

Leveraging LinuxONE to Maximize Your Data Serving Capabilities

Lydia Parziale

Sam Amsavelu

Dileep Dixith

Gayathri Gopalakrishnan

Pravin Kedia

Manoj Srinivasan Pattabhiraman

David Simpson



Cloud

IBM LinuxONE



IBM Redbooks

Leveraging LinuxONE to Maximize Your Data Serving Capabilities

June 2024

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

Second Edition (June 2024)

This edition applies to:

- ▶ This edition applies to LinuxONE 4 LA1, IBM LinuxONE 4 LA2, IBM LinuxONE 4 AGL and LinuxONE Express.
- ▶ FUJITSU Software Enterprise Postgres Advanced Edition Annual Subscription License per IFL 13 for Linux on IBM Z and FUJITSU Software Enterprise Postgres Advanced Edition Annual Subscription License per IFL 13 Operator Bundle for Kubernetes on IBM Z

© Copyright International Business Machines Corporation 2024. 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
Trademarksx
Preface	xi
Authors	xi
Now you can become a published author, too!	xiii
Comments welcome	xiv
Stay connected to IBM Redbooks	xiv
Chapter 1. LinuxONE benefits and customer value	1
1.1 Overview of IBM LinuxONE	2
1.2 Key Technological Advancements and Features of IBM LinuxONE	3
1.3 Data serving capabilities and customer value	7
1.4 Challenges addressed by IBM LinuxONE	8
1.5 Example of the Benefits of LinuxONE in Financial Institutions	9
Chapter 2. Oracle and LinuxONE	11
2.1 Obtaining the Oracle 19c software for IBM LinuxONE	12
2.2 Red Hat Enterprise Linux (RHEL) server setup	12
2.2.1 Creating the database installation owner user	14
2.2.2 User login security and limits configuration	14
2.2.3 oracle / grid user .bash_profile	15
2.2.4 Shared memory file system	15
2.2.5 Required Red Hat Package Manager (RPM) checking	16
2.2.6 Linux large pages and Oracle databases	16
2.2.7 Disable Transparent HugePages	17
2.2.8 Tiger VNC xterm Window Interactive GUI installation (optional)	18
2.2.9 Host name file configuration	18
2.3 Configuring storage for database workloads	19
2.3.1 FCP/SCSI open storage configuration	19
2.3.2 Extended Count Key Data (ECKD) disk storage	20
2.3.3 Configuring Oracle on File System Storage	22
2.3.4 Required software directories	22
2.4 Oracle single instance installation	23
2.4.1 Installing Oracle Grid Infrastructure for a stand-alone server	23
2.4.2 Oracle database binary file installation	28
2.4.3 Creating the Oracle Database	31
2.5 Oracle Real Application Cluster (RAC) Installation	34
2.5.1 Oracle network requirements on IBM LinuxONE	35
2.5.2 Oracle SCAN configured in DNS	36
2.5.3 Public, Private and Virtual IP address requirements	37
2.5.4 Network Interface Name (NIC) consistency for Oracle RAC	37
2.5.5 Oracle RAC firewalld configuration (optional)	38
2.5.6 Time synchronization across cluster nodes	39
2.5.7 SSH user equivalency for Oracle RAC installs	40
2.6 Implementing Oracle RAC in containers	40
2.7 Database migration from other platforms to LinuxONE	41
Chapter 3. Db2 and LinuxONE	45

3.1	Benefits of migrating Db2 to LinuxONE	46
3.2	Installing Db2 on LinuxONE	46
3.2.1	Db2 install pre-requisites on LinuxONE	47
3.2.2	Db2 installation on LinuxONE	51
3.2.3	Db2 DPF install on LinuxONE	58
3.2.4	Db2 HADR on LinuxONE	67
3.2.5	Db2 install on Red Hat OpenShift on LinuxONE	67
3.2.6	Installing Db2 on Red Hat OpenShift on LinuxONE	68
3.3	Db2 migration to LinuxONE	82
3.3.1	Database migration planning	82
3.3.2	Database workloads considerations before you migrate	82
3.3.3	Database migration from Db2 on x86 to Db2 on LinuxONE	83
Chapter 4. Postgres and LinuxONE		87
4.1	Fujitsu Enterprise Postgres on IBM LinuxONE	88
4.1.1	Availability and reliability features	88
4.1.2	Application development features	94
4.1.3	Security	95
4.1.4	Performance	103
4.1.5	Performance tuning	106
4.1.6	Fujitsu developed database management software	107
4.2	Migrating to Fujitsu Enterprise Postgres	108
4.2.1	Migrating from OSS PostgreSQL to Fujitsu Enterprise Postgres	108
4.3	EnterpriseDB Postgres Advanced Server on IBM LinuxONE	109
4.3.1	Overview	109
4.3.2	EnterpriseDB Postgres with IBM LinuxONE	109
4.4	EnterpriseDB Postgres deployment scenarios	110
4.4.1	IBM LinuxONE single box per site: Asynchronous streaming	110
4.4.2	IBM LinuxONE Two Box Option - Asynchronous streaming	111
4.4.3	Deployment Option-3 - Synchronous Streaming	111
4.5	EDB Postgres installation and configuration	112
4.5.1	Red Hat Enterprise Linux environmental prerequisites	112
4.5.2	EnterpriseDB Postgres installation	113
4.5.3	Customizing and starting EDB Postgres Server	114
4.5.4	EDB Streaming Replication between primary and standby node	117
4.5.5	Enterprise Failover Manager (EFM)	118
Chapter 5. MongoDB and LinuxONE		119
5.1	MongoDB on IBM LinuxONE overview	120
5.1.1	MongoDB architecture	121
5.1.2	MongoDB high availability	123
5.1.3	MongoDB scalability	124
5.2	Installing MongoDB on LinuxONE	126
5.2.1	Install MongoDB Enterprise Edition	127
5.2.2	Running MongoDB Enterprise Edition	129
5.2.3	Why MongoDB on LinuxONE	132
5.2.4	MongoDB as a Service on LinuxONE	132
Chapter 6. Open source data base management systems and LinuxONE		135
6.1	Migration from MySQL to MongoDB	136
6.2	Migration from MySQL to PostgreSQL	136
Chapter 7. Leveraging containers		137
7.1	Containerized databases	138

7.2 Benefits of automation when using containers	139
7.2.1 Key qualities of modern database systems.	139
7.3 Oracle containers	140
7.4 Db2 containers	140
7.5 FUJITSU Enterprise Postgres containers	140
7.5.1 Automatic instance creation	142
7.5.2 Operation	146
7.5.3 Fluctuation	174
7.5.4 Next steps	178
7.6 MongoDB and containers	193
7.7 Open source databases and containers	193
Chapter 8. Database migration	195
8.1 Increasing demand for database migration.	196
8.2 Key considerations for database migration	196
8.2.1 Business continuity	199
8.2.2 Mitigating security threats	201
8.2.3 SQL performance tuning.	205
8.2.4 Optimizing database migration costs	213
8.3 Success-focused migration methodology	214
8.3.1 Experience-based migration technical knowledge	215
8.3.2 Performance tuning tips	242
8.4 Use cases: Migration from an Oracle Database system	247
8.4.1 Target system architecture	247
8.4.2 Planning and designing your migration journey	248
8.4.3 Migration scenario for large-scale legacy systems	250
8.4.4 Migration scenario for small and medium-scale systems	263
Appendix A. Application use case-geospatial data	285
8.1 Geospatial data	286
8.1.1 Introducing geospatial information systems and geospatial data	286
8.1.2 Using FUJITSU Enterprise Postgres server for geospatial data	288
8.1.3 Key PostGIS functions	288
8.1.4 Urban landscaping use case.	297
8.2 MongoDB as a Service	301
8.2.1 IBM lab environment	301
8.2.2 Deployment automation overview.	306
8.2.3 Playbooks	312
8.2.4 MongoDB deployment	323
8.2.5 Terminating	331
8.2.6 Replica set deletion.	332
Appendix B. MongoDB as a service with IBM LinuxONE	335
IBM lab environment	337
Federal Financial Institutions Examination Council Appendix J	339
Appendix J, IBM Safeguarded Copy, and data serving	340
Deployment automation overview	342
Base image deployment overview	343
Replica set virtual machine instantiation overview	343
Shadow overview	343
MongoDB deployment overview	344
Required files	344
Deployment.	345
Destroying replica sets	347

Playbooks	348
Base deployment	348
Parameters	351
Replica set virtual machine instantiation	352
OpenStack support	355
Shadow instance deployment	356
Hosts file	358
MongoDB deployment	359
Controller	359
Tasks	360
Quiescing	366
Resuming	366
Terminating	367
Controller	367
Embedded task	368
Replica set deletion	368
Master playbook	368
Image and volume tasks	369
Deploying or destroying the variables file	370
Appendix C. Converting SQL and PL/SQL to Fujitsu Enterprise Postgres SQL and PL/pgSQL	371
Challenges that are caused by the specification differences of SQL and PL/SQL	372
Case of error	372
Use case with different runtime results	373
Key to a successful SQL and PL/SQL conversion	374
SQL	375
SELECT statement	376
DELETE or TRUNCATE statements	379
ROWNUM pseudocolumn	381
Sequence	383
Conditions	385
Function	387
Others	390
PL/SQL	394
Database trigger migration pattern	394
Cursors	397
Error handling	402
Stored functions	411
Stored procedures migration pattern	415
Other migration patterns	417
Related publications	423
IBM Redbooks	423
Online resources	423
Help from IBM	423

Notices

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

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

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

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

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

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

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

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

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

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

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

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


COPYRIGHT LICENSE:

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

Trademarks

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

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

DB2®	IBM Spectrum®	Redbooks (logo)  ®
Db2®	IBM Z®	System z®
IBM®	IBM z Systems®	z Systems®
IBM Cloud®	Interconnect®	z/OS®
IBM Cloud Pak®	Passport Advantage®	z/VM®
IBM FlashSystem®	Redbooks®	zEnterprise®

The following terms are trademarks of other companies:

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

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

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

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

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Preface

Data modernization refers to initiatives or processes that lead to more pertinent and precise data, as well as quicker and more effective data processing and analysis, thereby enhancing both organizational management and regulatory compliance while catering to industrial and societal requirements..

This IBM® Redbooks® publication describes how to maximize the data serving capabilities on LinuxONE and how you can take advantage of these capabilities to modernize your enterprise.

We start by describing the value of using and migrating your databases to IBM LinuxONE in order to maximize your data serving capabilities. We describe information on migrating your current databases to LinuxONE in the following chapters:

- ▶ “Oracle and LinuxONE” on page 11
- ▶ “Db2 and LinuxONE ” on page 45
- ▶ “Postgres and LinuxONE” on page 87
- ▶ “MongoDB and LinuxONE” on page 136
- ▶ “Open source data base management systems and LinuxONE” on page 151

We also describe using containers and the benefits of automation by using containers (see section 7.2, “Benefits of automation when using containers” on page 155).

In Appendix A, “Application use case-geospatial data” on page 301, we demonstrate geospatial processing when using PostGIS and FUJITSU Enterprise Postgres on IBM LinuxONE as well as using MongoDB as a Service.

FUJITSU Enterprise Postgres on IBM LinuxONE supports PostGIS functions and on a high-performance platform with scalability and advanced security.

Appendix B, “MongoDB as a service with IBM LinuxONE” on page 351 demonstrates the benefits of using IBM LinuxONE in your enterprise. This chapter describes how IBM LinuxONE, combined with IBM Storage, provides high availability (HA), performance, and security by using a sample-anonymized client environment and includes a use case that demonstrates setting up a database as a service that can be replicated on a much larger scale across multiple client sites.

Authors

This book was produced by a working at IBM Redbooks, Poughkeepsie Center.

Lydia Parziale is a Project Leader for the IBM Redbooks team in Poughkeepsie, New York, with domestic and international experience in technology management including software development, project leadership, and strategic planning. Her areas of expertise include business development and database management technologies. Lydia is a PMI certified PMP and an IBM Certified IT Specialist with an MBA in Technology Management and has been employed by IBM for over 30 years in various technology areas.

Sam Amsavelu has worked for IBM for the past 29 years advising customers about how to build highly available, reliable, and secure infrastructure architectures for their databases. Expertise in implementing virtualization, server consolidation, and cloud technologies to help customers choose the correct platform and technologies for an optimized environment with the lowest TCO possible. SME of multiple operating systems (IBM z/OS®, IBM z/VM®, UNIX, and Linux distributions), relational databases (IBM Db2®, Oracle, PostgreSQL and SQL Server) and NoSQL databases (MongoDB).

Dileep Dixith is a Security Architect in IBM India as well as a Senior Inventor. He has 20 years of experience in Security and Storage domains. He has worked at IBM for 14 Years. His areas of expertise include Storage, Security and Confidential computing area. He is passionate about patents, blogs etc.

Gayathri Gopalakrishnan is an IT Architect for IBM India and has over 23 years of experience as a technical solution specialist, working primarily in consulting. She is a results-driven IT Architect with extensive experience in spearheading the management, design, development, implementation, and testing of solutions. A recognized leader, applying high-impact technical solutions to major business objectives with capabilities transcending boundaries. Adept at working with management to prioritize activities and achieve defined project objectives, ability to translate business requirements into technical solutions.

Pravin Kedia Pravin Kedia is a CTO for Data & AI Expert Labs running the Centre of Excellence in India. Pravin has 26+ years of experience in watsonx, CP4D Data Fabric, Data Science and DataOps Solutions, Big Data, Data Warehouse, Replication, Data Science and AI Architect with international work experience in Banking, Telecommunication, Insurance and Financial Markets domain across WW customers. He holds a degree in MBA (Finance) from University of Kansas and Bachelors (Electronics and Telecommunications) from VESIT, Mumbai university. He has written extensively on Database and Replication technologies. Pravin regularly conducts Data Strategy, Data Discovery and Design workshops with customer C-level executives. As CTO and solutions Architect for IBM Data and AI solutions, Pravin is responsible for technical Delivery for IBM Expert Lab Services Projects across the Hybrid Cloud Product portfolio. Pravin is a PMP since 2005 and is a Master Inventor (Plateau 4) and on the IDT for Data and Governance teams. Pravin is an Open Group Certified Executive IT Specialist Thought Leader and Certified ITS Profession Champion. Pravin has received numerous rewards and led/participated in AOT and Cloud Studies on Data and Governance.

Manoj Srinivasan Pattabhiraman is an Executive IT Specialist and Emerging Technology (Open Source) Enthusiast focused on helping organizations to pursue and pioneer with new transformative and disruptive technologies (like Blockchain, Machine Learning, Analytics) through collaborative strategy development, architect solutions and providing deep dive workshops. Distinguished speaker, Co-author of several advanced technology publications and a vocal advocate for open innovation and Startup culture. Above all – Avid Hiker and an aspiring Farmer - focusing on regenerative farming techniques and self-sustainable living.

David Simpson is an Oracle Certified Database Administrator on IBM LinuxONE. David has authored several IBM Redbooks publications and has presented at numerous user conferences, including Oracle OpenWorld. David has assisted many IBM customers with implementing Oracle, PostgreSQL, MongoDB, and Open-source database solutions with IBM Z® hardware and IBM Hybrid Cloud solutions.

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

Lydia Parziale
IBM Redbooks, Poughkeepsie Center

Robert Haimowitz
IBM, Poughkeepsie Center

Romney White, IBM Z: Architecture and Technology
IBM, Poughkeepsie

Vijaya Katikireddy, Senior Product Manager
IBM, San Jose

Abhishek Midha, Sudipta Sen
IBM India

Thanks to the authors of the previous editions of this book. Authors of the first edition, *Leveraging LinuxONE to Maximize Your Data Serving Capabilities*, SG24-8518, published in April, 2022 and updated May, 2022 were:

Kurt Acker, Sam Amsavelu, Elton de Souza, Gary Evans, Neale Ferguson, Aya Hoshino, Yuki Ishimori, Niki Kennedy, Riho Minagi, Daiki Mukai, Colin Page, Anand Subramanian, Kaori Suyama, Kazuhisa Tanimoto and Yoshimi Toyoshima

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

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

Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:

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

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

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

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

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



LinuxONE benefits and customer value

Many organizations are now embarking on digital transformation (DX) initiatives and are adopting continuous modernization strategies built on a Continuous Integration and Continuous Development (CI/CD) methodology. CI/CD delivers an agile, cost-effective, and lower-risk approach to modernization; reduces time to market; and accelerates return on investment (ROI) in contrast to lengthy, risky, and expensive 'big bang' rip-and-replace projects.

The versatility of open source-based technologies in accelerating containerization and microservices-based hybrid cloud journeys has led to their adoption and gained momentum and ubiquity in application modernization initiatives across all industries and organizations.

For example, when an organization's transformation strategy includes public or private cloud, the move to CI/CD, open source, and containerization is foundational because the organization can use built-in automation for agility and portability to achieve cloud-like release cycles for new competitive functions with pre-packaged scalability, security, and resilience.

This chapter focuses on the enterprise class IBM LinuxONE server, its benefits and how it can add value to your enterprise data serving needs. We describe how organizations can undertake application modernization initiatives with flexibility, agility, and cost-effectiveness without compromising critical data serving requirements for performance, scalability, resilience, or security.

