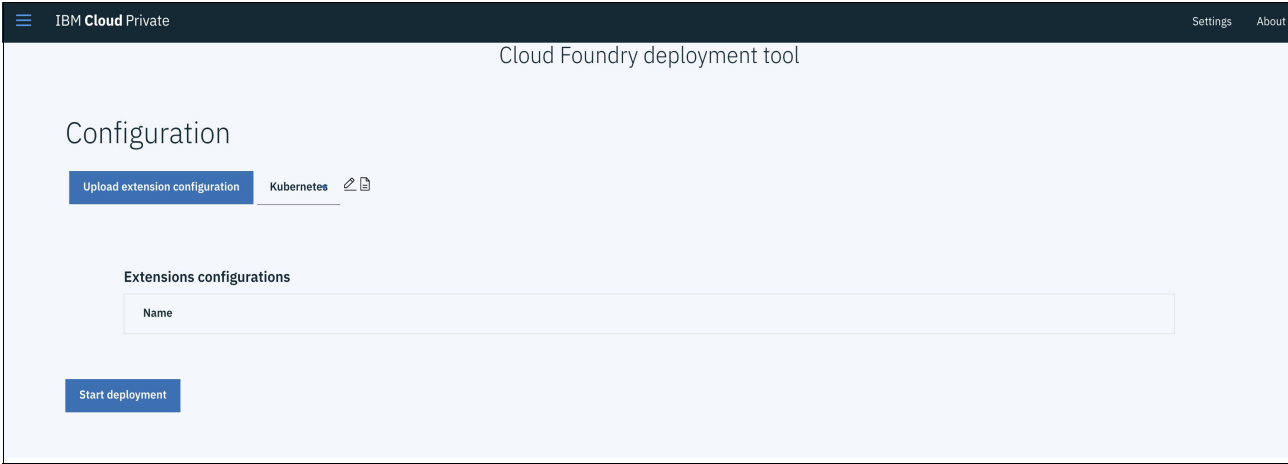


Figure 10-3 Configuration page

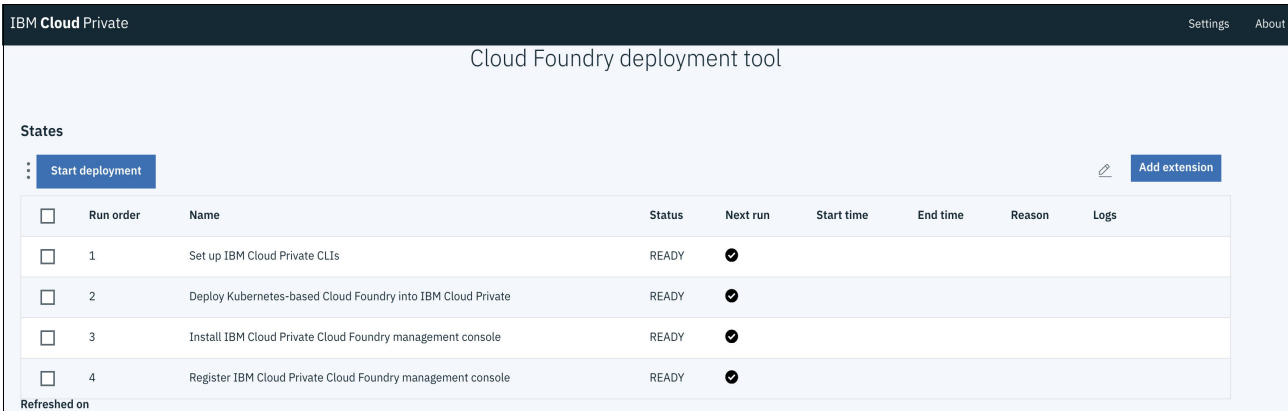


Figure 10-4 States page

10.2.4 Extensions

As mentioned, the base role of the config-manager is to deploy extensions.

IBM provides a number of embedded extensions to integrate external services. For example for Cloud Foundry Full Stack, IBM provides ready to use extensions to integrate LDAP, application logs, and system logs.

Figure 10-5 shows the extensions page.

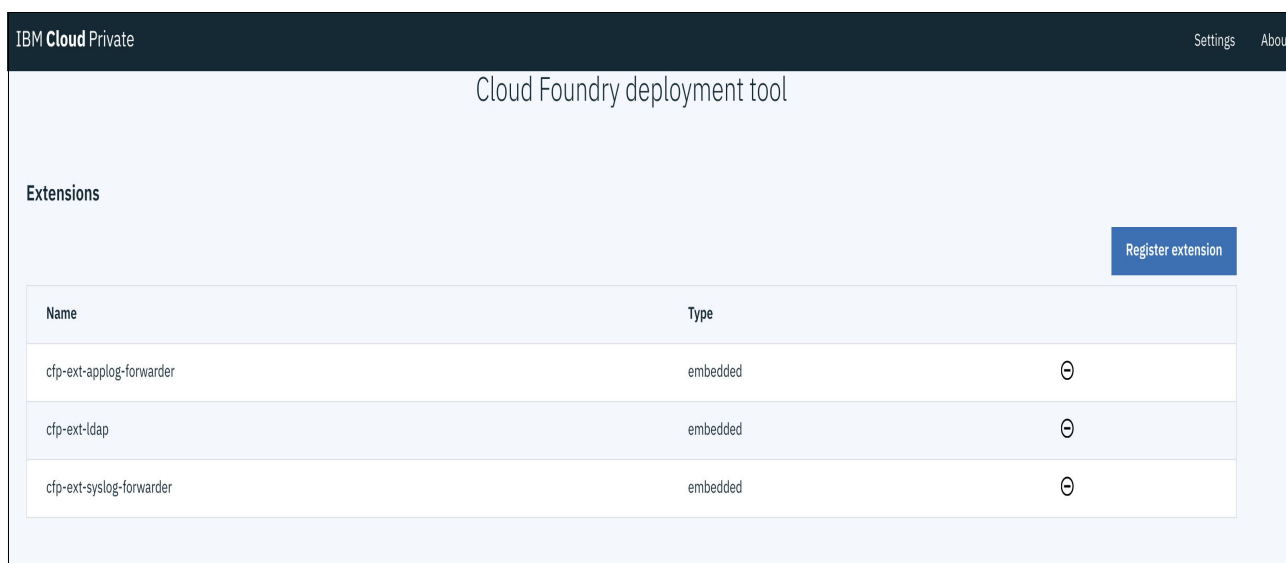


Figure 10-5 Extensions page

An extension developer can create their own extensions by using the extension framework provided in the CFEE and Cloud Foundry Full Stack solution. These extensions are called “custom” extensions.

An extension is first developed, then registered, and then integrated in the main deployment flow.

An extension is basically composed of an `extension-manifest.yml`, script, and assets. The `extension-manifest.yml` contains different sections:

- **global parameters:** These define how the extension should behave when getting registered.
- **states:** These define the different steps of the extension deployment. Each step calls a command (script or executable), and provides the log path.
- **ui_metadata:** If defined, the Cloud Foundry deployment tool provides pages to enter the parameters needed for that extension.

Once the extension is developed, it is compressed in a .zip file. Through the deployment console or the command line, the deployer can register the extension and integrate it in the deployment flow.

An extension can be also set to auto-place itself into the flow by defining in the `extension-manifest.yml` where the extension needs to be placed.

Figure 10-6 shows the extension directory structure.

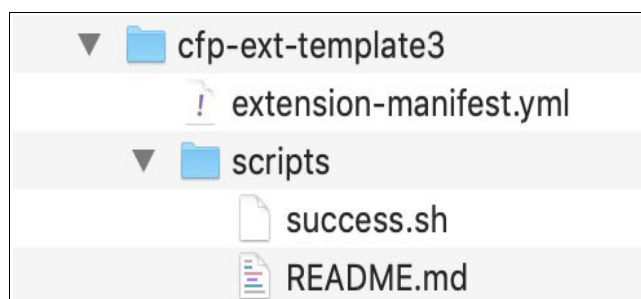


Figure 10-6 Extension directory structure

Example 10-1 shows an extension-manifest.yml file.

Example 10-1 An extension-manifest.yml file

```
extension:
  name: extension-sample
  version: 1.0
  type: bosh-release
ui_metadata:
  default:
    groups:
      - name: "ext_example"
        title: "Extension example"
        properties:
          - name: "param_1"
            label: "First parameter"
            description: "This is a sample"
            type: "text"
            validation-regex: ".*"
            mandatory: true
            sample-value: "E.g. my sample value"
            default: "my sample value"
states:
- name: task1
  phase: ""
  script: scripts/success.sh task1
  next_state:s [ task2 ]
- name: task2
  phase: ""
  script: scripts/success.sh task2
  next_states: [ task3 ]
- name: task3
  phase: ""
  script: scripts/success.sh task3
```

This configuration generates the following state sequence.

Figure 10-7 shows the extension states page.