In this chapter, we discuss the following:

- ▶ “Overview of IBM LinuxONE” on page 2
- ▶ “Key Technological Advancements and Features of IBM LinuxONE” on page 3
- ▶ “Data serving capabilities and customer value” on page 7
- ▶ “Challenges addressed by IBM LinuxONE” on page 8
- ▶ “Example of the Benefits of LinuxONE in Financial Institutions” on page 9

1.1 Overview of IBM LinuxONE

IBM launched LinuxONE in 2015, positioning it as a specialized mainframe system optimized specifically for the Linux operating system. The primary goal was to blend IBM's robust mainframe technology with the flexibility and openness of Linux, appealing to enterprises that required scalability, security, and reliability without compromising on the benefits of an open-source environment. Additionally, known for its energy efficiency, LinuxONE can reduce the physical footprint and power consumption compared to traditional server architectures, offering a more sustainable solution.

Over the years, IBM has continuously upgraded LinuxONE systems, integrating more advanced technology to handle growing data demands and to support emerging technologies like blockchain and artificial intelligence.

The initial models, such as the Emperor and Rockhopper, were named after penguins, reflecting their Linux-centric nature. Subsequent releases, like the LinuxONE Emperor II and III, introduced improvements in processing power, storage capabilities, and energy efficiency, making them suitable for even more intensive computing tasks.

The latest iteration of the system is the IBM LinuxONE 4, which is available in the following models and feature AI and Quantum security technologies:

- ▶ IBM LinuxONE 4 LA1

The newest member of the LinuxONE family, the IBM LinuxONE 4 was generally available in late 2022 and maintains the new form factor introduced in LinuxONE III, featuring a 19-inch frame that flexibly scales 1 - 4 frames. It is designed around the new Telum processor with 8 core per chip with Dual Chip Module packaging, 5.2 Ghz processor and is configurable with up to 200 processor cores, up to 40 TB of RAM, and 10 TB of Redundant Array of Independent Memory (RAIM) per central processing drawer.

- ▶ IBM LinuxONE 4 LA2

Released in may 2023, the IBM LinuxONE 4 LT2 is the newest entry model into the IBM LinuxONE family of servers. It delivers a 19-inch single-frame (versus the option of up to four frames for the LT1) with an efficient design with a low entry cost that can easily coexist with other platforms in a cloud data center. This model is designed around the new Telum processor with 8 core per chip with Dual Chip Module packaging. It is configurable with up to 68 cores running at 4.6 GHz, up to 16 TB of RAM, and 8 TB of RAIM per central processing drawer.

- ▶ IBM LinuxONE 4 AGL

Released in may 2023, the IBM LinuxONE 4 AGL is a new entry model option which consists of a rack mount model. It delivers a rack mount option from 10U to 39U that can be collocated with other technologies with a client-supplied 19 inch rack. This model is designed around the new Telum processor with 8 core per chip with Dual Chip Module packaging. It is configurable with up to 68 cores running at 4.6 GHz, up to 16 TB of RAM, and 8 TB of RAIM per central processing drawer.

- ▶ IBM LinuxONE Express

It is a special offering addition that is based entirely on IBM LinuxONE 4 AGL hardware but preconfigured and bundled specifically to three sizes: small, medium and large. The small size consists of 4 Cores and 384GB of RAM. The medium size consists of 6 cores and 512GB of RAM. The large size consists of 12 Cores and 736GB of RAM.

IBM LinuxONE 4 is the industry's first quantum-safe enterprise Linux system, which integrates new hardware encryption capabilities, that allows users to apply quantum-safe encryption to protect data workloads and infrastructure. Each core includes a dedicated

coprocessor for cryptographic functions, which is known as the Central Processor Assist for Cryptographic Functions (CPACF). CPACF supports pervasive encryption and is providing hardware acceleration for encryption operations.

The compression capabilities have been improved, delivering greater compression throughput than previous generation systems. This on-chip compression co-processor uses industry standard compression algorithms and can reduce data storage requirements and costs. This compression can also increase data transfer rates to boost throughput above comparable x86 CPUs; all without adversely impacting response times.

The new Telum chip is the first server-class chip with a dedicated on-chip AI accelerator that provides IBM LinuxONE with the capacity to execute real-time inferences at speed and scale by co-locating data and AI.

A significant benefit of Linux is that it is open-source. The software is unencumbered by licensing fees and its source code is freely available. Hundreds of Linux distributions are available for almost every computing platform. The following three supported LinuxONE enterprise Linux distributions¹ are available:

- ▶ [Red Hat: Red Hat Enterprise Linux \(RHEL\)](#)
- ▶ [SUSE: SUSE Linux Enterprise Server](#)
- ▶ [Canonical: Ubuntu Server](#)

These Linux distributions provide customers who use Linux with various support options, including 24 x 7 support with one-hour response time worldwide for customers running production systems. In addition to the Linux operating system, all the major Linux distributions offer a number of other open source products that they also fully support.

The increased interest and usage of Linux resulted from its rich set of features, including virtualization, security, Microsoft Windows interoperability, development tools, a growing list of independent software vendor (ISV) applications, performance, and, most importantly, its multiplatform support.

This multiplatform support allows customers to run a common operating system across all computing platforms, which means significantly lower support costs and, for Linux, no incremental license charges. It also allows customers to easily move applications to the most appropriate platform. For example, many IT organizations choose Linux because of its ability to scale databases across highly scalable hardware.

IBM supports a vibrant ecosystem around LinuxONE, encouraging developers to innovate and build applications tailored to the platform, thereby continuously enhancing its capabilities and applications.

1.2 Key Technological Advancements and Features of IBM LinuxONE

The following are some of the key technological advancements of IBM LinuxONE:

- ▶ **Blockchain Integration:** Recognizing the importance of blockchain in modern business transactions, IBM optimized LinuxONE to become an ideal environment for blockchain applications. The platform's inherent security features and high transaction throughput

¹ A Linux distribution is a complete operating system and environment. It includes compilers, file systems, and applications such as Apache (web server), SAMBA (file and print), sendmail (mail server), Tomcat (Java application server), MySQL (database), and many others.

capabilities make it perfectly suited for deploying and running distributed ledger technologies which require robust, tamper-resistant environments.

- ▶ Artificial Intelligence (AI): AI and machine learning workloads require significant computational power and data throughput capabilities. LinuxONE was enhanced to handle these needs by incorporating AI-specific accelerators and optimizing its architecture to speed up AI inferencing tasks, thus enabling real-time analytics and insights at unparalleled speeds.

Because of these technological advancements, IBM LinuxONE provides the following key features:

- ▶ Scalability
 - Flexible workload management: LinuxONE's ability to scale involves its seamless handling of varying workload sizes—from smaller databases to large-scale enterprise operations. It achieves this through a unique architecture that supports the vertical scaling of resources, allowing for the addition of CPUs, memory, and storage without disrupting ongoing operations. Valuable to businesses that need to increase their processing power or data storage capacity quickly due to growth spurts or seasonal spikes in demand.
 - Dynamic Resource Allocation: LinuxONE can dynamically allocate resources based on workload demands. This allows for real-time redistribution of computing power and storage, based on current demand and workload requirements. This ability ensures optimal performance and efficiency, particularly useful in environments where workload volumes can fluctuate significantly, such as cloud services or large e-commerce platforms.
- ▶ Performance
 - High-speed processing: LinuxONE systems are designed for high-speed data processing, utilizing advanced processor technology and optimized I/O pathways that ensure rapid data throughput. This capability makes it ideal for environments that require quick transaction processing, such as financial trading floors or real-time analytics applications.
 - Enhanced data handling: The systems are engineered to handle large volumes of data and transactions with minimal latency, supporting the demanding performance requirements of modern data-intensive applications.
 - High Reliability and Performance Consistency: LinuxONE is designed to manage varied workloads without compromising on performance or reliability. This capability is crucial for businesses that operate in sectors where time and accuracy are critical, such as financial services, healthcare, and e-commerce.
- ▶ Security
 - LinuxONE's pervasive encryption capabilities allow you to encrypt massive amounts of data with little affect on your system performance. The LinuxONE hardware benefits from encryption logic and processing on each processor chip in the system.
 - The Central Processor Assist for Cryptographic Function (CPACF) is well-suited for encrypting large amounts of data in real time because of its proximity to the processor unit. CPACF supports:
 - DES
 - TDES
 - AES-128
 - AES-256
 - SHA-1
 - SHA-2

- SHA-3
- SHAKE
- DRNG
- TRNG
- PRNG

With the LinuxONE 4, CPACF supports Elliptic Curve Cryptography clear key, improving the performance of Elliptic Curve algorithms.

The following algorithms are supported:

- EdDSA (Ed448 and Ed25519)
- ECDSA (P-256, P-384, and P-521)
- ECDH (P-256, P-384, P521, X25519, and X448)

Protected key signature creation is also supported.

- Optional cryptography accelerators provide improved performance for specialized functions:
 - Can be configured as a secure key coprocessor or for Secure Sockets Layer (SSL) acceleration.
 - Certified at FIPS 140-3 and Common Criteria EAL 4+.
- IBM's Hyper Protect Virtual Server offering is exclusive to IBM LinuxONE because it delivers more security capabilities to protect Linux workloads from internal and external threats throughout their lifecycle, build, management, and deployment phases. Some of the security benefits include:
 - Building images with integrity, which Secures continuous integration and delivery
 - Managing infrastructure with least privilege access to applications and data
 - Deploying images with trusted provenance
- IBM LinuxONE 4 maintains the Secure Execution for Linux. It is a hardware-based security technology that is designed to protect and isolate workloads on-premises, or on IBM LinuxONE and IBM Z hybrid cloud environments. Users, and even system administrators, cannot access sensitive data in Linux-based virtual environments.
- Secure Service Containers: LinuxONE utilizes secure service containers to provide a highly secure environment for running applications and workloads. These containers are isolated from one another and from the host system, protecting them from cross-container and external threats, as well as from internal vulnerabilities.
 - For organizations handling sensitive data or operating under strict regulatory frameworks, secure service containers offer a compliance-ready environment. The containers ensure that data is processed and stored in compliance with security policies and standards, significantly reducing the risk of data leakage or unauthorized access.
 - The security within these containers is managed and enforced at the hardware level, providing stronger protection than software-based security solutions. This hardware-level enforcement minimizes the risk of security being compromised through software vulnerabilities or configuration errors.
- Telum processor: Designed specifically for high-speed data processing, the Telum processor supports enhanced data throughput and reduced latency. This technological advancement is pivotal for industries where speed and efficiency are directly linked to operational success.
 - Enhanced Transaction Processing: With the Telum processor, LinuxONE excels in environments that require rapid transaction processing, such as banking and financial services. The processor's ability to handle massive volumes of

transactions in real-time ensures that businesses can deliver fast and reliable service to their customers.

- Real-Time Analytics: Beyond transaction processing, the Telum processor is also adept at facilitating real-time analytics. This capability allows businesses to analyze data as it is being collected, enabling immediate insights and decision-making, which is critical for sectors like retail and telecommunications where market conditions can change rapidly.
- ▶ Optimized data pathways
 - Efficient Data Movement: The architecture of LinuxONE is optimized to ensure efficient data movement through the processor and memory, minimizing bottlenecks that can slow down operations. This design is essential for high-performance computing environments that require fast access to large datasets.
 - Application Performance: For applications that demand real-time data access—such as financial trading platforms and online transaction systems—optimized data pathways ensure that information is processed and available without delays. This optimization supports critical business operations, enhancing the overall agility and competitiveness of enterprises.
 - Scalability and Flexibility: The optimized architecture not only supports current performance needs but also provides scalability. As business requirements grow and evolve, LinuxONE can continue to deliver optimal performance without significant reconfiguration, thereby protecting investment and future-proofing the infrastructure.
- ▶ Reliability
 - Near-Zero downtime: The architecture of LinuxONE is designed to offer unmatched reliability, capable of providing continuous service with near-zero downtime. This architecture includes:
 - Redundant processors, I/O, and memory.
 - Error correction and detection.
 - Remote Support Facility.
 - Disaster recovery: Robust disaster recovery capabilities ensure that operations can be quickly restored after any unplanned outages, further reinforcing its reliability.
- ▶ Open source support
 - Community engagement: IBM actively fosters a strong community around LinuxONE, integrating with various open-source projects and platforms. This support includes tools like Kubernetes for orchestration, and databases such as MongoDB and PostgreSQL.
 - Innovation through flexibility: The support for open-source software not only provides flexibility in terms of application development and deployment but also drives innovation as developers can leverage the best tools available without vendor lock-in.
- ▶ Energy efficiency
 - Reduced energy consumption: LinuxONE systems are designed to be energy-efficient, using less power than traditional server arrays of comparable capacity. This efficiency is achieved through optimized hardware that requires less cooling and energy. With its low power and cooling requirements, IBM LinuxONE is an ideal platform for the consolidation of distributed servers.
 - Environmental impact: The reduced energy consumption contributes to a lower carbon footprint, aligning with the sustainability goals of many organizations. This feature is particularly appealing to companies committed to environmental stewardship.

1.3 Data serving capabilities and customer value

Customer value refers to the overall worth that a product, service, or brand delivers to customers, encompassing both tangible and intangible benefits. This value is shaped by a variety of factors that are influenced by technological advancements and changing consumer expectations.

IBM LinuxONE's advanced data serving capabilities are designed to significantly enhance customer value across various critical aspects of modern business operations. From bolstering data security to ensuring system reliability and operational efficiency, LinuxONE provides comprehensive solutions that cater to the demanding needs of today's digital enterprises.

Some of the benefits of the features described in section 1.2, "Key Technological Advancements and Features of IBM LinuxONE" on page 3 translate to satisfying your data serving needs, which include the following:

- ▶ Robust data serving
 - Unified data access: Implementing a robust data management system that integrates data from various silos and makes it accessible across the organization is crucial. Such systems ensure that all departments have a unified view of customer interactions and can deliver consistent and informed customer experiences.
 - Reliability and accuracy: These systems must not only integrate data but also ensure its accuracy and timeliness, which are critical for maintaining trust and delivering value to customers.
 - Data-serving capabilities: LinuxONE is designed for structured and unstructured data consolidation and optimized for running modern relational and non-relational databases.
- ▶ Scalability
 - Handling Growth: As businesses grow and data volumes increase, the need for scalable solutions becomes critical. Scalable data architectures ensure that businesses can handle increased loads without performance degradation, thereby supporting growth without compromising customer service.
 - Flexibility for Future Expansion: Scalable solutions also provide the flexibility to expand and incorporate new technologies without extensive overhauls, thus protecting investment in existing technologies.
- ▶ Security and Data Protection
 - LinuxONE provides pervasive encryption that ensures that all data, whether at rest or in transit, is shielded from unauthorized access, thus maintaining confidentiality and integrity. LinuxONE's pervasive encryption is designed to secure data at-rest and in-transit, covering all levels of data interaction. This approach ensures that all data, regardless of its state, is shielded from unauthorized access, thereby protecting sensitive information across the entire data lifecycle.
 - As quantum computing advances, the potential threat to current cryptographic standards grows. LinuxONE addresses this emerging challenge by incorporating quantum-safe cryptography into its security architecture, ensuring that data remains protected even as quantum computing becomes more prevalent.
 - Preventing Breaches: Secure data solutions include advanced encryption, regular security audits, and real-time threat detection mechanisms to protect customer data and maintain business integrity.
- ▶ Regulatory compliance

- LinuxONE’s encryption strategy is designed not only to secure data but also simplifies compliance with global data protection regulations such as GDPR, the US Health Insurance Portability and Accountability Act (HIPAA), and the Payment Card Industry Data Security Standard (PCI DSS).
- ▶ Application modernization
 - LinuxONE plays a pivotal role in optimizing software licensing through consolidated and centralized assets. This simplified and cost-effective subscription model, when coupled with the high consolidation and scalability features of the IBM LinuxONE server, enable enterprises to realize significant and easily predictable operating cost savings over other database software solutions while continuing to meet Enterprise database requirements: automation, security, resilience, portability, and speed.

1.4 Challenges addressed by IBM LinuxONE

The following challenges faced by enterprises are addressed by LinuxOne by providing a powerful platform for data serving.

- ▶ Integration of Disparate Data Sources
 - Holistic Data Management: LinuxONE offers advanced data management capabilities that help consolidate disparate data sources into a unified system. This integration facilitates smoother workflows and data consistency across different departments, effectively breaking down data silos.
 - Enterprise-Wide Accessibility: The platform's architecture allows for easy access to integrated data across the organization, enabling departments to leverage comprehensive insights for decision-making and strategy formulation.
- ▶ Handling Real-Time Data Processing
 - High-Speed Processing: LinuxONE is designed for high-performance computing, making it exceptionally well-suited for real-time data processing needs. Its ability to handle numerous transactions quickly and efficiently is a cornerstone for businesses that require instant data processing capabilities.
 - Optimized for Peak Performance: With specialized hardware that supports extensive parallel processing, LinuxONE can manage large volumes and high velocity of data without lag, ensuring that real-time analytics and interactions are handled without delays.
- ▶ Scalable and Flexible Infrastructure
 - Dynamic Scalability: As business needs grow and data volumes increase, LinuxONE's scalable nature allows for seamless expansion without the need for frequent hardware upgrades or reconfigurations. This scalability ensures that businesses can continue to expand their data serving capabilities as needed with minimal disruption.
 - Flexible Configuration: The platform supports various configurations and can be tailored to specific business needs, whether on-premises or as part of a hybrid cloud environment. This flexibility allows businesses to optimize their IT infrastructure according to their operational requirements and budget constraints.

1.5 Example of the Benefits of LinuxONE in Financial Institutions

LinuxONE has been designed to help the world's most complex financial organizations quickly grow their enterprises to outthink their competition. LinuxONE can assist in delivering agility and efficiency through the use of the cloud, transact faster when using blockchain, create an outstanding customer experience through analytics, and ensure service and data protection through one of the world's most secure systems.

Encryption capabilities of LinuxONE can help defend and protect business critical data against external and internal attacks, including privileged users. Financial institutions can protect all business critical data at rest and in-flight – transparently with no changes to applications with IBM Pervasive Encryption. Centralized data encryption policy-based controls significantly reduce the costs associated with data security and achieving compliance mandates, including new General Data Protection Regulations (GDPR). Integrated encryption, data protection, identity and access management and security intelligence and audit on LinuxONE provide financial institutions with a highly optimized, cost effective security environment.

To read more on how LinuxONE helps the world's most complex financial organizations quickly grow their enterprises to outthink their competition, see:
http://tes-es.com/wp-content/uploads/2019/12/IBM_LinuxONE_Banking_FM_Point_of_View.pdf

To read about how a bank in Jamaica accelerated its core process and positioned itself to launch new services with LinuxONE technology, see the following case study:
<https://www.ibm.com/case-studies/sagicor-bank-jamaica>



Oracle and LinuxONE

This chapter covers the pre-installation steps that are required for setting up Oracle 19c with the newly certified Red Hat Enterprise Linux Release 8 (RHEL 8) on an IBM LinuxONE. We include information about how to obtain Oracle documentation, code, and My Oracle Support (MOS) Notes.

For product longevity and patching, IBM and Oracle recommend upgrading to Oracle 19c, which is the long-term release with the longest support end date. Oracle 23ai is the next projected long-term release for the LinuxONE (s390x) platform.

Although the minimum RHEL 8 Oracle Database release level is 19.11, we also recommend using the latest level of RHEL 8 (in our lab environment we use release 8.10 for Oracle 19c installs).

The Oracle Master Note is intended to provide an index and references to the most frequently used MOS articles concerning Linux OS requirements for an Oracle Database software installation.

For more information about the master note, see MOS Doc ID 851598.1, which is available by logging in to your [Oracle Support](#) account

2.1 Obtaining the Oracle 19c software for IBM LinuxONE

Depending on your Oracle licensing agreement, the latest Oracle code can be downloaded from the [Oracle Software Delivery Cloud](#) or from the [Oracle Technology Network \(OTN\)](#) developers download site.

If you have a commercial Oracle license, always download your software from the [Oracle Software Delivery Cloud](#).

For trial and developer network downloads, the Oracle License agreement must be fully understood before agreeing and downloading the Oracle 19c database software from the OTN download site.

In addition to the base software, the latest Patch Set or Release Update must be downloaded from Oracle Support.

Oracle 19c provides the latest Release Update (RU) for the Grid Infrastructure and Database from the Oracle support site. The latest RU for your release can be found in Oracle Note Master Note for Database Proactive Patch Program (Doc I 756671.1).

An Oracle 19c base installation is release 19.3 for Linux on System Z. It is recommended to be on the latest Grid Infrastructure and Database RU when the patches are made available every fiscal quarter.

The latest (as of July 2024) Oracle 19.24 Grid Infrastructure and Database security and RU patch is [36582629](#) for Grid/ASM (including DB) or [36582781](#) for just the DB RU. Follow the ReadMe file for the patch you plan to install. Make sure to download the latest OPatch installer utility patch [6880880](#) as well for the RU patch as described in the patch readme file.

2.2 Red Hat Enterprise Linux (RHEL) server setup

In this section we describe the minimum requirements need to install Oracle on RHEL 8 for LinuxONE.

Server minimum requirements

The following are the minimum requirements for the RHEL server.

- ▶ Disk Space
 - Enterprise Edition database: A total of 10 GB of disk space for the Oracle software, depending on the installation type. You might need extra space for patching logs.
 - Grid Infrastructure: A total of 20GB is minimally required for installing Oracle ASM single instance (restart) or an Oracle RAC install. It is also recommended to provide more space for Oracle grid infrastructure log files.
- ▶ Recommended memory
 - A minimum of 8 GB of virtual memory is recommended in the Linux guest for Oracle 19c installations of Oracle Database, including the Grid Infrastructure requirements.
 - For single instance file system-based installations, 4 GB minimum virtual memory is recommended.
- ▶ Swap
 - Swap disk space is proportional to the system's physical memory (as shown in Table 2-1) and recommended by Oracle for general Linux swap.

Table 2-1 Oracle Swap size recommendation

RAM	Swap space
1 - 8 GB	Twice the size of RAM
8 - 32 GB	Equal to the size of RAM
More than 32 GB	32 GB

The Oracle guidelines for Linux swap can be reduced (if needed) when enough memory is available to run all the databases in the Linux guest. The Oracle Installer requires a minimum 500 MB of configured Linux swap to complete an installation and 1GB of swap for database upgrades and patches.

Customers that use IBM LinuxONE can use a layered virtual in-memory disk or VDisk for the Linux swap devices. Linux swap to a memory device (VDisk) is much quicker than using a physical disk storage device.

Figure 2-1 shows an example of a recommended VDisk configuration with a VDisk being used for the first and second Level Swap with higher priority. Then, a physical disk or DASD disk can be used as a lower priority swap in the case of unexpected memory usage. Linux use, the higher priority swap devices first. When the swap device is fully exhausted, the next priority swap device is used next.

```
# swapon -s
```

Filename	Type	Size	Used	Priority
/dev/dasdo1	partition	131000	0	10
/dev/dasdp1	partition	524216	0	5
/dev/mapper/u603_swap3	partition	6291448	0	1

Figure 2-1 Swap VDisk Configuration Priorities

► Temporary disk space required for installation

A minimum of 1 GB of disk space in the /tmp directory on LinuxONE systems is required. If you do not have enough space in /tmp for an Oracle installation, you can set the TMP and TMPDIR environment variables to another directory with space by setting these variables in oracle user's environment.

► Linux kernel parameters

Verify that the required operating system kernel parameters are installed.

An example of the required Oracle kernel parameters is shown in Table 3-2 for a system with an 8 GB Oracle System Global Area (SGA) database.

If the current value of any parameter is higher than the value listed in Table 3-2, we suggest not to change that value of that parameter. These parameters must be configured in the Linux /etc/sysctl.conf file.

Example 2-1 shows a sample /etc/sysctl.conf file for an 8GB Oracle database.

Note: Pay special attention to the vm.nr_hugepages value which is measured in 1 MB pages, over allocating this value can render the Linux unbootable by allocating all the memory into large page memory.

Example 2-1 Sample Oracle parameters

```
# Oracle specific parameters
fs.file-max = 6815744
fs.aio-max-nr = 1048576
#set shmmax to largest sga memory segment in bytes
```

```

kernel.shmmax = 8589934592
#set shmall to (sum of sga memory in bytes)/4096
kernel.shmall = 2097152
kernel.sem = 250 32000 100 128
kernel.shmmni = 4096
kernel.panic_on_oops = 1
net.ipv4.ip_local_port_range = 9000 65500
net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048576
vm.swappiness = 1
#set nr_hugepages to sum of all Oracle SGAs in MB if using large pages
#vm.nr_hugepages = 4096
vm.hugetlb_shm_group=1001

```

After kernel parameter values are updated, validate the kernel parameters by using the following `sysctl` command:

```
/sbin/sysctl -p
```

2.2.1 Creating the database installation owner user

The installation process requires at least an Oracle Database installation owner (**oracle**), an Oracle Inventory group (**oinstall**), and an Oracle administrative privileges group (**dba**).

Issue the following commands to create the oracle group and user ids.

```

groupadd -g 1001 oinstall ; groupadd -g 1002 dba ; groupadd -g 1003 asmdba ;
groupadd -g 1004 asmoper ; groupadd -g 1005 oper ;
useradd -m -g oinstall -G dba oracle echo "oracle:newPassw0rd" | chpasswd

```

For separation of duty, some sites set up a Linux grid user ID to manage the grid infrastructure - both the Oracle ASM and Oracle Grid Infrastructure components. If this is the case, use the following `useradd` and `change password` command:

```
useradd -m -g oinstall -G dba grid echo "grid:newPassw0rd" | chpasswd
```

2.2.2 User login security and limits configuration

To avoid operational system resource issues when using the Oracle Database, the Linux `ulimit` values must be adjusted to allow for the suitable operation of an Oracle Database.

The settings shown in Example 2-2 must be verified before an installation is performed. These settings can be found in the `/etc/security/limits.conf` file.

If the Linux *oracle* user performs the installation, add the settings that are shown in the example below to your `/etc/security/limits.conf` file. If your current values are higher than these values, leave the higher value in place.

If using a grid user, include the grid user settings as well. For large systems with many users, increasing the `nproc` value to 131072 is typical.

Example 2-2 Oracle user limits

```

oracle soft nproc 2047
oracle hard nproc 16384

```

```

oracle soft  nofile 1024
oracle hard  nofile 65536
oracle soft  stack 32768
oracle hard  stack 32768
# memlock for Huge Pages support
oracle soft  memlock unlimited
oracle hard  memlock unlimited

```

2.2.3 oracle / grid user .bash_profile

The ulimit values take effect upon logging into a Linux guest by setting the ulimit values in each Linux users .bash_profile file. You can also update the system base /etc/profile for the system for all Linux users.

The values that are shown in Example 2-3 can be added to the oracle and grid (if used) users .bash_profile file.

Example 2-3 .bash_profile oracle user limits settings

```

ulimit -u 16384
ulimit -n 65536
ulimit -s 32768

```

If installing Oracle 19c on RHEL 8 or greater, make sure the following environment variable is set in your .bash_profile or is set manually.

```
export CV_ASSUME_DISTID=RHEL8.0
```

If using the graphical user interface (GUI) when running the Oracle Universal Installer (OUI), Database Configuration Assistant (DBCA) or any related tasks, make sure the environment variable is set in the xterm window as well.

No environment variable **LD_ASSUME_KERNEL** value should be used with the Oracle 19c product.

2.2.4 Shared memory file system

For Oracle Database Installations that do not use Linux large pages but are using Automatic Memory Management (AMM) with the MEMORY_TARGET parameter, it is important to ensure that the /dev/shm mount area is of type tmpfs and is mounted with the following options:

- ▶ With RW (read/write) and execute permissions set on it
- ▶ With noexec or nosuid not set on it

Use the following command to check the shared memory file system:

```
cat /etc/fstab |grep tmpfs
```

If needed, change the mount settings. As the root user, open the /etc/fstab file with a text editor and modify the tmpfs line. If you are planning to use AMM, set the tmpfs file system size to the sum of all the MEMORY_TARGET on the system, as shown in the following example:

```
tmpfs /dev/shm tmpfs rw,exec,size=30G 0 0
```

LinuxONE can use large pages of 1 M with Oracle running under z/VM and 2GB when running under LPAR mode.

2.2.5 Required Red Hat Package Manager (RPM) checking

The following provides the pre-installation Linux steps that are required to set up Oracle RDBMS 19c release on IBM LinuxONE (s390x). This section also describes more pre-installation checking that you must perform after you ensure that a GUI is established.

- ▶ Operating system requirements for IBM LinuxONE

RHEL 8 servers must be running at least Red Hat kernel (8.3) 4.18.0-240.el8.s390x (s390x) or higher to Install Oracle RDBMS 19c on IBM LinuxONE. We recommend that latest Red Hat 8.x Support pack be applied as well for security and bug fixes.

- ▶ Oracle RPM Checker with Red Hat Enterprise Linux Server

Use the Oracle RPM Checker utility to verify that the required Linux packages are installed on the operating system before starting the Oracle Database or Oracle Grid Infrastructure installation.

See the Oracle support note, Doc ID [2553465.1](#), to download the latest RPM checker to ensure your environment is ready when installing and implementing Oracle 19c. From this support note, you will download the Linux RPM that will check for other required RPM packages that are needed on your system for a successful installation. The downloaded file from the support note is RH8-DB-19c.s390x.rpm.

After you have downloaded the suitable Red Hat Package Manager checker, upload it to your system and install it as root. Use the following Red Hat YUM command to install the required Linux packages and their dependencies.

```
# yum install RH8-DB-19c.s390x.rpm
```

Running the YUM command automatically pulls in the required RPMs required for Oracle 19c on LinuxONE.

2.2.6 Linux large pages and Oracle databases

It is recommended for performance and availability reasons to implement Linux large pages for Oracle databases that are running on IBM LinuxONE systems. Linux large pages are especially beneficial for systems in which the database's Oracle System Global Area (SGA) is greater than 8 GB or if more than 50 database connections are needed.

Complete the following steps to implement large pages:

1. Use Automatic Shared Memory Management (ASMM).

Each database that is planned for Linux large pages cannot use Automatic Memory Management (AMM) by setting the MEMORY_MANAGEMENT parameter. It is recommended to use Automatic Shared Memory Management (ASMM) by setting the SGA_TARGET and PGA_AGGREGATE_TARGET Oracle parameters.

2. The Oracle use_large_pages parameter can be set to *true*, *false*, or *only*. If you have a LinuxONE system with one large database that requires large pages, and other smaller databases that do not require large pages, you can set the larger SGA database to *only* and the smaller databases to *false*. The default setting for use_large_pages is *true*.

3. Set or update the following parameters in the /etc/sysctl.conf kernel parameters, with the vm.nr_hugepages value being the size of the Oracle SGA in 1 MB pages rounded up to the nearest granule size of the databases SGA sizes. This nr_hugepage value is the sum

of all the Oracle Database SGAs on the Linux guest requiring large pages, as shown in the following equations:

```
vm.nr_hugepages = ((sum of all large page SGA's)* 1024)+16 (Granule)= N
vm.hugetlb_shm_group = <Oracle user Linux group number from /etc/group>
```

Note: Take care when checking that these kernel parameters are correct. A mis-configured `vm.nr_hugepage` kernel parameter setting can cause a Linux system to start incorrectly because of a lack of memory.

- At the Linux level, it is recommended to set the `/etc/security/limits.conf` file's `memlock` ulimit parameters to unlimited or the maximum size of the Oracle SGA needed. The Linux kernel parameter (`nr_hugepages`), Oracle SGA values, and the ulimits must be synchronized. By setting the ulimit to unlimited, one less change is needed if the Oracle SGA or Linux large page values must be increased, as shown in Example 2-4.

Example 2-4 Set memlock ulimits

```
cat /etc/security/limits.conf | grep memlock | grep -v grep
oracle soft memlock unlimited
oracle hard memlock unlimited
```

- Restart your Linux image and Oracle for the changes to take effect. Review your Oracle Database alert log to verify that the database was started with large pages enabled.

2.2.7 Disable Transparent HugePages

Transparent Huge Pages are enabled by default in RHEL. It is recommended to disable Transparent Huge Pages for the best performance and stability. Transparent HugePages are different from Linux large pages, which are still highly recommended to use. You can check whether your system features Transparent HugePages enabled by using the following command:

```
# cat /sys/kernel/mm/transparent_hugepage/enabled [always] madvise never
```

To disable Transparent HugePages, it is recommended to update the LinuxONE grub file by using `transparent_hugepage=never` at the end of the parameters line. Then, run the command `zipl -VV` for the changes to take effect with the next restart of your system.

The following are the steps to disable Transparent Huge Pages:

- Use the `grubby` utility to add `transparent_hugepage=never` to the end of the kernel parameters line:


```
grubby --update-kernel=ALL --args="transparent_hugepage=never"
```
- Run the `zipl -VV` command to activate the Linux kernel parameters for next system restart:


```
# zipl -VV
```
- Restart your Linux guest and verify transparent huge pages are disabled:


```
shutdown -r now
```
- Upon restart, verify that transparent HugePages are disabled by using the following command:


```
# cat /sys/kernel/mm/transparent_hugepage/enabled always madvise [never]
```

2.2.8 Tiger VNC xterm Window Interactive GUI installation (optional)

Install or configure the VNC viewer or xterm emulator if you want to perform GUI-based Oracle installations with runInstaller, asmca, dbca, or netca Oracle GUI installer programs.

Silent install options are available for these Oracle installers as well. However, first-time installations include GUI wizards that provide the system checks to verify your system configuration and then can be scripted with the silent installations later.

For Red Hat systems, the VNC Configurator summarizes the steps to install the VNC server and client RPMs, and then to configure your VNC xstartup file based on the xterm emulator of your choosing. Use the following command to install the VNC server:

```
# yum install tigervnc-server
```

To install the GNU Desktop for VNC server you can use the following command.

```
# yum group install GNOME base-x Fonts
```

To start a VNC server session as the Oracle user, run the `vncserver` command. You may be required to enter a password for the `vncviewer` session. Take note of the port (in Figure 2-2 it is :1) that is used. This port is needed when connecting with the VNC Viewer to the `vncserver` session that was started.

```
[oracle@zdbl08 ~]$ vncserver
WARNING: vncserver has been replaced by a systemd unit and is now considered deprecated and removed in upstream.
Please read /usr/share/doc/tigervnc/HOWTO.md for more information.

You will require a password to access your desktops.

Password:
Verify:
Would you like to enter a view-only password (y/n)? n
A view-only password is not used
xauth: file /home/oracle/.Xauthority does not exist
New 'zdbl08:1' (oracle)' desktop is zdbl08:1

Creating default startup script /home/oracle/.vnc/xstartup
Creating default config /home/oracle/.vnc/config
Starting applications specified in /home/oracle/.vnc/xstartup
Log file is /home/oracle/.vnc/zdbl08:1.log
T
```

Figure 2-2 VncServer Configuration

If you have problems connecting with the client verify your Linux firewall settings in section 2.5.5, “Oracle RAC firewall configuration (optional)” on page 38.