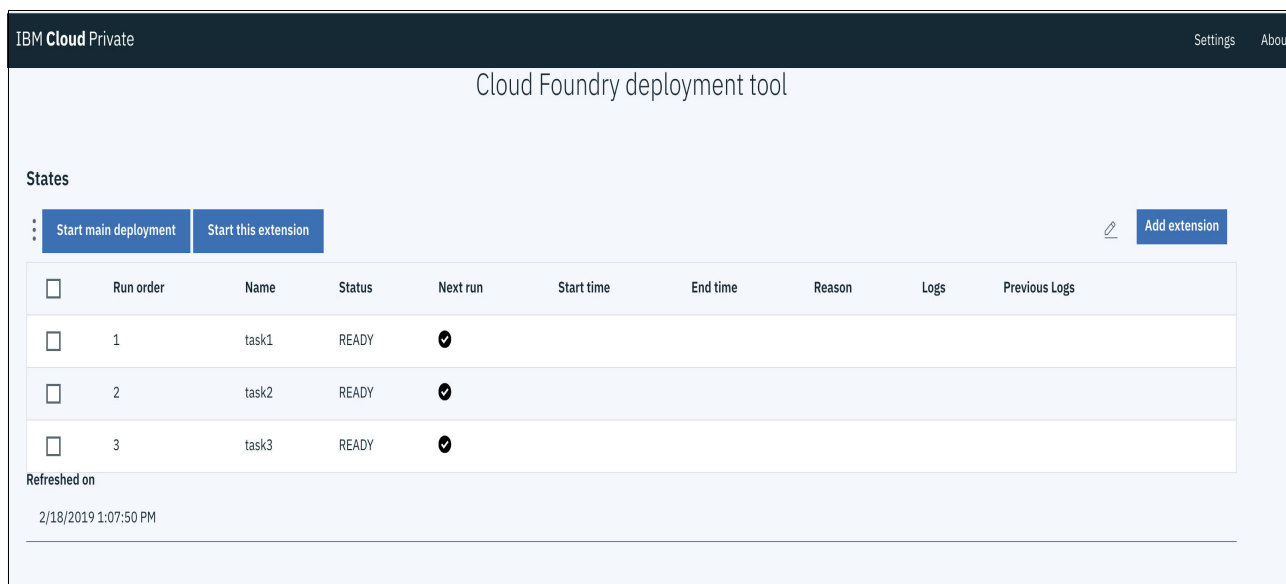


Figure 10-7 Extension states page

Figure 10-8 shows this deployment tool page.

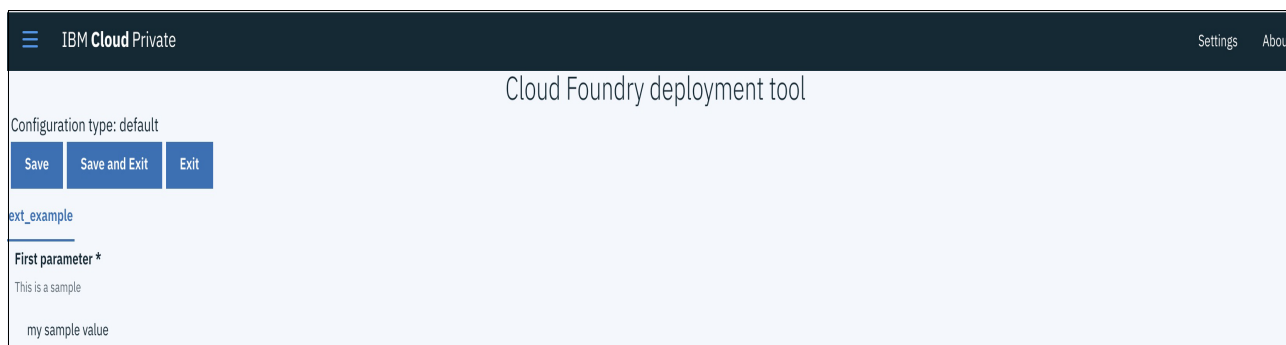


Figure 10-8 Generated user interface page

More information can be found at

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/cloud_foundry/integrating/extensions.html

10.3 High availability installation

This section describes the different options to install a Cloud Foundry high availability environment.

10.3.1 Zoning

Availability zones represent functionally independent segments of infrastructure where an issue or outage is likely to affect only a single availability zone. This means that resources in other availability zones continue to function as normal. This concept is implemented by most public IaaS providers, usually in the form of geographically distinct data centers.

The same concept can be implemented to some degree with on-premises IaaS solutions by dividing compute, storage, and networking infrastructure into groups.

Cloud Foundry supports deployment of its components into different availability zones for better high availability. It is often beneficial to have two instances of a component in separate availability zones, so that if one zone becomes unavailable, the remaining instance can keep the platform running. When three or more availability zones are configured, platform availability can usually be maintained as long as the majority of the zones remain operational.

By default, Cloud Foundry Full Stack uses two availability zones, z1 and z2, but both are configured with a single set of Cloud Provider Interface properties that you provide when configuring your Cloud Foundry installation. The effect of this is that if a single instance of a Cloud Foundry component fails you might still have some protection for components that deploy multiple instances. But if the entire availability zone experiences an outage, the Cloud Foundry platform will as well.

You can customize the z2 availability zone to use different infrastructure, thereby enhancing the high availability of your deployment, because for several of the components, multiple instances are deployed by default, and these instances are distributed over zones z1 and z2.

While availability zones cannot be configured in the main configuration file that you create for your Cloud Foundry installation, additional configuration files do provide a placeholder for a third availability zone, z3. If you want to use this third availability zone, you edit it to provide the applicable Cloud Provider Interface properties.

Next, you must modify the deployment to specify which Cloud Foundry components will be deployed over the three availability zones and to adjust the instance counts as desired. (With the exception of the number of Diego cells, which is specified in the main configuration file, the instance counts of all other Cloud Foundry components are fixed.) Instances are distributed in a round-robin fashion over the specified zones.

Further information about how to plan the number of instances of each Cloud Foundry component can be found in the article [High Availability in Cloud Foundry](#). Full instructions for customizing the availability zones for your Cloud Foundry Full Stack deployment can be found in the Knowledge Center. See [Configuring Availability Zones for IBM Cloud Private Cloud Foundry](#).

10.3.2 External database

IBM Cloud Private Cloud Foundry deployment is managed by the BOSH Director. Some components of the Director store its deployment data in several databases. Various component instances of IBM Cloud Private Cloud Foundry store their environmental and operational data in multiple databases. The BOSH Director and IBM Cloud Private Cloud Foundry are deployed with internal Postgres database servers.

It also comes with the default backup and restore configuration which utilizes internal NFS or GlusterFS servers. This might be sufficient but a highly-available external database might be a better choice in production environments for improved performance, scalability and protection against data loss.

The default BOSH Director deployment manifest needs to be modified to use external databases. The instruction is available at the following link:

<https://github.com/cloudfoundry/bosh-deployment>

We provide a custom extension that can be registered and configured prior to IBM Cloud Private Cloud Foundry installation to use external PostgreSQL or MySQL database servers. The extension is available at <https://github.com/ibm-cloud-architecture/cfp-cf-ext-db-extension>. Follow the instructions in the extension's README.

10.3.3 External objects store

Support for external object store is used to add availability, performance, and resources to the location where build packs and droplets are stored. The default deployment uses a singleton blobstore. This is sufficient for normal use, and when two Cloud Foundry deployments are created in different data centers for disaster recovery, this is sufficient for the environment. We provide an example of using S3 on AWS as a blobstore alternative. The extension can be found at <https://github.com/jnpacker/cfp-s3-objectstore-extension>.

In AWS you would create the three buckets that are recommended by the extension, update the extension .yaml file if you changed the bucket names, then follow the instructions to load the extension. Loading the extension does not transfer existing data, but will reset the blobstore, so objects like apps might need to be pushed again.

10.4 Backup and restore strategy

In a Cloud Foundry environment, there are two main types of data: first the data used to install the Cloud Foundry environment which includes the environment configuration, the deployment configuration for each component that comprises the Cloud Foundry environment, and second, the runtime data like the user account and authorization database and the deployed application configuration. In this section we discuss the different backup procedures that have to be put in place to quickly recover after a disaster.

10.4.1 Installation data

In the installation procedure it is requested that you provide a directory where installation data will be stored. It contains the environment configuration, the configuration and release assets of each extension, and also certificates and credentials auto-generated during the installation. This directory, depending of the type of installation, can be a local directory, an NFS, or a GlusterFS. For a quick recovery, it is recommended to back up this directory after each installation and upgrade.

10.4.2 Director

In a Cloud Foundry Full Stack environment, the *BOSH Backup and Restore (BBR)* is responsible for backing up the BOSH director as well as the BOSH deployment of the Cloud Foundry environment. Backups can be configured to be copied to an external customer NFS server. BBR backup can be set up by configuring values in the uiconfig.yml files to define when backups occur, where the files should be copied to, and how many backups to keep before being overwritten.

Configuration options are explained in detail at https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/cloud_foundry/installing/common_params.html.

The restore procedure allows recovery of the BOSH director and deployment should a significant error occur. The restore procedure is detailed at https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/cloud_foundry/configuring/backups.html.

10.4.3 Cloud Foundry database

The *db_nfs_copy backup* is responsible for the backup of the UAADB and CCDB. These are configured to back up hourly and if configured can be copied to a customer NFS share.

10.5 Storage and persistent volumes

In this section we discuss the storage and persistent volume options for a Cloud Foundry environment.