2.2.9 Host name file configuration

Make sure the Fully Qualified Domain Name (FQDN) is in the `/etc/hosts` file. A sample of our file is shown in Example 2-5. A FQDN is required for Oracle SQL*Net and RAC network communications.

Example 2-5 Edit the `/etc/hosts` file

```
cat /etc/hosts
9.76.60.121 oracle2.cp0lab.ibm.com oracle2
```

```
9.76.60.131 oracle3.cpolab.ibm.com oracle3
```

If configuring an Oracle RAC environment, it is recommended to include the variable IP addresses (VIP) in the hosts file as well on the same subnet as the public network interface. A sample of our file is shown in Example 2-6.

Example 2-6 Include vip addresses

```
9.76.60.122 oracle2-vip.cpolab.ibm.com oracle2-vip
9.76.60.123 oracle3-vip.cpolab.ibm.com oracle3-vip
```

2.3 Configuring storage for database workloads

In this section, we describe the following disk storage configuration options that are available for database files on IBM LinuxONE:

- ▶ Open System SAN Storage FCP/SCSI LUNs
- ▶ IBM Extended Count Key Data (ECKD) DASD with HyperPAV or PAV.
- ▶ File systems and/or Shared Storage (for example, Spectrum Scale GPFS)

To configure disk storage for LinuxONE, assign the logical unit numbers (LUNs) with multiple paths to Linux to ensure high availability and better performance. This spread the I/O workload across more channel paths/host bus adapters (HBAs) and is designed to improve performance and high availability.

2.3.1 FCP/SCSI open storage configuration

If using FCP/SCSI open system storage for your ASM disk devices, it is important to configure a multipathing and a UDEV rule to ensure device persistence and disk permissions for the Oracle ASM disks.

To configure multiple disk paths with FCP/SCSI open storage, it is necessary to set up a `/etc/multipath.conf` file to enable multipathing for high availability to the disk paths and for performance reasons.

It is recommended to use the `user_friendly_names` parameter in the `multipath.conf` so that the disk permissions for the Oracle Database ASM LUNs can be set correctly. Example 2-7 shows a sample `multipath.conf` file.

Note: It is also recommended to consult your storage vendor for their recommendations for the `multipath.conf` file settings for the Linux distribution and level that you use.

Example 2-7 A sample `/etc/multipath.conf` file for FCP/SCSI disk storage

```
defaults {
    dev_loss_tmo infinity
    fast_io_fail_tmo 5
    user_friendly_names yes
}
blacklist {
# devnode ".*"
}
multipaths {
```

```

multipath {
  wwid 20020c240001221a8
  aliasASMDISK00
}

```

To restart the multipath service for any changes to take effect, run the following command to read in the new multipath.conf settings:

```

/sbin/multipath -v2
systemctl restart multipathd.service

```

The `/etc/udev/rules.d/12-dm-permissions.rules` file also must be configured to set the `/dev/mapper/` user friendly disk names to the correct device permissions for the oracle or grid Linux user ID that owns the disk storage devices.

Example 2-8 shows that any disks that are configured with a `user_friendly_name` that begins with "ASM*" are owned by the grid or oracle Linux user.

Example 2-8 12-dm-permissions.rules file for FCP/SCSI open system disk storage

```

ENV{DM_NAME}=="ASM*",OWNER=="oracle",GROUP=="dba",MODE=="660"

```

You can get a sample `/etc/udev/rules.d/12-dm-permissions.rules` template file from the following location of your installation:

```

/usr/share/doc/device-mapper-<your kernel level>/12-dm-permissions.rules

```

for Red Hat systems.

You can run the following Linux commands to enable any changes to the UDEV rule:

```

udevadm control --reload-rules; udevadm trigger

```

Verify the disk permissions are set correctly by using the following `ls` command:

```

ls -lL ASM*

```

The following is a sample of the output from that command:

```

brw-rw---- 1 oracle dba 253, 14 Jun 24 08:30 ASMDISK00

```

2.3.2 Extended Count Key Data (ECKD) disk storage

With ECKD/DASD storage, it is recommended to set up disk path aliases with Hyper Parallel Access Volumes (HyperPAV). Parallel Access Volumes (PAV) is the concept of using multiple devices or aliases to address a single ECKD disk device. While the usage of PAV devices requires the usage of the multipath daemon and its configuration, the advantage of HyperPAV is that it only needs to be enabled on Linux, everything else is handled by the kernel, and is transparent to the user.

HyperPAV provides more I/O paths to help avoid any disk I/O path bottlenecks.

If using ECKD/DASD storage, consider the following points:

- ▶ To configure the correct disk permissions for the Oracle ASM disk user to write and read from the ASM devices, a UDEV rule is needed to [assign](#) the correct disk permissions.
- ▶ You must always create a partition on an ECKD DASD device.

- To configure the ECKD/DASD devices, it is helpful to use the lsdasd device name in the disk name to make it easier to assign the disk storage, as shown in Example 2-9.

Example 2-9 List DASD devices

```
# lsdasd
Bus-ID  Status  Name      Device Type  BlkSz Size  Blocks
=====
0.0.0200 active dasda 94:0  ECKD      4096 7043MB 1803060
0.0.7406 active dasdp 94:60 ECKD      4096 7043MB 1803060
0.0.7407 active dasds 94:72 ECKD      4096 7043MB 1803060
```

It is also recommended to configure aliases for each DASD device containing Oracle datafiles, to allow for more concurrent IO. HyperPAV can be setup in z/VM or in LPAR to configure aliases for the base ECKD volumes. The “lsdasd -u” command can be used to verify that aliases are configured correctly. If you see no “alias” defined, then the configuration is not setup correctly.

Run the “lsdasd -u” command as shown in Figure 2-3 to ensure PAV aliases are defined for any ECKD Disk storage devices being used for database files for IO performance.

```
[root@hsoral ~]# lsdasd -u
Bus-ID  Name      UID
=====
0.0.0200 dasda      IBM.750000000ANV21.1551.00.000000000000eac500000000000000000
0.0.0201 dasdb      IBM.750000000ANV21.1551.01.000000000000eac500000000000000000
0.0.0202 dasdc      IBM.750000000ANV21.1551.02.000000000000eac500000000000000000
0.0.0203 dasdd      IBM.750000000ANV21.1551.03.000000010000eac500000000000000000
0.0.0204 dasde      IBM.750000000ANV21.1551.04.000000010000eac500000000000000000
0.0.0205 dasdf      IBM.750000000ANV21.1551.05.000000010000eac500000000000000000
0.0.0100 alias      IBM.750000000ANV21.1551.xx.0000000000000000000000000000000000
0.0.0101 alias      IBM.750000000ANV21.1551.xx.0000000000000000000000000000000000
0.0.0102 alias      IBM.750000000ANV21.1551.xx.0000000000000000000000000000000000
0.0.0103 alias      IBM.750000000ANV21.1551.xx.0000000000000000000000000000000000
0.0.0104 alias      IBM.750000000ANV21.1551.xx.0000000000000000000000000000000000
0.0.0105 alias      IBM.750000000ANV21.1551.xx.0000000000000000000000000000000000
```

Figure 2-3 lsdasd with aliases

If configuring Oracle RAC, the same disk devices must be set up as shareable on each of the nodes in the RAC cluster. Configure a new UDEV, such as /etc/udev/rules.d/99-udev-oracle.rules, as shown in Figure 2-10.

Example 2-10 Example of changing the UDEV

```
# cat 99-udev-oracle.rules
ACTION=="add|change", ENV{ID_PATH}=="ccw-0.0.7406", ENV{DEVTYPE}=="partition",
SYMLINK+="oracleasm/asm7406", GROUP="oinstall", OWNER="oracle", MODE="0660"
ACTION=="add|change", ENV{ID_PATH}=="ccw-0.0.7407", ENV{DEVTYPE}=="partition",
SYMLINK+="oracleasm/asm7407", GROUP="oinstall", OWNER="oracle", MODE="0660"
```

A restart is required for modifications of UDEV rules to take effect. For Red Hat 8 / SUSE Linux Enterprise Server 12, you can run the following UDEV commands.

```
# udevadm control --reload-rules; udevadm trigger
```

This change can be done dynamically. Then, run the ls commands to confirm that the file permissions are set correctly (based on the devices configured in your UDEV rules), as shown with the following command and its results.

```
# ls -lL /dev/oracleasm
brw-rw---- 1 oracle oinstall 94, 61 Jun 12 19:18 asm7406
```

2.3.3 Configuring Oracle on File System Storage

For file systems, the recommended file system is the XFS file system. XFS is a high-performing, journaling Linux file system and is part of the mainline Linux Kernel. XFS is designed for high scalability and provides near native I/O performance even when the file system spans multiple storage devices.

It is recommended to separate the Oracle Data and redolog file systems. For high IO databases, the file system should be striped across multiple disk storage devices.

For configuration of spectrum scale shared file systems please refer to the following IBM Redbooks publication:

- ▶ Chapter 5 of [Best practices and Getting Started Guide for Oracle on IBM LinuxONE, REDP-5499](#)

2.3.4 Required software directories

The Oracle software installer requires the following directories be either identified or created. In our lab environment, we used /u01 as our mount point-directory (the Oracle base directory) and performed the following steps.

- ▶ Oracle Base Directory

The following directory is used:

```
ORACLE_BASE = /mount_point/app/software_owner
```

In this installation example, we created a 20GB Logical Volume to host Oracle software and data.

Where software owner is the operating system user name of the software owner that is installing the Oracle software; for example, oracle or grid.

- ▶ Oracle Inventory Directory

This directory stores an inventory of all software that is installed on the system. It is required and shared by all Oracle software installations on a single system. If Oracle Inventory paths exist, the Oracle Universal Installer continues to use that Oracle Inventory.

The Oracle Inventory path is in /var/opt/oracle/oraInst.loc.

Example 2-11 shows the command that is used to check the contents and the existence of the oraInst.loc file with the results displaying the contents of the file.

Example 2-11 Examining the oraInst.loc

```
# cat /var/opt/oracle/oraInst.loc
inventory_loc=/oraInventory
inst_group=oinstall
```

- ▶ Oracle Home Directory

The following directory is where the Oracle software product is installed:

```
ORACLE_HOME=$ORACLE_BASE/oracle_base/product/12.2.0/dbhome_1
```

If ASM is not used, create a Linux file system in the following steps by using Yast or Linux commands.

1. Create the initial mount point:

```
# mkdir /u01
# chown -R oracle:oinstall /u01 l3oradb2:~
# chmod -R 775 /u01
```

2. Create a simple file system:

- ECKD DASD: Run the following commands to create a simple file system on ECKD DASD:

```
dasdfmt -p -y -b 4096 -f /dev/dasdan
fdasd -a /dev/dasdan
```

- FCP/SCSI: Run the following command to create a simple file system on FCP/SCSI:

```
fdisk /dev/mapper/myp1 pvcreate /dev/mapper/myp1p1
```

For both disk storage types create a volume group, and then a logical volume by using the commands shown in Example 2-12.

Example 2-12 Create volumes

```
pvcreate /dev/dasdan1 vgcreate orvg /dev/dasdan1
vgcreate orvg /dev/mapper/myp1p1
lvcreate --name orlv -l 100%FREE orvg mkfs.xfs /dev/orvg/orlv
```

3. Mount the logical volume by using the following command:

```
# mount -t xfs /dev/orvg/orlv /u01
```

4. For resiliency after system reboot, insert your mount statement into the /etc/fstab file. Ours is shown in Example 2-13.

Example 2-13 Mount statement in the /etc/fstab file

```
/dev/oravg/oralv /u01 xfs defaults 1 1
```

2.4 Oracle single instance installation

In this section, we discuss the following:

- ▶ “Installing Oracle Grid Infrastructure for a stand-alone server” on page 23
- ▶ “Oracle database binary file installation” on page 28
- ▶ “Creating the Oracle Database” on page 31

2.4.1 Installing Oracle Grid Infrastructure for a stand-alone server

Oracle Grid Infrastructure for a stand-alone server is a version of Oracle Grid Infrastructure that supports single instance databases. Oracle Grid Infrastructure for a stand-alone server includes Oracle Restart and Oracle Automatic Storage Management (ASM).

Oracle Restart monitors and restarts Oracle Database instances, Oracle Net Listeners, and Oracle ASM instances.

Oracle ASM manages a small number of storage pools that are called ASM Disk Groups. Database-related files are assigned to ASM Disk Groups and ASM manages the layout and data organization.

Oracle combined these two infrastructure products into a single set of binaries that is installed into an Oracle Restart home directory. To use Oracle ASM or Oracle Restart, we must install Oracle Grid Infrastructure for a stand-alone server before we install and create the database.

The following documents were referenced for installing Oracle Grid Infrastructure for a standalone server:

- ▶ [Oracle Database Installation Guide, 19c for Linux, E96432-34](#)
- ▶ [RPM Checker when Installing Oracle Database 19c on Linux on System z® \(Doc ID 2553465.1\)](#)

The following is an overview of the steps that were completed to install Oracle Grid Infrastructure (19c) as a stand-alone server on a Red Hat Enterprise Linux guest on an IBM LinuxONE server. The following are the high level steps:

1. Validate Linux guest requirements.
2. Validate required Disk Space.
3. Validate required Linux RPMs for Oracle 19c.
4. Disable Linux Transparent Huge Pages.
5. Update Linux Kernel parameters.
6. Huge page setup for Oracle Databases.
7. Create users or groups.
8. Create user directories for Oracle product installations.
9. Set up udev rule for storage for ASM.
10. Install Grid infrastructure for the standalone server binary.
11. Verify Grid infrastructure for the standalone server binary.

Installation

The Oracle Grid Infrastructure software is available as an image, available for download from the Oracle Technology Network website, or the Oracle Software Delivery Cloud portal. For more information on downloading the Oracle Grid Infrastructure see the [Oracle Database Installation Guide, 19c for Linux, E96432-34](#).

Example 2-14 shows the image file that we downloaded and is available for our use.

Example 2-14 Grid download file

```
[oracle@zdblab08 grid]$ ls -aLF /mnt/oraclelfs/oracode/19c/V982652-01_19cGrid.zip
-rw-r--r--. 1 oracle dba 2519769777 Jun 7 2019 /mnt/oraclelfs/oracode/19c/V982652-01_19cGrid.zip
```

1. Create the grid home directory, and extract the image files in this directory by running the following commands:

```
mkdir -p /u01/grid
cd /u01/grid
unzip -q /mnt/oraclelfs/oracode/19c/V982652-01_19cGrid.zip
```

The extracted files are shown in Figure 2-4.


```

[oracle@zdbl08 u01]$ mkdir -p /u01/grid
[oracle@zdbl08 u01]$ cd /u01/grid
[oracle@zdbl08 grid]$ unzip -q /mnt/oraclefs/oracode/19c/V982652-01_19cGrid.zip

[oracle@zdbl08 grid]$
[oracle@zdbl08 grid]$ ls
addnode      css          demo        gnpnp       instantclient  jlib        nls          ord          perl         racg         rootupgrade.sh  sqlplus      utl
assistants  cv           diagnostics gridSetup.sh inventory      ldap        OPatch       ords         plsql       rdbms       runcluvfy.sh  srvrm        welcome.htm
bin          dbjava      dmu         has         javavm         lib         opmn         oss          precomp     relnotes      sdk           suptools     wwg
clone       dbs         env.ora     hs          jdbc           md          oracore      oui          QOPatch     rhp           slax          ucp          xag
crs         deinstall  evm         install     jdk            network     oradiag     aime        owm          qos           root.sh       sqlpatch     usm          xdk

```

Figure 2-4 Grid extract Files

2. Back up the OPatch directory, and first apply the 6880880 OPatch file (Example 2-15) and then the latest Release Update from the [My Oracle Support](#) site.

Example 2-15 Apply OPatch 6880880

```

[oracle@zdbl08 grid]$ mv OPatch OPatch.old
[oracle@zdbl08 grid]$ unzip -q /mnt/oraclefs/oracode/19c/p6880880_190000_Linux-zSer.zip

```

3. In a graphical environment, such as VNC, and as the grid user, start the Grid Infrastructure installation process by running the following command:

```
gridSetup.sh
```

or configure a response file for a silent install installation.

It is advisable to download the latest database patch RU for the Linux One (or Linux on IBM Z) platform, and unzip the patch ahead of time. The same patch can be used for the Grid and Database software installations.

In our case, we downloaded p36233126_190000_Linux-zSer.zip and unzipped the file into our software staging directory. Unzip will create a file directory under the patch number. In our case folder 36233126 was created.

To run the gridSetup.sh with the patch with the VNC GUI, run the following command. The command and its results are shown in Example 2-16.

```
./gridSetup.sh -applyRU <path to the latest unzipped RU Patch>
```

Example 2-16 gridSetup.sh with Oracle RU Patch

```

[oracle@hsora2 grid]$ ./gridSetup.sh -applyRU ../stage/36233126/
Preparing the home to patch...
Applying the patch ../stage/36233126/...

```

Once the RU patch is applied you should get the Oracle Grid Installer Step 1 of 7 display as shown in Figure 2-8.

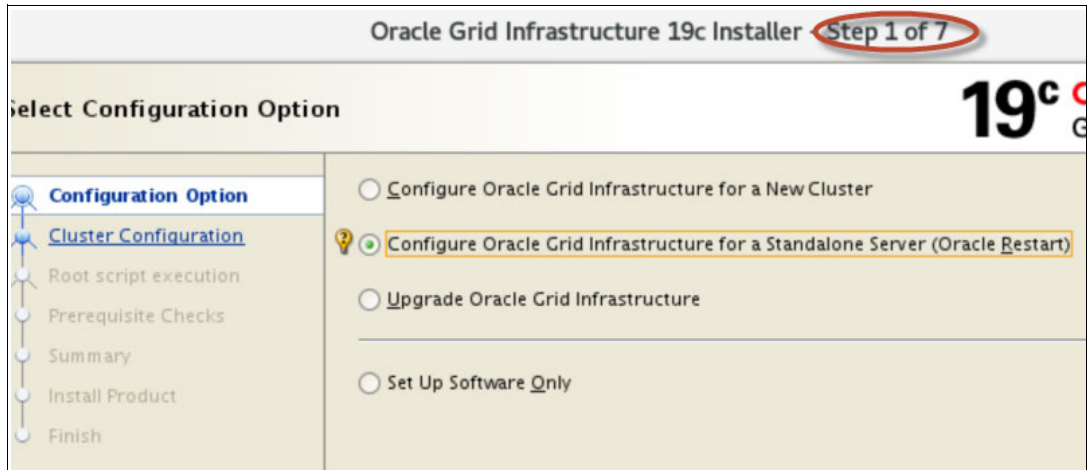


Figure 2-5 gridSetup.sh with Oracle RU Patch

4. Select Configure Oracle Grid Infrastructure for a Standalone Server (non-RAC) then select the second option (**Configure an Oracle Standalone Cluster**) and click the next button.
5. In the Create ASM Disk Group screen, specify the settings that are listed in Table 2-2.

Table 2-2 Create ASM Disk Group window settings

Field name	Value	Comments
Disk group name	DATA	
Redundancy	External	
Allocation Unit Size	4 MB	
Select Disks	/dev/mapper/orad1	
Disk Discovery Path	/dev/mapper/*	/dev/oracleasm/* for ECKD

6. In the Specify ASM Password window, choose Use same passwords for these accounts option and specify a password that conforms to your security requirements.
7. In the Enterprise manager grid control screen, specify the required information if the database is to be managed by Enterprise Manager (EM) Cloud Control. This step can be configured later if needed as well.
8. In the Operating System Groups screen, specify the settings listed in Table 2-3.

Table 2-3 Operating System Groups window settings

Field Name	Value
Oracle ASM Administrator (OSASM) Group	asmadmin
Oracle ASM DBA (OSDBA) Group	asmdba
Oracle ASM Operator (OSOPER for ASM) Group optional	oinstall

Ignore any invalid group chosen messages.

9. In the next window, specify the value for the Oracle base Directory. This value is the location where all the diagnostic information for all the Oracle products are stored. In our example, we specified /u01/app/oracle for the directory name.

10. In the **Create Inventory** window, specify the value for the Oracle Inventory Directory. This value is the location where Oracle stores the inventory of all the products that are installed. For our example, we specified `/u01/app/orainventory`.

11. In the Root Script execution configuration window, select the **Automatically run configuration scripts** option and enter the root user credentials.

In our installation, the installation process performed prerequisite checks and reported the following errors:

- Swap size error
- Package `cvuqdisk-1.0.10-1` missing error.

The swap size error can be ignored if encountered as long as at least 1GB of swap is configured. For the Package missing error, choose the Fix and Check again option. This choice created the fix and provided the information about where the fix available and how to run it, as shown in Figure 2-6.

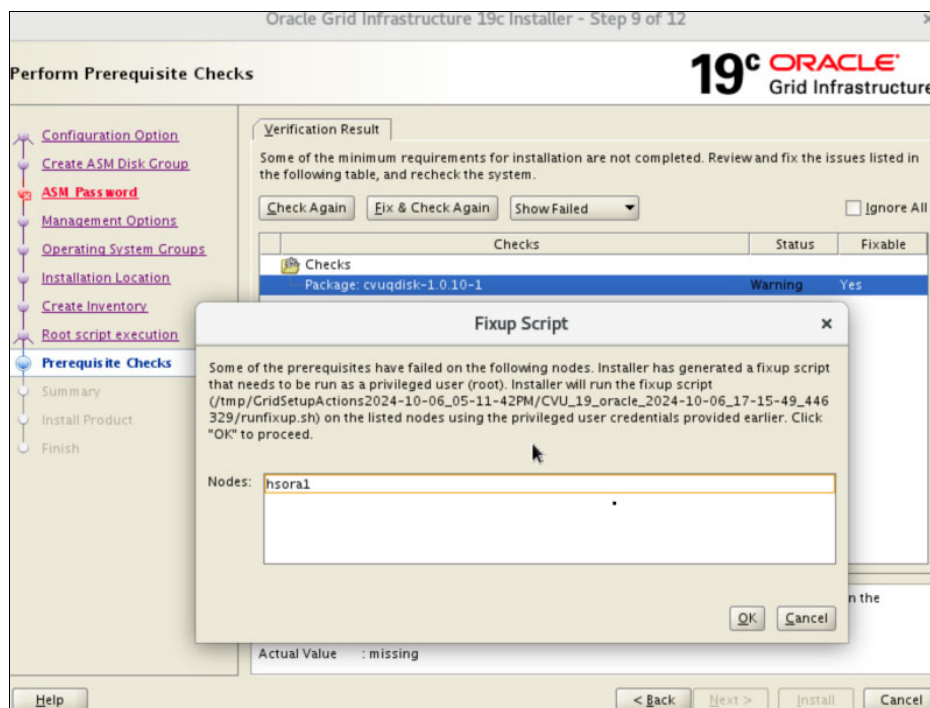


Figure 2-6 `runfixup.sh` script

12. For an Oracle Single Installation for a stand-alone server, click OK for to run the `runfixup.sh` script that was prepared by the installer. Selecting OK automatically runs the command by using root credentials. Otherwise, a script is created, which you can run manually.

13. When that has completed, an installation Summary window is displayed. Review all the information Select Install to start the Grid Installation which will run the configuration scripts. The installer will prompt that Configuration scripts as the root user need to be run, click "Yes" to allow the configuration scripts to be run as the root user.

14. After the installer is complete and the Grid Infrastructure is configured, the successful completion window is displayed, as shown in Figure 2-7.

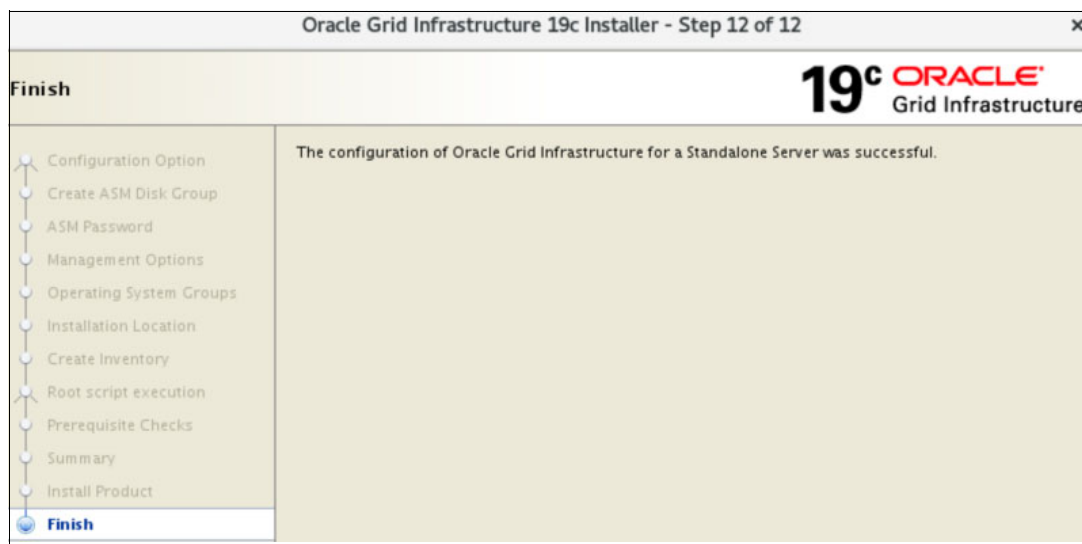


Figure 2-7 Installation complete

2.4.2 Oracle database binary file installation

Use the downloaded Oracle Database Infrastructure image file from “Obtaining the Oracle 19c software for IBM LinuxONE” on page 12 for the stand-alone server installation then perform the following steps.

1. Create the database home directory, and extract the image files in this directory by running the following commands with the Linux user ID, oracle.

```
mkdir -p /u01/db
cd /u01/db
unzip -q /mnt/oracle/ifs/oracode/19c/V982648-01_193DB_s390x.zip
```

2. As in the Grid installation, we recommend downloading and installing the latest OPatch and Oracle RU patch to avoid any known issues and save time with having to patch the database later.

To do this, back up the OPatch directory then uncompress the latest 6880880 Opatch for 19c which can be obtained from the support.oracle.com patches download directory from the Linux on IBM Z platform folder. Use the following commands to do this:

```
mv OPatch Opatch.old
unzip ../stage/p6880880_190000_Linux-zSer.zip
```

Download the latest database patch release update (RU) for the Linux on IBM Z platform and unzip the patch ahead of time. The same patch can be used for the Grid and Database software installations, so if this was done in section , “Installation” on page 24, you need not do it again.

In our case we downloaded p36233126_190000_Linux-zSer.zip and uncompressed the file into our software staging directory. This will create a file directory under the patch number. In our case, folder 36233126 was created. Run the runInstaller executable with the -applyRU option. The command we used in our lab environment is shown in Example 2-17 and is as follows:

```
./runInstaller -applyRU ../stage/36233126/
```

It may take a few extra minutes for the Oracle installer screen to display, while the patch is being applied.

Example 2-17 runInstaller with applyRU

```
[oracle@hsora2 db]$ ./runInstaller -applyRU ../stage/36233126
Preparing the home to patch...
Applying the patch ../stage/36233126/...
```

Once the Installation configuration screen is displayed, complete the following steps:

1. In the Select Configuration Option window, select Set Up Software Only, as shown in Figure 2-8 and click the "Next" button to continue the installation.

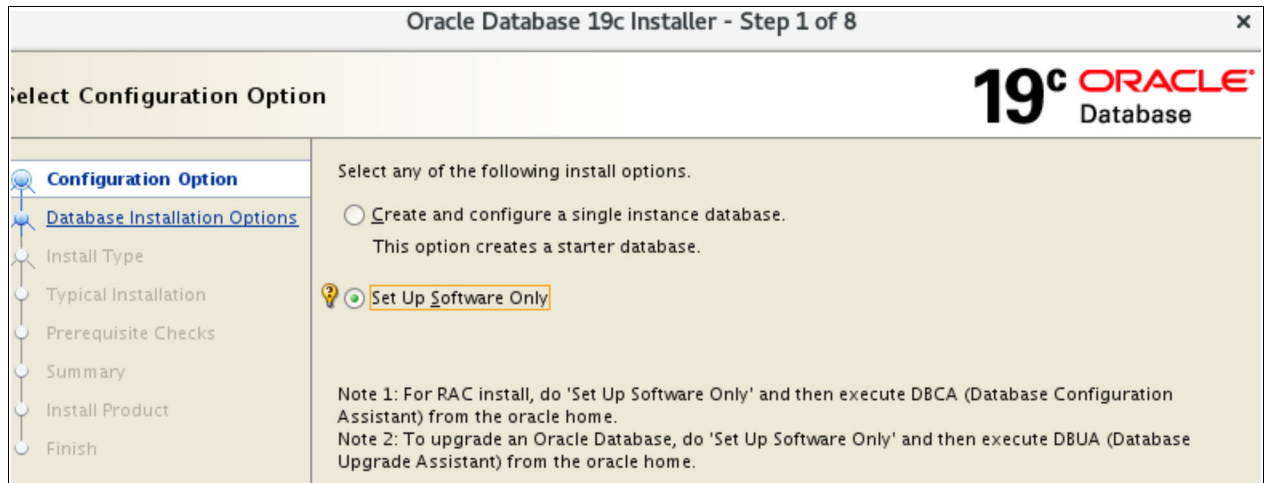


Figure 2-8 Set Up Software Only

2. In the Select Database Installation Option window, select Single Instance database installation.
3. In the next Select Database Edition window, click Enterprise Edition.
4. Specify the value for the Oracle base directory. This value is the location where all the diagnostic information (among which are trace files, dumps, and core files) for all the Oracle products that are stored. In our example, we specified /u01/app/oracle.
5. The Privileged Operating System Groups window is next.
 - a. The OSDBA group is mandatory. Specify dba for all groups. This group identifies operating system user accounts that have database administrative privileges (the SYSDBA privilege).
 - b. The OSOPER group for Oracle Database is optional. This group grants the OPERATOR privilege to start up and shut down the database (the SYSOPER privilege). Accept oper for the Database operator group to keep things simple.
6. The installation process will perform prerequisite checks and may report a Swap size warning, which can be ignored if at least 1GB is configured on the Linux system.
7. The summary window is displayed. Select Install.

The Install Product window will prompt you to run, as root, the root.sh script. Run it and then, click OK (as shown in Figure 2-9).

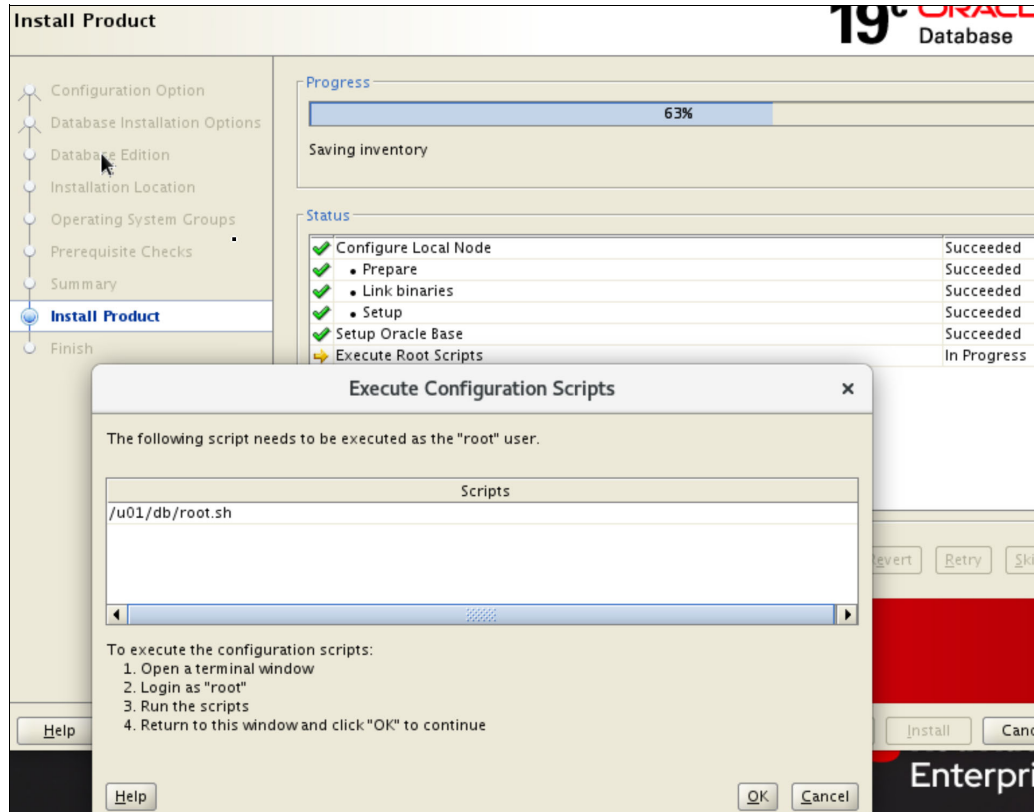


Figure 2-9 Run `root.sh` script as root user

Figure 2-10 shows the command and output run as the root user.

```
[oracle@hsora2 ~]$ su
Password:
[root@hsora2 oracle]# cd /u01/db
[root@hsora2 db]# ./root.sh
Performing root user operation.

The following environment variables are set as:
  ORACLE_OWNER= oracle
  ORACLE_HOME=  /u01/db

Enter the full pathname of the local bin directory: [/usr/local/bin]:
The contents of "dbhome" have not changed. No need to overwrite.
The contents of "oraenv" have not changed. No need to overwrite.
The contents of "coraenv" have not changed. No need to overwrite.

Entries will be added to the /etc/oratab file as needed by
Database Configuration Assistant when a database is created
Finished running generic part of root script.
Now product-specific root actions will be performed.
```

Figure 2-10 `root.sh`

Upon completion, the Finish window displays and indicates that the registration installation of the Oracle Database software was successful.