10.5.1 Cloud Foundry Full Stack

The traditional Cloud Foundry deployment requires both persistent and ephemeral storage allocation used by various deployed VMs. Specifically, persistent storage is needed in three categories of areas:

- ▶ **Inception Container:** The Inception container uses persistent storage mounted on “/data” of the container that it uses to store various deployment-related data. The data is used during the Cloud Foundry upgrade process, as well as for collecting deployment logs.
 - Deployment configuration data, certificates and secrets, and their backup copies. (Future versions of Cloud Foundry will provide an option to host the deployment data in a secure store such as CredentialHub as an alternative.)
 - Deployment extensions and the associated state files, scripts, and binaries
 - Deployment log files for troubleshooting
- ▶ **BOSH Director:** The Director is the orchestrator of the Cloud Foundry deployment. It needs persistent storage to store various BOSH releases and related binaries (blobstore), as well as storing the BOSH deployment state, secrets, and certificates information. Both blobstore and database information can be stored local to the Director VM on a persistent disk, or can be stored on an external persistent store via a configuration option.
- ▶ **Cloud Foundry components:** Persistent storage is needed by various components of Cloud Foundry.
 - Blobstore: An object store used for storing application binaries (can be internal or external) and buildpacks.
 - Database: Stores various databases such as Cloud Controller DB, UAA database, Locket, Consul (if present).
 - Backup-restore: Backs up deployment data for various Cloud Foundry releases
 - NFS WAL (Write-ahead logging) server: Hosts continuous backups of various databases, object stores and Director data.
 - External NFS server: It is recommended that an external NFS server is configured for taking a secondary backup to offload data from NFS WAL server for holding backups longer than one week.

Storage requirements also vary depending on whether you are deploying “developer” environment vs. “production” environment. Detailed description of storage requirements for various IaaS platforms can be found at:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/cloud_foundry/configuring/prep_cf.html.

10.5.2 Cloud Foundry Enterprise Environment (CFEE) technology preview

The Cloud Foundry Enterprise Environment (CFEE) is a technology preview at the time of this writing, and leverages storage services provided by IBM Cloud Private Kubernetes. This includes support for GlusterFS, NFS Volume service, or other storage services provided IBM Cloud Private platform. There are some differences between CFEE and Cloud Foundry Full Stack environment.

1. Inception container is deployed via the `ibm-cfee` Helm chart, it hosts containers and provides function similar to the Inception container in Cloud Foundry Full Stack environment. It needs persistent storage for storing Cloud Foundry Enterprise Environment configuration parameters, deployment extensions configuration data, and their states, and execution logs of various deployment extensions. You can configure your environment to use GlusterFS or NFS storage with inception container Helm chart.
2. Cloud Foundry Full Stack environments do not use BOSH for deployment. Instead they use IBM Cloud Private Kubernetes services for deployment orchestration of various Cloud Foundry services. Certificates and secrets are stored as Kubernetes secrets objects. The logic for various BOSH releases is built into Docker images, which are deployed as Helm charts delivered via a Cloud Foundry Enterprise Environment Cloud Pak.
3. Cloud Foundry components are deployed via Helm charts, and require persistent and ephemeral storage similar to Cloud Foundry Full stack deployment. You can configure GlusterFS or NFS storage to be used for persistent storage while deploying Cloud Foundry Enterprise Environment Helm charts.

Similar to Cloud Foundry Full Stack, Cloud Foundry Enterprise Environment supports both developer and high availability configuration modes. For detailed storage requirements for each of the modes, see

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/cloud_foundry/tech_preview/install_cfee.htm.

10.6 Sizing and licensing

The basic sizing for the management plane can be found here:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/cloud_foundry/configuring/prep_cf.html. There are also sizing for the application hosts (Diego cells). The reference size we use for these is 4x vCPU, 32 GB RAM, and 300 GB of ephemeral disk. This is the default Diego cell size for VMware and OpenStack. AWS uses a t2.xlarge which is 8x vCPU, 32 GB RAM and 300 GB of attached storage (ephemeral). Each vCPU is counted as one VPC license unit.

This means if you have 3 Diego Cells in VMware or OpenStack, you would license IBM Cloud Private Cloud Foundry for 12x VPC because you would be using 12x vCPU. Now that we understand the mappings of vCPU to VPC (1:1) and Diego cells to memory, a sizing calculation can be made. Most sizings are done based on memory. You might want 64 GB, 128 GB, or 256 GB of application memory. If you want to buy 256 GB of application memory, the VPC calculation would be as follows: $256 \text{ GB} / 32 \text{ GB} = 8$ (Diego Instances) $\times 4$ (vCPU/VPC) = 32 (vCPU/VPC).

You would want to purchase 32 VPCs of IBM Cloud Private Cloud Foundry. This would support running the required number of instances, until it hits the memory maximum. The default overcommit is 2:1, so the platform's users would have approximately 512 GB of virtual application memory available if an overcommit of 2:1 is used. 2:1 is fine for development and test, and a 1:1 is recommended in production, but this can be changed if the application profile warrants.

From a hardware perspective, the link provides a Totals table, that allows you to calculate the full hardware requirements based on management systems and Diego cells. You should also keep in mind, as the number of cells is increased, as you approach 512 GB of memory, your application profile might dictate that the management systems be horizontally scaled (routers, loggregators, or cloud controllers). This does not incur a licensing cost, but does require additional virtualization resources to host.

10.7 Networking

This section describes the network implementation for IBM CCloud Private Cloud Foundry.

The Cloud Foundry container-to-container networking feature enables application instances to communicate with each other directly. It uses an overlay network to manage communication between application instances without going through the router, thereby enabling efficient communication between applications in a controlled fashion. Each application container gets a unique IP address in the overlay network. Container networking is enabled by default in the IBM Cloud Private Cloud Foundry offering.

Cloud Foundry administrators can specify a set of network policies that enable communication between application instances. Each of the policy specifications typically includes source application, destination application, protocol (both TCP and UDP are supported) and port. The policies are dynamically applied and take effect immediately without the need to restart Cloud Foundry applications. Network policies also continue to work when applications are redeployed, scaled up or down, or placed in different containers.

By default, Cloud Foundry includes a Silk CNI plug-in for providing overlay IP address management and a VXLAN policy agent to enforce network policy for traffic between applications. These network components are designed to be swappable components in Cloud Foundry so they can be replaced with an alternate network implementation.

More details on customizing container networking in IBM Cloud Private Cloud Foundry can be found at

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/cloud_foundry/configuring/configure_container_networking.html.

10.8 Security

In this section, we describe the encryption, credentials, and certificates aspects of IBM Cloud Private Cloud Foundry.

10.8.1 TLS encryption

The Cloud Foundry deployment is encrypted throughout with TLS. Cloud Foundry uses its own encryption starting at the ingress routers. The router is TLS-encrypted to the cell, where the traffic then travels over the cell's private network to the container.

TLS is also used to guarantee routes to the application. Other traffic between cell and the Diego components as well as the Cloud Foundry management back end use TLS to remain encrypted at every step.

10.8.2 Inbound routing

Domain SSL is terminated at the router. The management as well as the application domains are hosted on the router. Certificates loaded for management and application (can be multiple) domains are specified in the installer. These certificates are loaded in the router, which allow for customer-provided wildcard certificates to be used. This means a company can use its own certificates, for a domain like `https://acme.company.com`. These certificates can be signed by the company's root certificate, or a commercial registrar.

10.8.3 Credentials and certificates

Changing credentials can be straight forward, but requires the equivalent changes to the environment of a major version upgrade. This means that your standard procedure for maintenance must be followed. This can be a notification of possible disruptions, or shifting traffic to an alternate production deployment.

This procedure can be used to either have the installer auto-generate new certificates, secrets, or tokens. It can also be used to update values generated by you or your security department. If the user or certificates are not part of the `uiconfig` file (has wildcard certificates and the admin password), then you need to locate the installer configuration directory. This is the directory that was specified when running the `launch.sh` command. Inside this directory, you will find a "CloudFoundry" directory.

This directory contains a `certificates.yml` and `credentials.yml` file. These files might be encrypted and stored elsewhere when the installer is not being used. Make sure these files are restored and editable. Open the appropriate file and find the certificate, secret, or token you want to change. If you want to let the installer generate a new value, remove the entry. Alternately you can replace the entry with your won value, making sure to meet the complexity and length described for the value.

When all of the changes have been made, make sure that the `launch.sh` has been run and the installer is available. Run `cm engine reset` and then re-run the `launch_deployment.sh` command you used to install or update the system last. If you used the Install Deployment Console, you can launch the installer there. The `cm` command sets all the installation and update states to "READY".

After the installer has completed, the new certificates, secrets, or tokens will have been applied. This works for both the director and the Cloud Foundry installation. The process can take a long time if director certificates are involved, because this might require all of the BOSH virtual machines to be rebuilt.

10.9 Monitoring and logging

This section discusses how to manage monitoring and logging in an IBM Cloud Foundry environment.

10.9.1 Monitoring

Logging is supported through internal and external options. External options include Prometheus, Splunk, and third party Cloud Foundry compatible tools. Internal monitoring is provided through a shared Prometheus and Grafana tool set. The internal monitoring is provided in IBM Cloud Private Kubernetes and is accessible by that administrator. To enable the capability, you can use Prometheus exporters, which are configured by the installer.

The exporters are provided as a configured Helm chart that can be loaded, and automatically starts tracking monitoring metrics for the Cloud Foundry deployment. This monitoring is available through Full Stack and Enterprise Environment (Kubernetes-based Cloud Foundry).

Figure 10-9 shows the flow from IBM Cloud Foundry to Exporter to Prometheus and pull from Grafana.

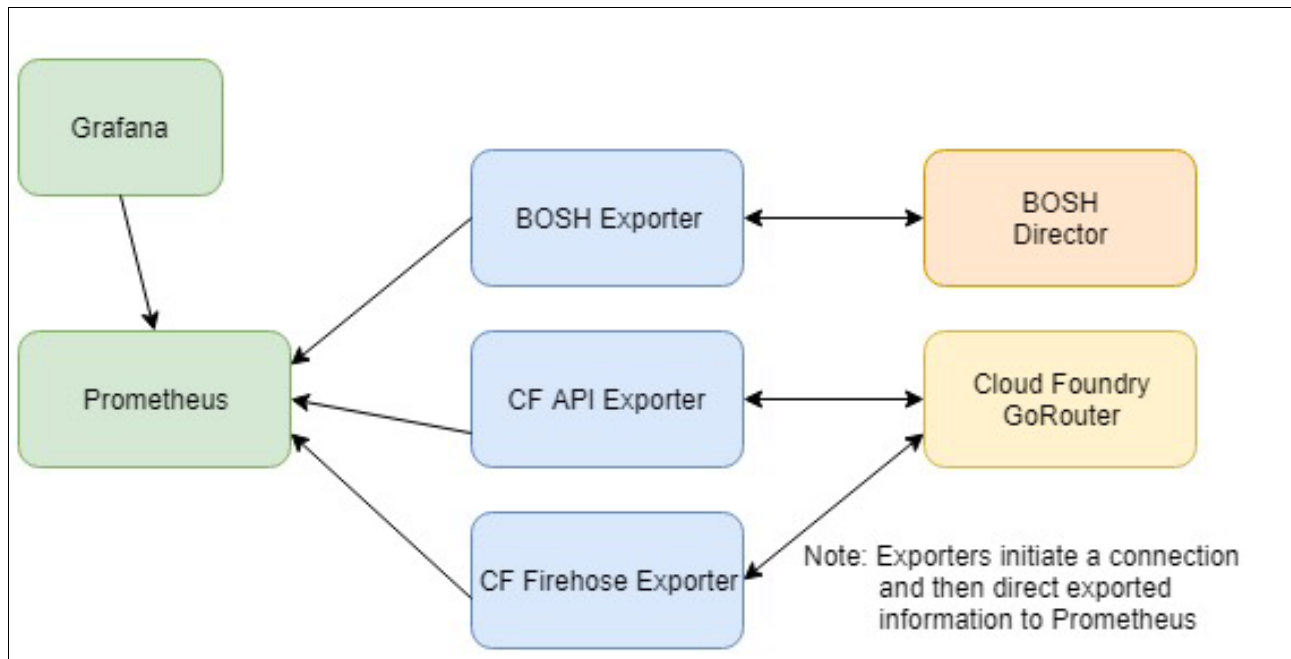


Figure 10-9 The flow from IBM Cloud Foundry to Exporter to Prometheus and pull from Grafana

When the Helm chart is loaded, you have access to BOSH, API, and Firehose metrics. In Grafana, you still need to find or define dashboards. There are a number of dashboards available in the community to choose from and you can create your own.

The following are recommended BOSH dashboards found in the files `bosh_overview` and `bosh_deployments` located at:

https://github.com/bosh-prometheus/prometheus-boshrelease/tree/master/jobs/bosh_dashboards/templates.

The following are recommended Cloud Foundry dashboards found in the files `cf_summary`, `cf_router`, and `cf_cell_summary`, located at:
https://github.com/bosh-prometheus/prometheus-boshrelease/tree/master/jobs/cloudfoundry_dashboards/templates.

10.9.2 Logging

A Cloud Foundry deployment can produce a lot of log output, both from the components that make up the Cloud Foundry platform itself and the applications you run in the environment.

Logs produced by Cloud Foundry components, which we see as syslogs, can be viewed directly on the BOSH instances (for Cloud Foundry Full Stack) or the Kubernetes pods (for Cloud Foundry Enterprise Environment). Most logs are output to a subdirectory corresponding to the job name under the `/var/vcap/sys/log` directory.

Syslogs are subject to rotation to avoid using up all the storage available to the Cloud Foundry components, so their lifespan is limited.

Logs from applications deployed in the environment, or applogs, can be accessed using the Cloud Foundry CLI or any tooling that integrates with the Cloud Foundry firehose, such as the Cloud Foundry management console. These logs are a real-time stream, with a limited amount of history available.

Cloud Foundry Full Stack provides extensions for log forwarding that allow you to forward logs to a logging system of your choice for retention and analysis. An integration with the Elastic Stack logging system in IBM Cloud Private is also provided.

The `cfp-ext-syslog-forwarder` and `cfp-ext-applog-forwarder` are embedded extensions for the Cloud Foundry deployment tool that allow you to configure forwarding of syslogs and applogs. When you add either extension, it is automatically inserted after the “Prepare Cloud Foundry” state and before the “Deploy Cloud Foundry” state. You configure the extensions by providing the details of where and how the logs should be forwarded.

Then, when the deployment is run, the extensions insert the configuration values in the required configuration files so that the “Deploy Cloud Foundry” state will configure the Cloud Foundry components to forward logs according to the configuration. For syslog forwarding, all Cloud Foundry components are modified and individually forward their logs according to the configuration. For applog forwarding, the Doppler instances (typically 4), a core component of the Cloud Foundry logging system, are modified to forward all application logs. Both extensions can be configured automatically if you are using the integration with the Elastic Stack provided in IBM Cloud Private.

Integration with Elastic Stack in IBM Cloud Private is achieved by using a Helm chart that Cloud Foundry Full Stack exports during the “Prepare Helm charts” state. After running this state of the deployment, an archive file containing the `ibm-cflogging` Helm chart and required images can be found in the `IBMCloudPrivate` subdirectory of the installation configuration directory.

The `ibm-cflogging` Helm chart provides a Logstash deployment that can be configured to accept syslogs and applogs from the `cfp-ext-syslog-forwarder` and `cfp-ext-applog-forwarder` extensions. This deployment of Logstash outputs logs to the Elasticsearch deployment provided by IBM Cloud Private, which allows you to view, search, and visualize logs in Kibana. By default, all connections are made securely using mutual TLS.

If you provide the Configuration Manager endpoint address and access token when installing the `ibm-cflogging` Helm chart, a job is run during installation that adds the chosen extensions (`cfp-ext-syslog-forwarder` or `cfp-ext-applog-forwarder` or both) to the deployment and configures them with the required IP addresses, ports, certificates, keys, and other settings. A typical setup process looks like this:

1. If not already installed, deploy the Cloud Foundry Full Stack environment, then obtain the `ibm-cflogging` Helm chart archive from the installation configuration directory.
2. Use the `cloudctl` CLI to load the Helm chart archive into IBM Cloud Private.
3. Obtain the Configuration Manager address and token from your Cloud Foundry Full Stack deployment. Create a Kubernetes secret containing the token.
4. Install the `ibm-cflogging` chart from the IBM Cloud Private catalog, configuring as desired and providing the Kubernetes secret for the Configuration Manager token.
5. In the Cloud Foundry deployment tool, launch deployment to run the newly inserted logging extensions. The “Deploy Cloud Foundry” state will be rerun with new configuration and the affected instances will be updated to forward syslogs or applogs to the Logstash deployment created by `ibm-cflogging`.

Full instructions for installing the `ibm-cflogging` chart for integration with Elastic Stack in IBM Cloud Private, and for installing the logging extensions to forward syslogs and applogs to other logging solutions, are available in the Knowledge Center. See the following articles:

- ▶ Connecting to Elastic Stack in IBM Cloud Private:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/cloud_foundry/integrating/icplogging.html
- ▶ Configuring platform system log forwarding:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/cloud_foundry/configuring/syslog_forwarding.html
- ▶ Configuring application log forwarding:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/cloud_foundry/configuring/applog_forwarding.html

10.10 Integrating external services

This section describes the integration process with external services.

10.10.1 IBM Cloud Private services

IBM Cloud Private comes with some database and messaging software Helm charts: MongoDB, PostgreSQL, MariaDB, Db2 Developer-C Edition, and RabbitMQ. The `ibm-osb-database` which comes with IBM Cloud Private Cloud Foundry implements Open Service Broker API to make these IBM Cloud Private database and messaging Helm charts available as services in IBM Cloud Private Cloud Foundry’s marketplace. That way the cloud application developers can provision instances of these services and bind applications to the service instances to use them. The lifecycle of these service instances can also be managed by the Cloud Foundry administrative console or the CLI.

The `ibm-osb-database` is installed and run on IBM Cloud Private using these high-level steps.

1. The `ibm-osb-database` IBM Cloud Private catalog archive can be found in the IBM Cloud Private Cloud Foundry installation configuration directory. The archive contains a Docker image and a Helm chart.

2. Use the IBM Cloud Private `cloudctl` CLI to load the archive into IBM Cloud Private.
3. Install the `ibm-osb-database` Helm chart.
4. Then once the `ibm-osb-database` is registered in IBM Cloud Private Cloud Foundry, the services and their service plans are available in the marketplace.