2.4.3 Creating the Oracle Database

The next step is to create the Oracle Database and for that you can use Database Configuration Assistant (dbca) installer in a GUI xterm window or use the silent install with configuration file methodology.

In our example we will show the xterm GUI method. For silent Install steps see section 8.1.15 Silent installation in the IBM Redbooks publication, [Oracle 19c on IBM LinuxONE, SG24-8487](#).

1. Add or uncomment the ORACLE_SID in the oracle user’s profile, which is in the user’s HOME directory by using the **export ORACLE_SID=orc1fs** command.
2. Start the Database Configuration Assistant (dbca) installer to create a database by running the dbca command in a VNC client terminal.
3. Figure 2-11 shows the Select Database Operation window. Select the Create a database option.

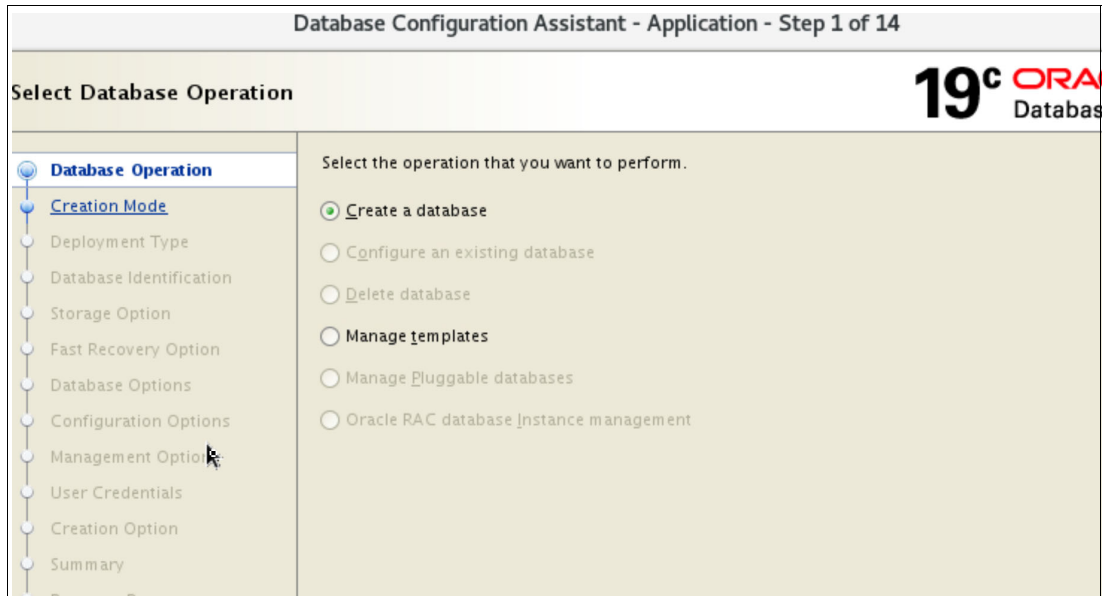


Figure 2-11 Database Configuration Assistant - create database

4. The database creation panels are listed by appearance in Table 2-4 with our selected options in bold.

Table 2-4 Database creation panels in order of appearance with their options

Panel	Comments and options
Step 1 of 15 Creation mode	The following options are available to choose in this panel: <ul style="list-style-type: none"> ▶ Create a database ▶ Manage templates
Step 2 of 15 Deployment Type	Choose the Advanced Configuration option.

Panel	Comments and options
Step 3 of 15 Select Database Deployment Type	In this panel, we have a choice to select the Database type: <ul style="list-style-type: none"> ▶ Oracle Single instance database ▶ Oracle RAC Database ▶ Oracle RAC One Node Choose the template for your database: <ul style="list-style-type: none"> ▶ Custom Database ▶ Data Warehouse
Step 4 of 15 Specify Database Identification Details	In this panel, specify the Global Database name and SID. You also can create as a Container database. If you selected the Container database option, you can choose to configure Local Undo table space for PDBs. You can also choose to create either an empty Container database or you can specify the number of PDBs and their prefix names. In our case, we selected "test1" in our configuration, and we unchecked the "create container database" dialog box.
Step 5 of 15 Select Database Storage Option	Provide your database storage attributes. In our case, we chose the "Automatic Storage Management (ASM) option and selected the ASM disk group that we chose to store Oracle Datafiles.
Step 6 of 15 Fast Recovery Option	Specify the attributes (size and location) for the Fast Recovery Area for the database. You also can choose to enable the Archive mode. We did not specify the Fast Recovery Area and will not enable archiving until our database is loaded with data.
Step 7 of 15 Network Configuration	In this panel, you can use any Listeners that are running or create a Listener for the database. In our lab environment, we chose the default listener that is configured, named LISTENER.
Step 8 of 15 Data Vault Option	Database Vault and Oracle Label security are configured here. Select the DB options that are required for your database. We did not select any additional options in order to provide the best performance for our test workload.
Step 9 of 15 Configuration Options	This panel provides the complete set of configuration options from selecting memory management to initialization parameters and includes character sets for your database. We suggest using "Automatic Shared Memory Management" in order to leverage Linux large pages for the Oracle SGA memory.

Panel	Comments and options
Step 10 of 15 Management Options	Specify the Enterprise Manager information if one is running in your environment. You also can use the Enterprise Manager Database express, with which you manage this database in your server. We accepted the default Enterprise Manager values. You can unset the Configure Enterprise Manager Express option if you plan to use a centralized Oracle Enterprise Manager configuration.
Step 11 of 15 User Credential	Specify the passwords. We chose Use the same administrative passwords for all accounts and specified the password that conformed to Oracle specifications.
Step 12 of 15 Database Creation Options	We selected Create Database . In this install panel you can also choose to change any of the default Oracle parameters and the size and location of the default database files such as TEMP, UNDO and the Oracle REDO logs You can save the configured database as a template and run any other scripts after the database is created and save the database creation scripts. Also, the panel provides more opportunities to customize any storage and initialization parameters.
Step 13 of 15 Create database - Summary	A summary of the chosen configuration is displayed. This configuration can be saved in a response file by clicking Save Response file .
Step 14 of 15 Create database - Progress	This panel will display the progress of the database creation, this step may take some time (up to 20 minutes).
Step 15 of 15 Finish	If everything completes successfully you should see the database completion status, as shown in Figure 2-12. You can exit the installation panel.

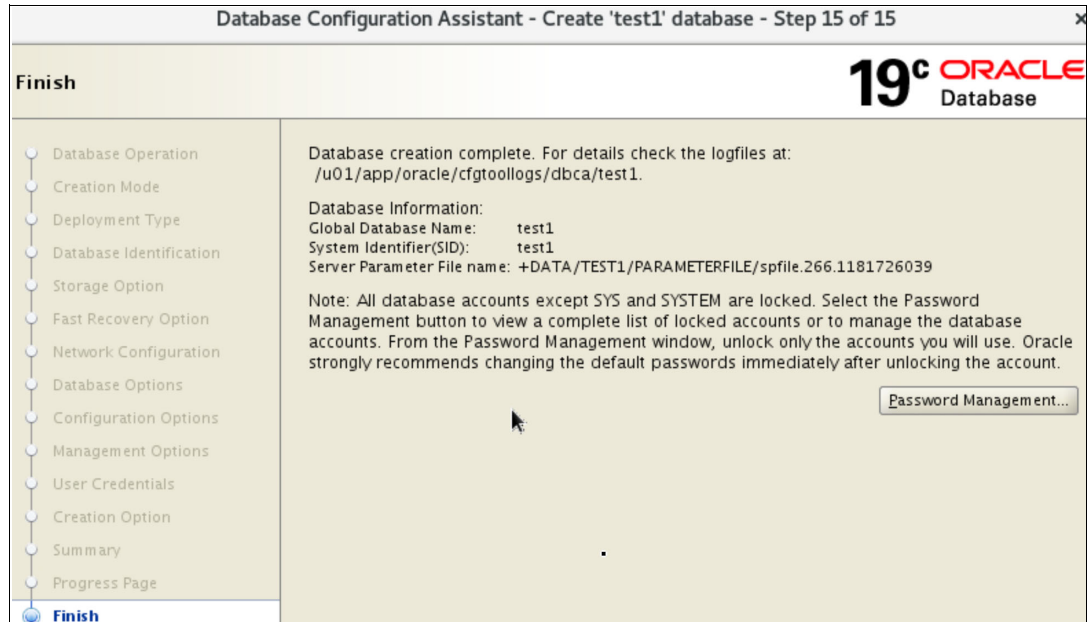


Figure 2-12 Successful completion

2.5 Oracle Real Application Cluster (RAC) Installation

In this section we will provide the latest updates on installing Oracle RAC on IBM LinuxONE.

For more detailed steps, see Chapter 9, of the IBM Redbooks publication, [Oracle 19c on IBM LinuxONE, SG24-8487](#).

The Oracle Advanced Cluster File System (ACFS) is not supported on IBM LinuxONE. IBM and Oracle recommend using secure IBM Spectrum® Scale (General Parallel File System or GPFS) for your shared file system needs, including Oracle Goldengate implementations. See the following support note for more information on this:

[IBM Spectrum Scale 5.0 and Oracle RAC on IBM Linux on System z \(Doc ID 2437480.1\)](#)

The Linux pre-installation processes in section 2.1, “Obtaining the Oracle 19c software for IBM LinuxONE” on page 12 apply for Oracle RAC installs as well. In this section, we discuss additional requirements that are needed for a successful RAC installation, including the following:

- ▶ “Oracle network requirements on IBM LinuxONE ” on page 35
- ▶ “Oracle SCAN configured in DNS ” on page 36
- ▶ “Public, Private and Virtual IP address requirements” on page 37
- ▶ “Network Interface Name (NIC) consistency for Oracle RAC ” on page 37
- ▶ “Oracle RAC firewall configuration (optional)” on page 38
- ▶ “Time synchronization across cluster nodes” on page 39
- ▶ “Time synchronization across cluster nodes” on page 39
- ▶ “SSH user equivalency for Oracle RAC installs ” on page 40

2.5.1 Oracle network requirements on IBM LinuxONE

For a deeper dive into the Oracle network requirements, see section 7.2 and 9.1.3 in the IBM Redbooks publication, [Oracle 19c on IBM LinuxONE](#), SG24-8487.

The minimum recommendation for an online transaction processing (OLTP) database is a 1 Gb network. Decision Support or Data Warehouse databases can require greater than a 1 Gb network interface for Oracle Interconnect® for a cloud platform.

The network interconnect for Oracle RAC must be on a private network. If you are configuring the private interconnect between separate IBM LinuxONE machines, there must be a physical switch configured ideally with one network hop between systems. The private interconnect should also be a private IP address in the range that is shown in Example 2-18.

Example 2-18 Private IP address for private interconnect

Class A: 10.X.X.X
 Class B: 172.(16-31).X.X
 Class C: 192.168.X.X where (0 <= X <= 255)

An Oracle RAC workload sends a mixture of short messages of 256 bytes and database blocks of the database block size for the long messages. Another important consideration is to set the Maximum Transition Unit (MTU) size to be a little larger than the data base block size for the database. Oracle RAC configuration options are shown in Table 2-5.

Table 2-5 Oracle RAC configuration options on IBM LinuxONE

Architecture	Oracle Private Network (interconnect)	Oracle Public Network
All z/VM Linux guests in one LPAR	<ul style="list-style-type: none"> ▶ Private Layer2 VSwitch Guest LAN OSA recommended ▶ Real layer 2 HiperSocket possible ▶ Guest LAN HiperSocket not supported 	<ul style="list-style-type: none"> ▶ Shared Public VSwitch recommended ▶ Shared or dedicated OSA card is possible
z/VM Linux guests on different LPARs	<ul style="list-style-type: none"> ▶ Real layer 2 HiperSocket recommended ▶ Private Layer 2 Gigabit OSA card possible 	<ul style="list-style-type: none"> ▶ Shared Public VSwitch recommended ▶ Shared or dedicated OSA card
z/VM Linux guests on different physical machines	Private Layer 2 Gigabit OSA card recommended with physical switch in between (one hop)	Dedicated OSA card possible

The following methods can be used to configure a high availability (HA) solution for an IBM Z environment that uses a multi-node configuration:

- ▶ **Virtual Switch (Active/Passive)**
 When one Open System Adapter (OSA) Network port fails, z/VM moves the workload to another OSA card port; z/VM handles the fail over.
- ▶ **Link Aggregation (Active/Active)**
 Allows up to eight OSA-Express adapters to be aggregated per virtual switch. Each OSA-Express port must be exclusive to the virtual switch (for example, it cannot be shared); z/VM handles the load balancing of the network traffic.

- ▶ Linux Bonding

Creates two Linux interfaces (for example, eth1 and eth2) and create a bonded interface bond0 made up of eth1 and eth2, which the application uses. Linux can be configured in various ways to handle various failover scenarios.

- ▶ Oracle HAIP

Oracle can have up to four Private Interconnect interfaces to load balance Oracle RAC Interconnect traffic. Oracle handles the load balancing and is exclusive to Oracle Clusterware implementations. To use of HAIP, following parameters must be set in /etc/sysctl.conf file otherwise communication between nodes will be interrupted causing the node eviction, one of the nodes not joining cluster

The Linux kernel parameters may need to set at /etc/sysctl.conf file in order to avoid any routing issue between the network interfaces.

Example 2-19 Linux kernel parameters

```
net.ipv4.conf.eth2.rp_filter = 2 ## Private Interface 1
net.ipv4.conf.eth1.rp_filter = 2 ## Private Interface 2
net.ipv4.conf.eth0.rp_filter = 1 ## Public Interface
```

2.5.2 Oracle SCAN configured in DNS

In order to complete a successful Oracle RAC install, an Oracle Single Client Access Name (SCAN) should be configured in your Domain Name Server (DNS) system.

Oracle SCAN provides a single name in which a client server can use to connect to a particular Oracle database. The main benefit of SCAN is the ability to keep a client connection string the same, even if changes within the Oracle RAC Database environment occur, such as adding or removing of nodes within the cluster.

To configure SCAN in your DNS, the following points should be considered:

- ▶ Network administrator to create a single name in DNS that resolves to 3 IP addresses using a round-robin algorithm.
- ▶ The SCAN domain name must be unique within the network.
- ▶ Three IP addresses are recommended.
- ▶ All of the public IP, VIP and the SCAN IP addresses should be in the same subnet
- ▶ The name must be 15 characters or less in length, not including the domain, and must be resolvable without the domain suffix (for example: “sales1-scan” must be resolvable as opposed to “scan1-scan.example.com”).
- ▶ The IPs must not be assigned to a network interface (on the cluster), since Oracle Clusterware will take care of it.

In Example 2-20, orascan0087, our SCAN, is configured with three IP addresses pointing to the SCAN name DNS entry.

Example 2-20 Sample lookup of SCAN configuration

```
nslookup orascan0087
Server:          129.40.106.1
Address:         129.40.106.1#53
Name:   orascan0087.pbm.ihost.com
Address: 129.40.24.9
```

Name: orascan0087.pbm.ihost.com
Address: 129.40.24.8
Name: orascan0087.pbm.ihost.com
Address: 129.40.24.7

Run an **nslookup** command using your SCAN name to help ensure the DNS name is configured correctly with three IP addresses configured to each SCAN name for each Oracle RAC cluster that you require.

2.5.3 Public, Private and Virtual IP address requirements

Oracle RAC has the following network requirements:

- ▶ One IP per node for Variable IP (VIP) is required
- ▶ One Private IP per node for Inter-Connect
- ▶ One Public IP for Host

When setting up your network, consider the following:

- ▶ Each node should have at least two network interface cards (NIC), or network adapters. Public interface names must be the same for all nodes. If the public interface on one node uses the network adapter 'PublicLAN', then you must configure 'PublicLAN' as the public interface on all nodes.
- ▶ Four network cards are preferred (per node) when bonded and for high availability.
- ▶ You should configure the same private interface names for all nodes as well. If 'priv1' is the private interface name for the first node, then 'priv1' should be the private interface name for your second node, and subsequent nodes of the RAC cluster.
- ▶ Multicast based communication should be enabled on the nodes of the cluster and on the network switches used for the private interconnect. Refer Oracle Doc ID 1212703.1 for more information.
- ▶ For the private network, the end points of all designated interconnect interfaces must be completely reachable on the network. Every node in the cluster should be able to connect to every private network interface in the cluster.
- ▶ Dedicated Gigabit switch ports should be used for Oracle RAC interconnect network interfaces.
- ▶ If a shared network switch is used for Oracle RAC interconnect, then a dedicated untagged private nonroutable VLAN should be configured. VLAN should not span the switchblades. An overloaded backplane or processor on the physical switch will adversely impact the performance of the cache fusion traffic over the private network. For this reason Oracle recommends dedicated switch for private interconnect Jumbo frames (MTU 9000) should be configured for interconnect (provided the private interconnect switch supports the jumbo frames).
- ▶ All the entries should be in /etc/hosts file and the VIP should not be accessible.
- ▶ All of the public IP, VIP and the SCAN IP addresses should be in the same subnet.

2.5.4 Network Interface Name (NIC) consistency for Oracle RAC

If installing Oracle RAC, the network interface names on each RAC node need to be consistent. It is recommended to have the same Linux network interface name on all nodes of the Oracle RAC cluster. By default, some operating systems change the network interface names. You must confirm that the interface name is consistent between nodes.