For more information, see the following articles:

- ▶ https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/cloud_foundry/configuring/osb.html
- ▶ https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/cloud_foundry/integrating/icp_db_srvc.html

10.10.2 IBM Cloud services

IBM provides a number of syndicated services. These services are running in the IBM datacenter and now are available for IBM Cloud Private Cloud Foundry, too. The administrator can decide which services are available for each organization and space. The Cloud Foundry administrator creates a Cloud Foundry service broker. Then the administrator can enable the service and the service plans that the end user can use.

More details at:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/cloud_foundry/buildpacks/service_broker.html.

10.10.3 Legacy services

The integration with legacy services can be done by following the standard Cloud Foundry process. See <https://docs.cloudfoundry.org/devguide/services/user-provided.html>.

10.11 Applications and buildpacks

This section provides information about applications and buildpack management in an airgap environment.

10.11.1 Installing extra buildpacks

The IBM Cloud Private Cloud Foundry comes with a number of IBM supported buildpacks but the environment allows authorized users to upload their own or community buildpacks and choose the order in which they are used.

The standard Cloud Foundry CLI commands are used to manage the buildpacks: **`cf create-buildpacks`**, **`cf delete-buildpacks`**. The position can be changed by using **`cf update-buildpacks`**.

For more details see the following URL:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/cloud_foundry/buildpacks/buildpacks_cloudprivate.html

10.11.2 Application for an airgap environment

There are several Cloud Foundry buildpacks that assume the environment has internet access and has the ability to access external websites and download additional prerequisite software artifacts during **cf push** operations for an application. If you have an airgap environment and do not have internet access, **cf push** operations will fail anytime something needs to be downloaded from the internet. An example of this would be a Node.js app where Node modules are typically downloaded when the app is pushed.

To avoid these node module downloads in an airgap environment, there are some additional steps to perform prior to the **cf push** of the application. These steps need to be performed on a machine that has internet connectivity. For example:

1. Download the NodeJS app from the Git repository:\
`git clone https://github.com/IBM-Bluemix/get-started-node`
2. Change to the app directory
`cd get-started-node`
3. Install all the tools we need to create a Node.js airgap buildpack using the following commands:
 - a. **apt-get update**
 - b. **apt install nodejs**
 - c. **apt install npm**
 - d. **install nodejs-legacy**
4. Force all the Node.js modules to be downloaded and packaged with the application. Run the following:
 - a. **npm install**
 - b. **npm dedupe**
 - c. **npm shrinkwrap**

Now that the Node.js application is built to work in an airgap environment, a simple **cf push** can be used to deploy the application. If you need to copy the application files to the airgap environment, create a tar file and copy it to the airgap machine. After an **untar** of the file, the **cf push** can be performed.

If you make any changes to the package.json file, rerun:

- a. **npm dedupe**
- b. **npm shrinkwrap**

10.12 iFix and releases

IBM releases iFixes and fix packs between its quarterly releases. There is no set number of interim releases, and it is usually dictated by the severity of the security fixes that become available between major releases. Major releases are delivered every quarter, with one or more iFix or fix packs delivered in between. iFixes are small deliverables, affecting only one component and are made up of a quick start guide, container (used for binary packaging), and a BOM.yml (bill of materials that includes the new container). Most of the time, there are multiple security fixes to be delivered, so a fix pack is used. This is a standalone installable fix, available on Fix Central if you have the appropriate license.

10.12.1 Zero downtime

There are a few ways to obtain zero downtime updates. One is to deploy your infrastructure using external database and blobstores for the Cloud Foundry environment. These extensions are detailed in this book. In this scenario, there are no singleton elements in the deployment, so our deployment configuration which targets only one BOSH job at a time, is able to maintain full capability of the platform.

An alternative option to using external databases and blobstores is to deploy to instances of IBM Cloud Private Cloud Foundry. This has the added disaster recovery capability of using geographically separate data centers. In this scenario, traffic is drained from one deployment while it is being updated. This allows maintenance to appear seamless to the end user. Once maintenance is complete traffic is restored to both deployments.

There is also the option of doing both. This gives you seamless updates, with no need to scale either deployment to handle the additional load while traffic is being redirected. This gives you full HA in each data center, while allowing for disaster recovery through geographically separate deployments.



Command line tools

This appendix provides introduction to Command line tools like Helm cli, kubect!, cloudctl and a cheat sheet for kubect!/cloudctl that are handy in production environment.

This appendix has the following sections:

- ▶ “Helm command line interface (helmcli)” on page 348
- ▶ “Installing the Helm CLI” on page 348
- ▶ “Using helmcli” on page 349
- ▶ “IBM Cloud Private CLI (cloudctl)” on page 350
- ▶ “Kubect!” on page 353
- ▶ “Cheat sheet for production environment” on page 361

Helm command line interface (helmcli)

The **helmcli** commands manage the application lifecycle in IBM Cloud Private cluster. Before you set up the Helm CLI, you need to perform the following steps:

1. Install the Kubernetes command line tool, kubectl, and configure access to your cluster. See Kubernetes section on page 353.
2. Install the IBM Cloud Private CLI and log in to your cluster. See IBM Cloud Private CLI section on page 350.
3. Obtain access to the boot node and the cluster administrator account, or request that someone with that access level create your certificate. If you cannot access the cluster administrator account, you need an IBM Cloud Private account that is assigned to the administrator role for a team and can access the kube-system namespace.

Installing the Helm CLI

You can install the Helm CLI by downloading it from the Helm GitHub site. Complete the following steps to install the Helm CLI:

1. Download the Helm v2.9.1 binary from helm GitHub site.
2. Make a helm-unpacked directory and unpack the installation file into that directory with the following commands:

```
mkdir helm-unpacked
tar -xvzf ./<path_to_installer> -C helm-unpacked
```

3. Change the file to an executable, then move the file to your directory:
 - a. For Linux and MacOS, run the following commands to change the permissions and move the file:

```
chmod 755 ./helm-unpacked/<unpacked_dir>/helm
sudo mv ./helm-unpacked/<unpacked_dir>/helm /usr/local/bin/helm
```

- b. For Windows, rename the downloaded file to Helm and place the file in a directory that is listed in the PATH environment variable.

4. Delete the installer and the unpacked archives:

```
rm -rf ./helm-unpacked ./<path_to_installer>.
```

Verifying the installation

Perform the following steps to verify the installation:

1. If you are using Helm 2.9.1, you must set HELM_HOME:

```
export HELM_HOME=~/.helm
```
2. Initialize your Helm CLI. Do not use the **--upgrade** flag with the **helm init** command. Adding the **--upgrade** flag replaces the server version of Helm Tiller that is installed with IBM Cloud Private.
3. For environments with Internet access, run the following command:

```
helm init --client-only
```
4. For environments that do not have Internet access, run the following command:

```
helm init --client-only --skip-refresh
```

5. Verify that the Helm CLI is initialized. Run the following command:

```
helm version --tls
```

The output should be similar to Example A-1.

Example A-1 Helm CLI is initialized

```
: Client: &version.Version{SemVer:"v2.9.1",  
GitCommit:"20adb27c7c5868466912eebdf6664e7390ebe710", GitTreeState:"clean"}  
Server: &version.Version{SemVer:"v2.9.1+icp",  
GitCommit:"843201eceab24e7102ebb87cb00d82bc973d84a7", GitTreeState:"clean"}  
:
```

Using helmcli

Follow the steps to review a list of available or installed packages:

1. Add a Helm repository. To add the Kubernetes Incubator repository, run the following command:

```
helm repo add incubator  
https://kubernetes-charts-incubator.storage.googleapis.com/
```

2. View the available charts by running the following command:

```
helm search -l
```

3. Install a chart. Run the following command:

```
helm install --name=release_name stable/chart_in_repo --tls
```

4. In the above command, `release_name` is the name for the release to be created from the chart, and `chart_in_repo` is the name of the available chart to install. For example, to install the WordPress chart, run the following command:

```
helm install --name=my-wordpress stable/wordpress --tls
```

5. List the releases by running the following command:

```
helm list --tls
```

The output should be similar to Example A-2.

Example A-2 helm list charts

NAME	REVISION	UPDATED	STATUS	CHART	NAMESPACE
my-wordpress	1	Wed Jun 28 22:15:13 2017	DEPLOYED	wordpress-0.6.5	default

6. To remove a release, run the following command:

```
helm delete release_name --purge --tls
```

7. In this command, `release_name` is the name of the release to remove. For example, to remove the WordPress release, run the following command:

```
helm delete my-wordpress --purge --tls
```

IBM Cloud Private CLI (cloudctl)

The **cloudctl** commands are executed to view information about your IBM Cloud Private cluster, manage your cluster, install Helm charts, and more.

Installing the IBM Cloud Private CLI

After IBM Cloud Private is installed, install the CLI on Windows, Linux, or macOS using the following steps:

1. Synchronize the clocks between the client computer and the nodes in the IBM Cloud Private cluster. To synchronize your clocks, use the network time protocol (NTP). For more information about setting up the NTP, see the user documentation for your operating system.

2. From the IBM Cloud Private management console, click **Menu** → **Command Line Tools** → **Cloud Private CLI** to download the installer by using a **curl** command. Copy and run the **curl** command for your operating system, then continue the installation procedure.

3. Choose the **curl** command for the applicable operating system. For example, you can run the following command for macOS:

```
curl -kLo <install_file> https://<cluster  
ip>:<port>/api/cli/cloudctl-darwin-amd64
```

4. After you run the curl command for your operating system, continue to install the IBM Cloud Private CLI.

5. To install the IBM Cloud Private CLI, run the command that matches your node architecture. <path_to_installer> is the path to the directory where you downloaded the CLI file and <install_file> is the downloaded file name. For Linux and MacOS, run the following commands to change the permissions and move the file:

```
chmod 755 <path_to_installer>/<install_file>  
sudo mv <path_to_installer>/<install_file> /usr/local/bin/cloudctl
```

6. For Windows, rename the downloaded file to cloudctl and place the file on the PATH environment variable.

7. Confirm that the IBM Cloud Private CLI is installed:

```
cloudctl --help
```

The IBM Cloud Private CLI usage is displayed.

8. Log in to your cluster:

```
cloudctl login -a https://<cluster_host_name>:8443 --skip-ssl-validation
```

Where cluster_host_name is the external host name or IP address for your master or leading master node.

9. Use the IBM Cloud Private CLI to view information about your first cluster, install more content for your cluster, or configure more clusters. If you use the IBM Cloud Private CLI to configure a new cluster, you can use the CLI to add or remove worker nodes to and from it.

10. You can configure the single sign-on by using **cloudctl** commands. More information on how to configure is documented here:

https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.0/manage_cluster/cli_sso_configure.html

General cloudctl commands

To log in to IBM Cloud Private cluster use **cloudctl login** to interactively enter the password, which is invisible, as shown in Figure A-1.

```
[root@dexter1:~# cloudctl login
API endpoint: https://9.30.250.183:8443

[Username> admin

[Password>
Authenticating...
OK
```

Figure A-1 The cloudctl login

You can follow the prompt for choosing the account and namespace or provide those options except the username and password in the command. The same action can be performed in one command:

```
cloudctl login [-a CLUSTER_URL] [-c ACCOUNT_ID or ACCOUNT_NAME] [-n namespace]
[--skip-ssl-validation]
```

Important: Providing your password as a command line option is not recommended. Your password might be visible to others and might be recorded in your shell history.

cloudctl logout will log out the session from the IBM Cloud Private cluster.

For more information, see IBM Knowledge center or use **cloudctl -help** or **cloudctl -h** as shown in Figure A-2 on page 352.

```

root@dexter1:~# cloudctl --help
NAME:
  cloudctl - A command line tool to interact with IBM Cloud Private

USAGE:
  [environment variables] cloudctl [global options] command [arguments...] [command options]

VERSION:
  3.1.0-715+e4d4eeld28cc2a588dabc0f54067841ad6c36ec9

COMMANDS:
  api      View the API endpoint and API version for the service.
  catalog  Manage catalog
  cm       Manage cluster
  config   Write default values to the config
  iam      Manage identities and access to resources
  login    Log user in.
  logout   Log user out.
  plugin   Manage plugins
  pm       Manage passwords
  target   Set or view the targeted namespace
  tokens   Display the oauth tokens for the current session. Run cloudctl login to retrieve the tokens.
  version  Check cli and api version compatibility
  help

Enter 'cloudctl help [command]' for more information about a command.

ENVIRONMENT VARIABLES:
  CLOUDCTL_COLOR=false      Do not colorize output
  CLOUDCTL_HOME=path/to/dir Path to config directory
  CLOUDCTL_TRACE=true       Print API request diagnostics to stdout
  CLOUDCTL_TRACE=path/to/trace.log Append API request diagnostics to a log file

GLOBAL OPTIONS:
  --help, -h                Show help

```

Figure A-2 The `cloudctl help` command

To manage the API keys, IDs, and the service policies of an IBM Cloud Private cluster, use the `cloudctl iam` commands. Run the following to see the list of available commands:

cloudctl iam - -help.

For more information about using the `cloudctl iam` commands visit

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.0/manage_cluster/cli_iam_commands.html.

cloudctl catalog commands

To manage the Helm charts, use the following `cloudctl catalog` commands:

- ▶ **cloudctl catalog charts:** Lists helm charts of the cluster helm repositories.
- ▶ **cloudctl catalog create-archive:** Creates a catalog archive files containing Docker images and Helm charts.
- ▶ **cloudctl catalog delete-chart - delete-helm-chart:** Deletes a Helm chart from the IBM Cloud Private internal registry.
- ▶ **cloudctl catalog load-archive - load-ppa-archive:** Load Docker images and Helm charts from a catalog archive file.
- ▶ **cloudctl catalog load-chart - load-helm-chart:** Loads a Helm chart archive to an IBM Cloud Private cluster.

- ▶ **cloudctl catalog load-images:** Loads Docker images into an IBM Cloud Private internal Docker registry.
- ▶ **cloudctl catalog repos:** Lists helm repositories.

cloud cm commands

To manage an IBM Cloud Private cluster, use the following **cloud cm** commands:

- ▶ **cloudctl cm credentials-set-openstack:** Sets the infrastructure account credentials for the OpenStack cloud provider.
- ▶ **cloudctl cm credentials-set-vmware:** Sets the infrastructure account credentials for the VMware cloud provider.
- ▶ **cloudctl cm credentials-unset:** Removes cloud provider credentials. After you remove the credentials, you cannot access the cloud provider.
- ▶ **cloudctl cm machine-type-add-openstack:** Adds an openstack machine type. A machine type determines the number of CPUs, the amount of memory, and disk space that is available to the node.
- ▶ **cloudctl cm machine-type-add-vmware:** Adds a VMware machine type. A machine type determines the number of CPUs, the amount of memory, and disk space that is available to the node.
- ▶ **cloudctl cm machine-types:** Lists available machine types. A machine type determines the number of CPUs, the amount of memory, and disk space that is available to the node.
- ▶ **cloudctl cm master-get:** Views the details about a master node.
- ▶ **cloudctl cm masters:** Lists all master nodes.
- ▶ **cloudctl cm nodes:** Lists all nodes.
- ▶ **cloudctl cm proxies:** Lists all proxy nodes.
- ▶ **cloudctl cm proxy-add:** Adds a proxy node to a cluster.
- ▶ **cloudctl cm proxy-get:** Views the details about a proxy node.
- ▶ **cloudctl cm proxy-rm:** Removes proxy nodes.
- ▶ **cloudctl cm registry-init:** Initializes cluster image registry.
- ▶ **cloudctl cm worker-add:** Adds a worker node to a cluster.
- ▶ **cloudctl cm worker-get:** Views the details about a worker node.
- ▶ **cloudctl cm worker-rm:** Removes worker nodes.
- ▶ **cloudctl cm workers:** Lists all worker nodes in an existing cluster.

Kubectl

To manage Kubernetes clusters and IBM Cloud Private you can use the **kubect1** commands. In this section we show you some of the **kubect1** commands that would help you manage your cluster and IBM Cloud Private Installation.

To see the details about how to install the **kubect1** see the Knowledge Center link:

https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_cluster/install_kubect1.html.

After installing the **kubectl** command line, you need to perform the authentication for the IBM Cloud Private cluster that you want to manage. For more information on kubectl authentication see the following link:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/manage_cluster/install_kubectl.html.

Tip: You can run **kubectl --help** to see the options for this command. See Example A-3.

Example A-3 The kubectl --help command

```
kubectl --help
kubectl get --help
kubectl describe pods--help
```

As you can see in Example A-3, it is possible to see the details specifically for each command.

The full list of options is available when you run **kubectl** as shown on Example A-4.

Example A-4 The kubectl command options

```
kubectl
kubectl controls the Kubernetes cluster manager.
```

Find more information at: <https://kubernetes.io/docs/reference/kubectl/overview/>

Basic Commands (Beginner):

create	Create a resource from a file or from stdin.
expose	Take a replication controller, service, deployment or pod and expose it as a new Kubernetes Service
run	Run a particular image on the cluster
set	Set specific features on objects

Basic Commands (Intermediate):

explain	Documentation of resources
get	Display one or many resources
edit	Edit a resource on the server
delete	Delete resources by filenames, stdin, resources and names, or by resources and label selector

Deploy Commands:

rollout	Manage the rollout of a resource
scale	Set a new size for a Deployment, ReplicaSet, Replication Controller, or Job
autoscale	Auto-scale a Deployment, ReplicaSet, or ReplicationController

Cluster Management Commands:

certificate	Modify certificate resources.
cluster-info	Display cluster info
top	Display Resource (CPU/Memory/Storage) usage.
cordon	Mark node as unschedulable
uncordon	Mark node as schedulable
drain	Drain node in preparation for maintenance
taint	Update the taints on one or more nodes

Troubleshooting and Debugging Commands:

describe	Show details of a specific resource or group of resources
logs	Print the logs for a container in a pod
attach	Attach to a running container
exec	Execute a command in a container
port-forward	Forward one or more local ports to a pod
proxy	Run a proxy to the Kubernetes API server
cp	Copy files and directories to and from containers.
auth	Inspect authorization

Advanced Commands:

apply	Apply a configuration to a resource by filename or stdin
patch	Update field(s) of a resource using strategic merge patch
replace	Replace a resource by filename or stdin
wait	Experimental: Wait for a specific condition on one or many resources.
convert	Convert config files between different API versions

Settings Commands:

label	Update the labels on a resource
annotate	Update the annotations on a resource
completion	Output shell completion code for the specified shell (bash or zsh)

Other Commands:

alpha	Commands for features in alpha
api-resources	Print the supported API resources on the server
api-versions	Print the supported API versions on the server, in the form of "group/version"
config	Modify kubeconfig files
plugin	Provides utilities for interacting with plugins.
version	Print the client and server version information

Usage:

kubectl [flags] [options]

Use "kubectl <command> --help" for more information about a given command.