If it is not consistent, change the interface names to a consistent name. For more information on how to do this, see [Configuring the System's Network](#).

In Example 2-21, we list our plan to rename encXXXX based interface names from Node1 and Node2 to pub1 and priv1 respectively.

Example 2-21 Changing the interface name

```
Node1: enc1480 -> pub1, enc1490 -> priv1
Node2: enc1483 -> pub1, enc1493 -> priv1
```

This will prevent your session from being disconnected with `ifdown` commands. Open an SSH session from Node2 into Node1 on the private interconnect IP, to change the Public Interface. Use the commands shown in Example 2-22 to change the interface name.

Example 2-22 Changing the interface name

```
export ONAME=enc1480
export NNAME=pub1
ifdown ${ONAME}
ifconfig ${ONAME} down
ip link set ${ONAME} name ${NNAME}
cat /etc/sysconfig/network-scripts/ifcfg-${ONAME} | sed -e
"s/${ONAME}/${NNAME}/g" > /etc/sysconfig/network-scripts/ifcfg-${NNAME}
ifconfig ${NNAME} up
ifup ${NNAME}
ifconfig ${NNAME}
```

Reboot the server and verify whether the network interface names have been properly changed.

Interface configuration (ifcfg) files control the software interfaces for individual network devices. As the system boots, it uses these files to determine what interfaces to bring up and how to configure them. We suggest creating an ifcfg script. We copied an ifcfg file from the `/etc/sysconfig/network-scripts/` directory and called ours `ifcfg-privora` and saved it in the `/etc/sysconfig/network-scripts/` directory.

We then edited our file and commented out “ONBOOT=yes and the “SUBCHANNELS” line. Ensure that `NM_CONTROLLED=yes`. In our new file, `ifcfg-privora` we added the following Subchannels:

- ▶ `DEVICE=privora`
- ▶ `ONBOOT=yes`

2.5.5 Oracle RAC firewalld configuration (optional)

If installing Oracle RAC, disable the Linux firewalld service in order for the Oracle RAC services to communicate properly between the Oracle RAC nodes. To do this, open an SSH session and run the following commands:

```
# systemctl status firewalld
# systemctl stop firewalld
# systemctl disable firewalld
```

If the firewalld service must be running, then various firewall ports will need to be opened up in order for Oracle RAC to be installed. Involve the Linux system administrator to modify firewall rules to allow the Oracle RAC private interconnect nodes to connect to each other per Oracle

MOS note: [RAC instabilities due to firewall \(netfilter/iptables\) enabled on the cluster interconnect \(Doc ID 554781.1\)](#).

The following is an example of the procedure we used to configure firewalld with Oracle RAC installations on LinuxONE.

1. To allow access to a specific database client with an IP address of 10.19.142.54 and to make requests to the database server via SQL*Net using Oracle's TNS (Transparent Network Substrate) Listener (default port of 1521), the following permanent firewall rule within the public zone must be added to the firewalld configuration on each node within the Oracle RAC cluster.

```
# firewall-cmd --permanent --zone=public --add-rich-rule="rule family="ipv4"
source address="10.19.142.54" port protocol="tcp" port="1521" accept"
```

2. Ensure the firewall allows all traffic from the private network by accepting all traffic (trusted) from the private Ethernet interfaces priv1 from all nodes within the Oracle RAC cluster. It is highly recommended that the private network be isolated and communicate only between nodes locally. We used the following command to accomplish this.

```
# firewall-cmd --permanent --zone=trusted --change-interface=priv1
```

3. On node one of the Oracle RAC cluster, use the following command to add the public source IP of all remaining nodes. This reference environment only adds the public IP of node two of the Oracle RAC cluster.

```
# firewall-cmd --permanent --zone=trusted --add-source=10.19.142.52/21
```

4. Once the rules have been added, run the following command to activate.

```
# systemctl restart firewalld.service
```

5. On node 2 of the RAC cluster, add the rule to allow temporary access for the install by using the following command.

```
# firewall-cmd --permanent --zone=trusted --add-source=10.19.142.52/21
```

6. Once the rules have been added, run the following command to activate:

```
# systemctl restart firewalld.service
```

7. To re-enable the firewalld rules on each of the RAC nodes after the RAC install is completed be sure to remove the public IP firewall rules that were previously created after the installation. Use the following command to do this.

```
# firewall-cmd --permanent --zone=trusted --remove-source=10.19.142.52/21
```

For more information on this topic, see the following:

- ▶ [Section 6.1.2 Removal of firewalld Trusted Source Address](#)
- ▶ [RAC instabilities due to firewall \(netfilter/iptables\) enabled on the cluster interconnect \(Doc ID 554781.1\)](#)
- ▶ [Chapter 1. Using and configuring firewalld](#)

2.5.6 Time synchronization across cluster nodes

You have the following two options for time synchronization:

- ▶ Oracle Cluster Time Synchronization Service
- ▶ An operating system configured network time protocol (NTP) such as chronyd or ntpd

The Oracle Cluster Time Synchronization Service is designed for organizations whose cluster servers are unable to access NTP services. If you use NTP, then the Oracle Cluster Time

Synchronization daemon (CTSSD) starts up in observer mode. If you do not have NTP daemons, then CTSSD starts up in active mode and synchronizes time among cluster members without contacting an external time server.

2.5.7 SSH user equivalency for Oracle RAC installs

User equivalence is when a single database user account can be used across multiple instances of an Oracle RAC environment. If installing Oracle RAC, user equivalency between the Linux **oracle** and **grid** user account to the other nodes is needed.

To manually configure SSH equivalency for each system, generate the SSH keys and then copy them to `userid@newserver` as shown in Example 2-23.

Example 2-23 Configure SSH equivalence

```
ssh-keygen
ssh-copy-id oracle@oracle3.cpolab.ibm.com
```

Next, run the “`ssh user@<newserver> date`” command to verify that SSH runs with prompting for a password by using the command shown in Example 2-24 with your `<newserver>` name.

Example 2-24 Verify SSH runs with prompting with results

```
ssh oracle@oracle3.cpolab.ibm.com date
Sat Aug 24 10:44:01 EDT 2024
```

If you encounter any issues, clean up the existing `~oracle/.ssh` folder and retry these steps.

For additional detailed steps in Installing Oracle RAC on LinuxONE please refer to Chapter 9, of the IBM Redbooks publication, [Oracle 19c on IBM LinuxONE, SG24-8487](#) for detailed screen shots of the Oracle RAC Installation.

2.6 Implementing Oracle RAC in containers

IBM LinuxONE supports Oracle RAC in containers with Podman currently for non-production (Dev/test) use only with Oracle 19.16 and greater.

With this support, IBM LinuxONE servers running Podman on Red Hat Enterprise Linux 8 or SLES 15 SP3+ with Oracle 19c release 19.16 or greater will be able to quickly spin up container images.

My Oracle support note, [Oracle RAC on PODMAN on IBM System z - Released Versions and Known Issues \(Doc ID 2924682.1\)](#) details what is and is not supported and any known issues.

For example, one issue is CTSSD not running in observer mode. To avoid this error, create an empty file `/etc/ntp.conf` file on each container.

For a detailed white paper on installing Oracle RAC with Podman, see [Real Application Clusters \(RAC\) in containers on IBM System Z-Implementing Oracle RAC with Podman](#).

2.7 Database migration from other platforms to LinuxONE

Customers that want to migrate their Oracle databases to IBM LinuxONE can do so knowing there are multiple methodologies available to seamlessly migrate their Oracle databases to the platform.

There are many Oracle tools available that work on the LinuxONE platform, whether the source platform is big endian or little endian. The IBM LinuxONE platform is a true hybrid-cloud platform for running not only Oracle databases, but many other open-source databases are also supported on LinuxONE. As the business needs change, CPU, memory, I/O or network system resources can be dynamically added or removed as the workload demands.

There are several migration approaches for moving databases including the following as shown in Table 2-6 Sample Oracle Migration Approaches.

- ▶ Data Pump Conventional Export/Import
- ▶ RMAN backup sets for Cross-Platform Database transport
- ▶ Cross Platform Transportable Tablespaces (XTTS) migration steps using incremental backup
- ▶ Replication including Oracle Golden Gate, for synchronizing both source and target databases.

Table 2-6 Sample Oracle migration approaches

Technology	Advantage	Disadvantage	Limit	Downtime
Oracle Data Pump	<ul style="list-style-type: none"> ▶ Cross platform ▶ Cross-version ▶ Simple to use ▶ Supports reorganizing data files 	<ul style="list-style-type: none"> ▶ Hot backup not supported 	<ul style="list-style-type: none"> ▶ Oracle BLOBs/CLOBs can sometimes have issues (verify with Oracle SR) 	<ul style="list-style-type: none"> ▶ Usually hour level depending on data volume or network transmission
RMAN Backup Sets	<ul style="list-style-type: none"> ▶ Cross-platform ▶ Mature technology ▶ Good performance ▶ High data consistency 	<ul style="list-style-type: none"> ▶ A bit higher technical skill requirements 	<ul style="list-style-type: none"> ▶ Source and target are the same version 	<ul style="list-style-type: none"> ▶ Minute-level
Cross Platform Transportable Tablespaces (XTTS)	<ul style="list-style-type: none"> ▶ Cross platform 	<ul style="list-style-type: none"> ▶ A bit higher technical skill requirements (+11.2.0.4) 	<ul style="list-style-type: none"> ▶ Compression is not supported 	<ul style="list-style-type: none"> ▶ Minute-level
Replication (OGG/CDC, etc)	<ul style="list-style-type: none"> ▶ Highly flexible 	<ul style="list-style-type: none"> ▶ Slightly poorer performance ▶ More data validation workload ▶ Additional software license costs 	<ul style="list-style-type: none"> ▶ Oracle BLOBs/CLOBs can sometimes have issues (verify with Oracle SR) 	<ul style="list-style-type: none"> ▶ Minute-level

Each migration methodology has advantages and disadvantages. For test/development, Oracle Data Pump is a popular choice, as it is simple and works across many platforms, including those with a different endian format. The disadvantages of Data Pump is that it requires more downtime than other approaches. Also, some data objects like BLOBs/CLOBs may not migrate with Data Pump.

On LinuxONE systems, you can take advantage of the IBM [zEnterprise® Data Compression \(zEDC\)](#) hardware accelerator to quickly compress export dumps and move them to other systems.

RMAN backup sets is a technique for migrating Oracle databases from other platforms (regardless of endian format), particularly if the source Oracle database version is 19.18 or greater. Oracle support note, [M5 Cross Endian Platform Migration using Full Transportable Data Pump Export/Import and RMAN Incremental Backups \(Doc ID 2999157.1\)](#) details all the steps needed.

The following are the high level steps involved:

- ▶ Perform a level 0 backup on the source
- ▶ Restore the L0 backup taken previously on the destination
- ▶ Perform incremental backups on source and apply on the destination. Repeat this procedure until final incremental is taken, which requires minimal read-only downtime.

Once Downtime Begins:

- ▶ Source Data Tablespaces converted to READ ONLY
- ▶ Final Incremental backup on source
- ▶ Metadata Tablespace Export on source
- ▶ Apply Incremental on destination.
- ▶ Metadata tablespace Import on destination.
- ▶ Downtime ends.

Cross Platform Transportable Tablespaces (XTTS) is another good methodology to migrate large amounts of data between systems including cross-platforms with different endian formats. Place a tablespace in read-only mode, copy the tablespace datafile to another system (ASM to ASM supported) and then re-enable read-write. This allows for migrating a production system to a dev/test system.

If an endian conversion is needed then, an RMAN command is needed to run the conversion. Most of the time needed for this migration approach is copying the data to the target system. One approach to reduce copy time is, if possible, unmount the migrated file system on the source system and remount the file system on the target.

Oracle Change Data Capture (CDC) and Oracle Goldengate (OGG) are also a good approach when close to zero downtime is needed or if a failback option is needed after some time of going live on a new system. Configuring the replication scripts, and ensuring things are in synch and highly available is one disadvantage to this approach.

With all these migration approaches, a good framework and project plan is paramount to ensure that each migration is tested and repeatable. The framework is important because it brings discipline and consistency to the project and helps minimize risk. Typically, we suggest the 5-step process shown in Figure 2-13.

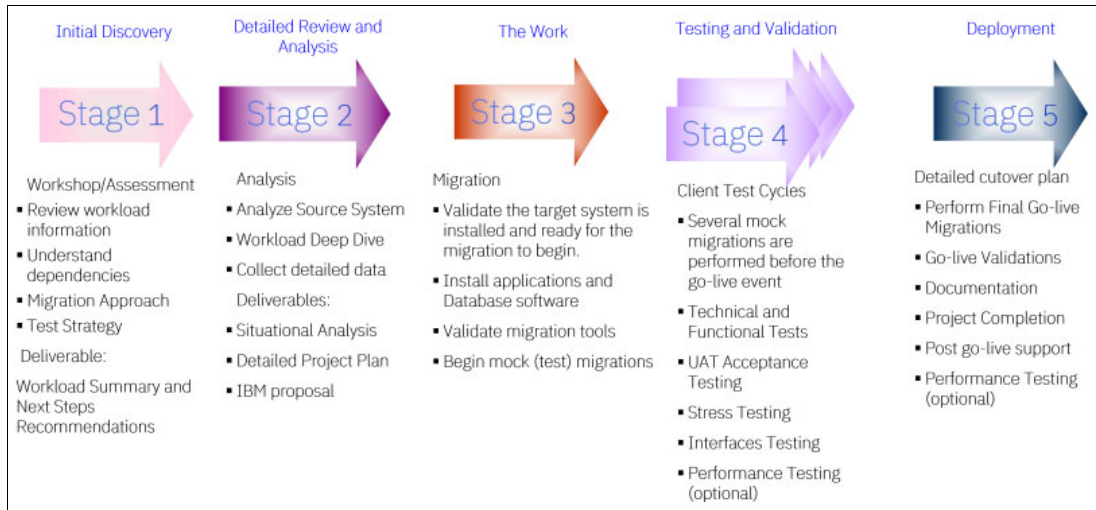


Figure 2-13 Migration Framework

The 5-step migration framework has the following stages:

1. Stage 1: Perform workshops and assessments for the initial discovery phase of the project.
2. Stage 2: Perform an analysis to select the deliverables and the best methodology for the migrations.
3. Stage 3: The migration process. This is where the initial migrations are tested and methodologies confirmed. Once that is done we install the applications and databases to be migrated, verify our migration tools and start the migration process.
4. Stage 4: The heart of the migration, whereby the migrations are tested to help eliminate any risk. For example, you should never modify the source databases during a migration project. Secondly, we isolate the migration from day-to-day operations. You can do this by using copies of the production databases and locate them on staging servers for most of the mock migrations.

This isolates the migration from the client's day-to-day operations. No outages are required on the client's production environment and there is no chance of the migration bringing down the business.

5. Stage 5: The final step is the go-live period and cut-over to the new IBM LinuxONE.



Db2 and LinuxONE

Db2 can be installed on LinuxONE to utilize the benefits of the LinuxONE platform.

LinuxONE is IBM's all-Linux enterprise platform for open innovation that combines the best of Linux and open technology with the best of enterprise computing in one system. The platform is designed with a focus on security, scalability, and performance to support customers who want an efficient and cost-effective solution to thrive in a data-centric economy.

LinuxONE's hardened Linux-based software stack can run most open-source software packages, such as databases and data management, virtualization platforms, and containers, automation and orchestration software, and compute-intensive workloads, such as blockchain.

You can use Db2 for Linux, UNIX, and Windows products on LinuxONE. It works seamlessly in the virtualized environment without any extra configuration. In addition, autonomic features, such as self-tuning memory management and enhanced automatic storage, help the database administrator to maintain and tune the Db2 server.

3.1 Benefits of migrating Db2 to LinuxONE

Db2 workloads that are running on LinuxONE benefit from a hardware platform that includes specialized processors, cryptographic cards with dedicated RISC processors, and a combination of hypervisors that allow flexibility in Linux deployment.

As mentioned in Chapter 1.5, “Example of the Benefits of LinuxONE in Financial Institutions” on page 9 a major benefit of Linux is that it is open source. The software is unencumbered by licensing fees and its source code is freely available. Hundreds of Linux distributions are available for almost every computing platform.

This multiplatform support allows customers to run a common operating system across all computing platforms, which means significantly lower support costs and, for Linux, no incremental license charges. It also offers customers the flexibility of easily moving applications to the most appropriate platform.

IBM LinuxONE delivers the best of enterprise Linux on the industry’s most reliable and highly scalable hardware. These systems are specialized scale-up enterprise servers that are designed exclusively to run Linux applications.

IBM LinuxONE provides the highest levels of availability (near 100 percent uptime with no single point of failure), performance, throughput, and security. End-to-end security is built in with isolation at each level in the stack and provides the highest level of certified security in the industry.

Additionally, LinuxONE Systems facilitate transparent use of redundant processor execution steps and integrity checking, which is necessary in financial services industries. LinuxONE servers typically enable hot swapping of hardware, such as processors and memory. This swapping is typically transparent to the operating system, enabling routine repairs to be performed without shutting down the system.