Use "kubectl options" for a list of global command-line options (applies to all commands).

The full list of commands for kubectl can be found at

<https://kubernetes.io/docs/reference/kubectl/overview/>

In this section we describe the commonly used commands for server management and troubleshooting.

kubectl get

The **kubectl get** command is used to get information about the Kubernetes environment such as resources, pods and node information.

The **kubectl get** command runs against the namespace that was selected during the login. If needed it could be specified with the flag **-n <namespace>**.

kubect1 get namespace

The command **kubect1 get namespace** is used to get all namespaces from the cluster, and also to get information on whether the namespace is active or not. See Example A-5 as a sample output.

Example A-5 The kubect1 get namespace command

kubect1 get namespace
NAME STATUS AGE
cert-manager Active 13d
default Active 13d
development Active 13d
finance Active 13d
ibmcom Active 13d
istio-system Active 13d
kube-public Active 13d
kube-system Active 13d
marketing Active 13d
platform Active 13d
services Active 13d

kubect1 get nodes

This command is used to get the nodes that are part of a cluster, including their roles and status.

If you need more information, you can see the command **kubect1 get nodes -o wide**.

Example A-6 shows the output of the command **kubect1 get nodes**.

Example A-6 The kubect1 get node command

NAME	STATUS	ROLES	AGE	VERSION
10.21.9.173	Ready	etcd,master	13d	v1.12.4+icp-ee
10.21.9.176	Ready	proxy	13d	v1.12.4+icp-ee
10.21.9.177	Ready	management	13d	v1.12.4+icp-ee
10.21.9.179	Ready	va	13d	v1.12.4+icp-ee
10.21.9.180	Ready	worker	13d	v1.12.4+icp-ee
10.21.9.181	Ready	worker	13d	v1.12.4+icp-ee
10.21.9.184	Ready	worker	13d	v1.12.4+icp-ee

Optionally, if you need to get the status or information about a specific node, you can specify the name of the node in the command.

kubect1 get pods

The **kubect1 get pods** command is used to get the list of pods running in a specific namespace. By adding the flag **--all-namespaces**, you can list all pods of a specific cluster.

See the output of it at Example A-7.

Example A-7 The kubectl get pods command

```
[root@icp-312-node-1 cluster]# kubectl get pods --all-namespaces
NAMESPACE      NAME
READY   STATUS    RESTARTS   AGE
cert-manager   ibm-cert-manager-cert-manager-7c77d68c7f-sp64b
1/1      Running    0           13d
development    test-ibm-nginx-dev-nginx-8677c69574-bqk8w
1/1      Running    0           12d
kube-system    audit-logging-fluentd-ds-4xkjp
1/1      Running    0           13d
kube-system    audit-logging-fluentd-ds-bssb8
1/1      Running    0           13d
kube-system    audit-logging-fluentd-ds-h6vch
1/1      Running    0           13d
kube-system    audit-logging-fluentd-ds-j2px9
1/1      Running    0           13d
kube-system    audit-logging-fluentd-ds-jtfv4
1/1      Running    0           13d
```

This command is useful when trying to get the status of the pod. It is frequently used when troubleshooting a pod issue as described in Chapter 8, “Troubleshooting” on page 273.

kubectl logs

The **kubectl logs** command shows the logs of a resource or a pod. This command is useful when troubleshooting an application and when you need more information about it.

See Example A-8 to get the logs from a container.

Example A-8 The kubectl logs command

```
kubectl logs web-terminal-597c796cc-r5cjh -n kube-system
App listening on https://0.0.0.0:443
Created terminal with PID: 184 for user: admin
Connected to terminal 184
admin login complete with exit code 0:
Targeted account mycluster Account (id-mycluster-account)
```

Select a namespace:

1. cert-manager
2. default
3. development
4. finance
5. ibmcom
6. istio-system
7. kube-public
8. kube-system
9. marketing
10. platform
11. services

Enter a number> 1

Targeted namespace cert-manager

The **kubectl exec** command is used to run commands on pods, such as running the name resolution, or even accessing the container shell.

Example A-9 shows how to get access to the container's shell.

Example A-9 Accessing the pod shell

```
[root@icp-312-node-1 ~]# kubectl exec web-terminal-597c796cc-r5czh -n kube-system
-i -t -- bash -il
Welcome to the IBM Cloud Private Web Terminal
```

```
Type "cloudctl" to use IBM Cloud Private CLI
Run "ls -m /usr/local/bin" for a list of tools installed
```

```
root@web-terminal-597c796cc-r5czh:/usr/local/web-terminal# ls
app.js  index.html  node_modules  package.json  style.css
certs  main.js     package-lock.json  static
root@web-terminal-597c796cc-r5czh:/usr/local/web-terminal#
```

To return to the server, just type **exit**.

It is also possible to run a specific command, such as **ls -ltr /tmp**. See Example A-10.

Example A-10 Running ls -ltr /tmp

```
[root@icp-312-node-1 ~]# kubectl exec web-terminal-597c796cc-r5czh -n kube-system
-i -t -- ls -ltr /tmp
total 0
-rw----- 1 admin nogroup 0 Feb 27 22:43 sh-thd-939284318
[root@icp-312-node-1 ~]#
```

As you can see in this example, this command runs the pod as a local VM.

kubectl describe

The command **kubectl describe** is used to get information about pods, nodes, and other Kubernetes resources:

To get information on a specific node run:

```
kubectl describe nodes <node_Name>
```

To get information on a specific pod run:

```
kubectl describe pods/nginx
```

To get information on all pods run:

```
kubectl describe pods
```

To get information on pods by label (for example name=myLabel) run:

```
kubectl describe po -l name=myLabel
```

Example A-11 shows an example.

Example A-11 The describe nodes command output

```
kubectl describe nodes
Name: 10.21.9.173
Roles: etcd,master
Labels: beta.kubernetes.io/arch=amd64
        beta.kubernetes.io/os=linux
        etcd=true
        kubernetes.io/hostname=10.21.9.173
        master=true
        node-role.kubernetes.io/etcd=true
        node-role.kubernetes.io/master=true
        role=master
Annotations: node.alpha.kubernetes.io/ttl: 0
              volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp: Wed, 27 Feb 2019 12:09:04 -0800
Taints: dedicated=infra:NoSchedule
Unschedulable: false
Conditions:
  Type           Status LastHeartbeatTime             LastTransitionTime
  Reason
  ----
  OutOfDisk      False  Wed, 13 Mar 2019 14:39:25 -0700  Wed, 27 Feb 2019
12:09:04 -0800  KubeletHasSufficientDisk      kubelet has sufficient disk space
available
  MemoryPressure False  Wed, 13 Mar 2019 14:39:25 -0700  Wed, 27 Feb 2019
12:09:04 -0800  KubeletHasSufficientMemory    kubelet has sufficient memory
available
  DiskPressure   False  Wed, 13 Mar 2019 14:39:25 -0700  Wed, 27 Feb 2019
12:09:04 -0800  KubeletHasNoDiskPressure      kubelet has no disk pressure
  PIDPressure    False  Wed, 13 Mar 2019 14:39:25 -0700  Wed, 27 Feb 2019
12:09:04 -0800  KubeletHasSufficientPID        kubelet has sufficient PID available
  Ready          True   Wed, 13 Mar 2019 14:39:25 -0700  Wed, 27 Feb 2019
12:36:18 -0800  KubeletReady                   kubelet is posting ready status
Addresses:
  InternalIP: 10.21.9.173
  Hostname:   10.21.9.173
Capacity:
  cpu: 8
  ephemeral-storage: 245640Mi
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  memory: 16266504Ki
  pods: 80
Allocatable:
  cpu: 7600m
  ephemeral-storage: 243492Mi
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  memory: 15115528Ki
  pods: 80
System Info:
  Machine ID: f7bbe4af93974cbfa5c55b68c011d41c
```

```

System UUID:          C08E1133-DBDD-4345-BEB9-6BABCEE56B08
Boot ID:              e2216849-3090-4860-94da-b3c1a34e3823
Kernel Version:       3.10.0-862.14.4.el7.x86_64
OS Image:             Red Hat Enterprise Linux Server 7.4 (Maipo)
Operating System:     linux
Architecture:         amd64
Container Runtime Version: docker://18.3.1
Kubelet Version:       v1.12.4+icp-ee
Kube-Proxy Version:    v1.12.4+icp-ee
Non-terminated Pods:   (37 in total)
  Namespace           Name
CPU Requests  CPU Limits  Memory Requests  Memory Limits
-----
-----
cert-manager      ibm-cert-manager-cert-manager-7c77d68c7f-sp64b
0 (0%)          0 (0%)          0 (0%)           0 (0%)
kube-system       audit-logging-fluentd-ds-j2px9
0 (0%)          0 (0%)          0 (0%)           0 (0%)
kube-system       auth-apikeys-pjj8w
200m (2%)       1 (13%)         300Mi (2%)       1Gi (6%)
kube-system       auth-idp-5qmct
300m (3%)       3200m (42%)     768Mi (5%)       3584Mi (24%)
kube-system       auth-pap-7j8hs
150m (1%)       1200m (15%)     456Mi (3%)       1536Mi (10%)
kube-system       auth-pdp-bmgqc
600m (7%)       200m (2%)       768Mi (5%)       512Mi (3%)
kube-system       calico-kube-controllers-79ff4c4cff-lwzsk
250m (3%)       0 (0%)          100Mi (0%)       0 (0%)
kube-system       calico-node-2krkk
300m (3%)       0 (0%)          150Mi (1%)       0 (0%)
kube-system       catalog-ui-2srgr
300m (3%)       300m (3%)       300Mi (2%)       300Mi (2%)
kube-system       heapster-745c899b68-pg4zx
20m (0%)        0 (0%)          64Mi (0%)        0 (0%)
kube-system       helm-api-567f86b4f6-djtdw
Allocated resources:
(Total limits may be over 100 percent, i.e., overcommitted.)
Resource  Requests  Limits
-----
cpu        5083m (66%)  10206m (134%)
memory     6878Mi (46%) 12364Mi (83%)
Events:    <none>

```

With the previous command it is possible to see the details for the running nodes, including pods and used resources.

Cheat sheet for production environment

In the following section we list some of the commands that are commonly used in a production environment.

Use `kubectl drain` to remove a node from service

You can use `kubectl drain` to safely evict all of your pods from a node before you perform maintenance on the node (for example for kernel upgrade or hardware maintenance). Safe evictions allow the pod's containers to gracefully terminate and respect the `PodDisruptionBudgets` that you have specified:

1. First, identify the name of the node you wish to drain. You can list all of the nodes in your cluster with:

```
kubectl get nodes
```

2. Next, tell Kubernetes to drain the node:

```
kubectl drain <node name>
```

3. Once it returns without giving an error, you can power down the node (or on a cloud platform, delete the virtual machine backing the node). If you leave the node in the cluster during the maintenance operation, you need to run:

```
kubectl uncordon <node name>
```

This will instruct Kubernetes that it can resume scheduling of new pods onto the node.

Enabling autocomplete for `kubectl`

It might be difficult to remember all of the commands and their arguments for the `kubectl` command. `kubectl` provides an *autocomplete function* to help you remember the commands and their arguments. You can enable `kubectl` autocomplete in your current shell using the following command:

```
source <(kubectl completion bash)>
```

Removing a pod from a service

Let's assume we have a service backed by a number of pods. If one of the pods starts misbehaving we need to find out the root cause for the same. We need to remove the failing pod from the list of pods to investigate the problem later.

Suppose we have pod `foo` with the label `'test'`. We update the pod `foo` by removing the label named `test` if it exists with the following command:

```
$ kubectl label pods foo test-
```

As you remove the label, the pod will be out of service and deployment will spawn a new pod with the label `test`.

Editing kubernetes resources

Sometimes in a production environment you need to fine tune Kubernetes resources by editing them. You can edit the resources using `kubectl edit <resource_name>`.

Example A-12 shows changing the port number for the service production.

Example A-12 Change the port number for the service

```
kubectl edit svc/product
```

```
app: product
  name: product
  namespace: default
  resourceVersion: "2336715"
  selfLink: /api/v1/namespaces/default/services/product
  uid: de98218d-42ea-11e9-9217-0016ac1010ec
spec:
  clusterIP: 10.0.0.221
  ports:
  - name: http
    port: 8000
    protocol: TCP
    targetPort: 8000
```

Taints and tolerations

Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints. Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

You add a taint to a node using **kubectl taint**. For example:

```
kubectl taint nodes node1 key=value:NoSchedule
```

places a taint on node node1. The taint has a key, a key value, and the taint effect **NoSchedule** parameter. This means that no pod will be able to schedule onto node1 unless it has a matching toleration. To remove the taint added by the command above, you can run:

```
kubectl taint nodes node1 key:NoSchedule-
```

You specify a toleration for a pod in the PodSpec. Both of the following tolerations match the taint created by the **kubectl taint** line above, thus a pod with either toleration would be able to schedule onto node1.

Viewing and finding resources

The following lists some example commands for viewing and finding resources:

- ▶ List all services in the namespace:
kubectl get services
- ▶ List all pods in all namespaces:
kubectl get pods --all-namespaces
- ▶ List all pods in the namespace, with more details:
kubectl get pods -o wide

- ▶ List a particular deployment:
kubect1 get deployment my-dep
- ▶ List all pods in the namespace, including uninitialized ones:
kubect1 get pods --include-uninitialized

Updating resources

The following list describes some example commands for updating resources:

- ▶ Rolling update “www” containers of “frontend” deployment, updating the image:
kubect1 set image deployment/frontend www=image:v2
- ▶ Rollback to the previous deployment:
kubect1 rollout undo deployment/frontend
- ▶ Watch rolling update status of “frontend” deployment until completion:
kubect1 rollout status -w deployment/frontend
- ▶ Create a service for a replicated nginx, which serves on port 80 and connects to the containers on port 8000:
kubect1 expose rc nginx --port=80 --target-port=8000
- ▶ Update a single-container pod’s image version (tag) to v4:
kubect1 get pod mypod -o yaml | sed 's/\(image: myimage\):.*\$/\1:v4/' | kubect1 replace -f

Scaling resources

The following lists some example commands for scaling resources:

- ▶ Scale a replicaset named ‘foo’ to 3:
kubect1 scale --replicas=3 rs/foo
- ▶ Scale a resource specified in “foo.yaml” to 3:
kubect1 scale --replicas=3 -f foo.yaml
- ▶ If the deployment named mysql’s current size is 2, scale mysql to 3:
kubect1 scale --current-replicas=2 --replicas=3 deployment/mysql
- ▶ Scale multiple replication controllers:
kubect1 scale --replicas=5 rc/foo rc/bar rc/baz

Interacting with running pods

The following lists some example commands for scaling resources:

- ▶ Dump pod logs (stdout):
kubect1 logs my-pod
- ▶ Dump pod logs (stdout) for a previous instantiation of a container:
kubect1 logs my-pod --previous
- ▶ Dump pod container logs (stdout, multi-container case):
kubect1 logs my-pod -c my-container

- ▶ Dump pod container logs (stdout, multi-container case) for a previous instantiation of a container:
kubect1 logs my-pod -c my-container --previous
- ▶ Stream pod logs (stdout):
kubect1 logs -f my-pod
- ▶ Stream pod container logs (stdout, multi-container case):
kubect1 logs -f my-pod -c my-container
- ▶ Run pod as interactive shell:
kubect1 run -i --tty busybox --image=busybox -- sh
- ▶ Attach to a running container:
kubect1 attach my-pod -i
- ▶ Listen on port 5000 on the local machine and forward to port 6000 on my-pod:
kubect1 port-forward my-pod 5000:6000
- ▶ Run command in an existing pod (1 container case):
kubect1 exec my-pod -- ls /
- ▶ Run command in an existing pod (multi-container case):
kubect1 exec my-pod -c my-container -- ls /
- ▶ Show metrics for a given pod and its containers:
kubect1 top pod POD_NAME --containers

Additional kubect1 commands

For a full list of supported kubect1 commands see
<https://kubernetes.io/docs/reference/generated/kubect1/kubect1-commands>



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 associated with this book is available in softcopy on the Internet from the IBM Redbooks GitHub repository:

<https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-s-Guide.git>

Cloning the GitHub material

Perform the following steps to clone the GitHub repository for this book:

1. Download and install Git client if not installed from <https://git-scm.com/downloads>.
2. Perform the following command to clone the GitHub repository

```
git clone  
https://github.com/IBMRedbooks/SG248440-IBM-Cloud-Private-System-Administrator-  
s-Guide.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. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *IBM Cloud Private Application Developer's Guide*, SG24-8441

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:

- ▶ IBM Cloud Private v3.1.2 documentation
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/kc_welcome_containers.html
- ▶ IBM Cloud Paks Knowledge Center link:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/app_center/cloud_paks_over.html
- ▶ IBM Cloud Private for Data link:
<https://www.ibm.com/analytics/cloud-private-for-data>
- ▶ IBM Cloud Private v3.1.2 supported platforms:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.2/supported_system_config/supported_os.html
- ▶ IBM Cloud Automation Manager Knowledge Center link:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/featured_applications/cam.html
- ▶ IBM Cloud Transformation Advisor Knowledge Center link:
https://www.ibm.com/support/knowledgecenter/en/SSBS6K_3.1.1/featured_applications/transformation_advisor.html
- ▶ IBM Cloud Architecture Center:
<https://www.ibm.com/cloud/garage/architectures/private-cloud/reference-architecture>
- ▶ Downloading Terraform:
<https://www.terraform.io/downloads.html>

- ▶ Downloading the IBM plugin for Terraform:
<https://github.com/IBM-Cloud/terraform-provider-ibm/releases>
- ▶ Downloading the Git Package:
<https://git-scm.com/download/>
- ▶ Istio concepts:
<https://istio.io/docs/concepts/what-is-istio/>
- ▶ Knowledge Center link for installing Istio:
https://www.ibm.com/support/knowledgecenter/SSBS6K_3.1.2/manage_cluster/istio.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

IBM Cloud Private System Administrator's Guide

SG24-8440-00
ISBN 0738457639



(0.5" spine)
0.475" <-> 0.873"
250 <-> 459 pages



SG24-8440-00

ISBN 0738457639

Printed in U.S.A.

Get connected