IBM LinuxONE delivers on the promise of a flexible, secure, and smart IT architecture that can be managed seamlessly to meet the requirements of today’s fast-changing business climate. LinuxONE provides the following benefits:

- ▶ Premium Linux experience with sub-seconds user response times and virtually unlimited scale.
- ▶ Broad portfolio of Open Source and other vendor products and tools delivered on the platform.
- ▶ Choice of Linux (RHEL, SUSE, and Ubuntu) and tools that best fit your environment.
- ▶ Eliminates risks by running Linux on industry’s most secure and resilient hardware platform.
- ▶ Easy integration of data and applications with existing IBM z Systems® based solutions.
- ▶ Overall increases the operational IT efficiency

3.2 Installing Db2 on LinuxONE

In this section, we discuss the steps we took to install Db2 on LinuxONE in our lab environment. From there, migration from one Db2 to another Db2 is fairly straightforward and is outlined in section 3.3, “Db2 migration to LinuxONE” on page 82.

3.2.1 Db2 install pre-requisites on LinuxONE

The following are the pre-requisites for an installation of Db2 on a LinuxONE machine (s390 64-bit).

1. Download the software.

Download the base software by either logging into the [IBM Passport Advantage®](#) website or [Universal Fix Pack from IBM Fix central](#) (Db2 products can be installed by using fix pack images).

Example 3-1 Download the fix pack binary

```
[root@rdbk1103 data]# ls -ltr
total 1494124
-rw-r--r--. 1 root root 1529980481 Jul 22 08:15
v11.5.9_linux390x64_universal_fixpack.tar.gz
```

2. Check the machine OS version and architecture by using the commands shown in Example 3-2.

Example 3-2 Check the machine OS version and architecture

```
[root@rdbk1103 data]# cat /etc/redhat-release
Red Hat Enterprise Linux release 9.4 (Plow)
[root@rdbk1103 data]# uname -a
Linux rdbk1103.cpolab.ibm.com 5.14.0-427.26.1.el9_4.s390x #1 SMP Fri Jul 5
11:39:09 EDT 2024 s390x s390x s390x GNU/Linux
```

3. Install the IBM DB2® pre-requisites by using the commands shown in Example 3-3.

Example 3-3 Decompress the Db2 executables and run the db2prereqcheck command

```
gunzip v11.5.9_linux390x64_universal_fixpack.tar.gz
tar -xvf 11.5.9_linux390x64_universal_fixpack.tar
cd universal
./db2prereqcheck
yum install patch -y
yum install gcc-c++ -y
yum install libxcrypt-compat -y
yum install perl -y
yum install ksh -y
yum install mash -y
yum install compat-openssl* -y
vi /etc/sysconfig/selinux
Change SELinux=enforcing to SELinux=disabled
reboot
```

4. Run the commands shown in Example 3-4 to verify the pre-requisites.

Example 3-4 Run the pre-requisite check

```
[root@rdbk1104 universal]# ./db2prereqcheck -v 11.5.9.0
=====
Wed Jul 24 02:29:57 2024
Checking prerequisites for DB2 installation. Version "11.5.9.0". Operating system
"Linux"

Validating "kernel level " ...
```

Required minimum operating system kernel level: "3.10.0".
Actual operating system kernel level: "5.14.0".
Requirement matched.

Validating "Linux distribution " ...
Required minimum operating system distribution: "RHEL"; Version: "9"; Service pack: "2".
Actual operating system distribution Version: "9"; Service pack: "4".
Requirement matched.

Validating "ksh symbolic link" ...
Requirement matched.

Validating "Bin user" ...
Requirement matched.

Validating "libxcrypt-compat" ...
Package (or file) found: "libxcrypt-compat"
Requirement matched.

Validating "C++ Library version " ...
Required minimum C++ library: "libstdc++.so.6"
Standard C++ library is located in the following directory:
"/usr/lib64/libstdc++.so.6.0.29".
Actual C++ library: "CXXABI_1.3.1"
Requirement matched.
Requirement matched.

Validating "libaio.so version " ...
DBT3553I The db2prereqcheck utility successfully loaded the libaio.so.1 file.
Requirement matched.

DBT3533I The db2prereqcheck utility has confirmed that all installation prerequisites were met.

=====
Wed Jul 24 02:29:57 2024

Checking prerequisites for DB2 installation with the DB2 pureScale Feature.
Version: "11.5.9.0". Operating system: "Linux".

Validating "kernel level " ...
Required minimum operating system kernel level: "3.10.0".
Actual operating system kernel level: "5.14.0".
Requirement matched.

Validating "Linux distribution " ...
Required minimum operating system distribution: "RHEL"; Version: "9"; Service pack: "2".
Actual operating system distribution Version: "9"; Service pack: "4".
Requirement matched.

Validating "ksh symbolic link" ...
Requirement matched.

Validating "Bin user" ...
Requirement matched.


```
Validating "libxcrypt-compat" ...
  Package (or file) found: "libxcrypt-compat"
  Requirement matched.

Validating "SELinux status " ...
  SELinux is "disabled ".
  Requirement matched.

Validating "libc.so version " ...
  glibc library is located in the following directory "/usr/lib64/libc.so.6".
  Required minimum glibc library version: "2.4.0"
  Actual glibc library version: "2.34.0"
  Requirement matched.

Validating "gcc" ...
  Package (or file) found: "gcc"
  Requirement matched.

Validating "binutils" ...
  Package (or file) found: "binutils"
  Requirement matched.

Validating "cpp" ...
  Package (or file) found: "cpp"
  Requirement matched.

Validating "gcc-c++" ...
  Package (or file) found: "gcc-c++"
  Requirement matched.

Validating "net-tools" ...
  Package (or file) found: "net-tools"
  Requirement matched.

Validating "m4" ...
  Package (or file) found: "m4"
  Requirement matched.

Validating "NetworkManager-config-server" ...
  Package (or file) found: "NetworkManager-config-server"
  Requirement matched.

Validating "kernel-devel" ...
  Package (or file) found: "kernel-devel"
  Package "kernel-devel" level "5.14.0-427.26.1.el9_4.s390x" match with the
  system "kernel" level "5.14.0-427.26.1.el9_4.s390x".
  Requirement matched.

Validating "kernel-headers" ...
  Package (or file) found: "kernel-headers"
  Package "kernel-headers" level "5.14.0-427.26.1.el9_4.s390x" match with the
  system "kernel" level "5.14.0-427.26.1.el9_4.s390x".
  Requirement matched.

Validating "file" ...
```

```
Package (or file) found: "file"
Requirement matched.

Validating "perl" ...
Package (or file) found: "perl"
Requirement matched.

Validating "libgomp" ...
Package (or file) found: "libgomp"
Requirement matched.

Validating "make" ...
Package (or file) found: "make"
Requirement matched.

Validating "libgcc" ...
Package (or file) found: "libgcc"
Requirement matched.

Validating "patch" ...
Package (or file) found: "patch"
Requirement matched.

Validating "elfutils-libelf-devel" ...
Package (or file) found: "elfutils-libelf-devel"
Requirement matched.

Validating "python36 link" ...
Found package "python3" on host "rdbk1104.cpolab.ibm.com".
Requirement matched.

Validating "ksh" ...
Required minimum version for "ksh": "1.0.0~beta.1"
Actual version of package: "1.0.6"
Requirement matched.

Validating "/usr/sbin/chronyd" ...
Required minimum version for "/usr/sbin/chronyd": "2.2.1"
Actual version of package: "4.5"
Requirement matched.

Validating "prereqSAM" ...
Requirement matched.

Validating "/boot/loader/entries conf files" ...
DBT3615E The db2prereqcheck utility was unable to validate the configuration of
the vmalloc kernel parameter on the following host machine:
"rdbk1104.cpolab.ibm.com". Reason code: "1".
ERROR : Requirement not matched.

Validating "free space" ...
The directory "/tmp" has enough space on host "rdbk1104.cpolab.ibm.com".
Requirement matched.

Validating "free space" ...
```

The directory "/var" has enough space on host "rdbk1104.cpolab.ibm.com".
Requirement matched.

Validating "free space" ...
The directory "/usr" has enough space on host "rdbk1104.cpolab.ibm.com".
Requirement matched.

Validating "sg3_utils" ...
Package (or file) found: "sg3_utils"
Requirement matched.

Validating "sg_persist" ...
Package (or file) found: "/usr/bin/sg_persist"
Requirement matched.

Requirement not matched for DB2 database "Server" with pureScale feature .
Version: "11.5.9.0".

Summary of prerequisites that are not met on the current system:
DBT3615E The db2prereqcheck utility was unable to validate the configuration of
the vmalloc kernel parameter on the following host machine:
"rdbk1104.cpolab.ibm.com". Reason code: "1".

3.2.2 Db2 installation on LinuxONE

The following provides the steps on how to install Db2 on LinuxONE.

1. Modify /etc/hosts as shown in Example 3-5, using your hostnames and IP addresses. Normally, resolving the local server's host name is handled by /etc/hosts and not by a domain name server (DNS). Hostname lookups will first use local files (/etc/hosts), and then resort to DNS, so you want to make sure your /etc/hosts file is modified accordingly.

Example 3-5 Make proper /etc/hosts entries

```
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1        localhost localhost.localdomain localhost6 localhost6.localdomain6
9.76.61.102 rdbk1103 rdbk1103.cpolab.ibm.com
9.76.61.103 rdbk1104 rdbk1104.cpolab.ibm.com
```

2. Before installing Db2, create the required instance USERID and fence USERID, following the example shown in Example 3-6.

Example 3-6 Create required instance and fence users and groups

```
groupadd db2grp1
groupadd db2fgrp1
useradd db2inst1
useradd db2fenc1
usermod --append --groups db2fgrp1 db2fenc1
usermod --append --groups db2grp1 db2inst1
[root@rdbk1104 universal]# cat /etc/passwd|grep -i db2
db2inst1:x:1001:1003::/home/db2inst1:/bin/bash
db2fenc1:x:1002:1004::/home/db2fenc1:/bin/bash
[root@rdbk1104 universal]# cat /etc/group|grep -i db2
db2grp1:x:1001:db2inst1
db2fgrp1:x:1002:db2fenc1
```

3. Install the Db2 Server Edition using the `db2_install` command, as shown in Example 3-7.

Example 3-7 Install the Db2 server

```
[root@rdbk1104 universal]# ./db2_install -y
Default directory for installation of products - /opt/ibm/db2/V11.5
*****
Install into default directory (/opt/ibm/db2/V11.5) ? [yes/no]
No
Enter the full path of the base installation directory:
-----
/data/db2_binary
Specify one of the following keywords to install DB2 products.
  SERVER
  CONSV
  CLIENT
  RTCL
Enter "help" to redisplay product names.
Enter "quit" to exit.
*****
SERVER
*****
Do you want to install the DB2 pureScale Feature? [yes/no]
No
DB2 installation is being initialized.
Total number of tasks to be performed: 56
Total estimated time for all tasks to be performed: 2430 second(s)
Task #1 start
Description: Checking license agreement acceptance
Estimated time 1 second(s)
Task #1 end
Task #2 start
Description: Base Client Support for installation with root privileges
Estimated time 3 second(s)
Task #2 end
Task #3 start
Description: Product Messages - English
Estimated time 14 second(s)
Task #3 end
Task #4 start
Description: Base client support
Estimated time 282 second(s)
Task #4 end
Task #5 start
Description: Java Runtime Support
Estimated time 228 second(s)
Task #5 end
Task #6 start
Description: Java Help (HTML) - English
Estimated time 7 second(s)
Task #6 end
Task #7 start
Description: Base server support for installation with root privileges
Estimated time 6 second(s)
Task #7 end
Task #8 start
```

Description: Global Secure ToolKit
Estimated time 43 second(s)
Task #8 end
Task #9 start
Description: Java support
Estimated time 11 second(s)
Task #9 end
Task #10 start
Description: SQL procedures
Estimated time 3 second(s)
Task #10 end
Task #11 start
Description: ICU Utilities
Estimated time 60 second(s)
Task #11 end
Task #12 start
Description: Java Common files
Estimated time 17 second(s)
Task #12 end
Task #13 start
Description: Base server support
Estimated time 564 second(s)
Task #13 end
Task #14 start
Description: Relational wrappers common
Estimated time 3 second(s)
Task #14 end
Task #15 start
Description: DB2 data source support
Estimated time 7 second(s)
Task #15 end
Task #16 start
Description: ODBC data source support
Estimated time 161 second(s)
Task #16 end
Task #17 start
Description: Teradata data source support
Estimated time 3 second(s)
Task #17 end
Task #18 start
Description: Spatial Extender server support
Estimated time 22 second(s)
Task #18 end
Task #19 start
Description: Scientific Data Sources
Estimated time 5 second(s)
Task #19 end
Task #20 start
Description: JDBC data source support
Estimated time 126 second(s)
Task #20 end
Task #21 start
Description: IBM Software Development Kit (SDK) for Java(TM)
Estimated time 42 second(s)
Task #21 end

Task #22 start
Description: DB2 LDAP support
Estimated time 3 second(s)
Task #22 end
Task #23 start
Description: DB2 Instance Setup wizard
Estimated time 23 second(s)
Task #23 end
Task #24 start
Description: Structured file data sources
Estimated time 5 second(s)
Task #24 end
Task #25 start
Description: Oracle data source support
Estimated time 4 second(s)
Task #25 end
Task #26 start
Description: Connect support
Estimated time 3 second(s)
Task #26 end
Task #27 start
Description: Application data sources
Estimated time 4 second(s)
Task #27 end
Task #28 start
Description: Spatial Extender client
Estimated time 3 second(s)
Task #28 end
Task #29 start
Description: SQL Server data source support
Estimated time 4 second(s)
Task #29 end
Task #30 start
Description: Communication support - TCP/IP
Estimated time 3 second(s)
Task #30 end
Task #31 start
Description: Tivoli SA MP
Estimated time 300 second(s)
Task #31 end
Task #32 start
Description: Base application development tools
Estimated time 32 second(s)
Task #32 end
Task #33 start
Description: Parallel Extension
Estimated time 3 second(s)
Task #33 end
Task #34 start
Description: EnterpriseDB code
Estimated time 3 second(s)
Task #34 end
Task #35 start
Description: Replication tools
Estimated time 60 second(s)

Task #35 end
Task #36 start
Description: Sample database source
Estimated time 4 second(s)
Task #36 end
Task #37 start
Description: itlm
Estimated time 3 second(s)
Task #37 end
Task #38 start
Description: DB2 Text Search
Estimated time 109 second(s)
Task #38 end
Task #39 start
Description: Command Line Processor Plus
Estimated time 7 second(s)
Task #39 end
Task #40 start
Description: Informix data source support
Estimated time 4 second(s)
Task #40 end
Task #41 start
Description: Federated Data Access Support
Estimated time 3 second(s)
Task #41 end
Task #42 start
Description: First Steps
Estimated time 3 second(s)
Task #42 end
Task #43 start
Description: Pacemaker
Estimated time 100 second(s)
Task #43 end
Task #44 start
Description: Product Signature for DB2 Server Edition
Estimated time 8 second(s)
Task #44 end
Task #45 start
Description: Guardium Installation Manager Client
Estimated time 3 second(s)
Task #45 end
Task #46 start
Description: Setting DB2 library path
Estimated time 180 second(s)
Task #46 end
Task #47 start
Description: Installing or updating DB2 HA scripts for IBM Tivoli System
Automation for Multiplatforms (Tivoli SA MP)
Estimated time 40 second(s)
Task #47 end
Task #48 start
Description: Installing or updating Db2 resource agent scripts for Pacemaker
Estimated time 20 second(s)
Task #48 end
Task #49 start

```

Description: Executing control tasks
Estimated time 20 second(s)
Task #49 end
Task #50 start
Description: Updating global registry
Estimated time 20 second(s)
Task #50 end
Task #51 start
Description: Starting DB2 Fault Monitor
Estimated time 10 second(s)
Unit db2fmcd.service could not be found.
Task #51 end
Task #52 start
Description: Updating the db2ls and db2greg link
Estimated time 1 second(s)
Task #52 end
Task #53 start
Description: Registering DB2 licenses
Estimated time 5 second(s)
Task #53 end
Task #54 start
Description: Setting default global profile registry variables
Estimated time 1 second(s)
Task #54 end
Task #55 start
Description: Initializing instance list
Estimated time 5 second(s)
Task #55 end
Task #56 start
Description: Updating global profile registry
Estimated time 3 second(s)
Task #56 end
The execution completed with warnings.
For more information see the DB2 installation log at
"/tmp/db2_install.log.22797".
[root@rdbk1104 universal]#

```

4. Create the Db2 instance as shown in Example 3-8.

Example 3-8 Create Db2 instance

```

cd $INSTALL_PATH/instance
[root@rdbk1103 instance]# ./db2icrt -u db2fenc1 db2inst1
DBI1446I The db2icrt command is running.
DB2 installation is being initialized.
Total number of tasks to be performed: 4
Total estimated time for all tasks to be performed: 309 second(s)
Task #1 start
Description: Setting default global profile registry variables
Estimated time 1 second(s)
Task #1 end
Task #2 start
Description: Initializing instance list
Estimated time 5 second(s)
Task #2 end
Task #3 start

```


Description: Configuring DB2 instances
 Estimated time 300 second(s)
 Task #3 end
 Task #4 start
 Description: Updating global profile registry
 Estimated time 3 second(s)
 Task #4 end
 The execution completed successfully.
 For more information see the DB2 installation log at `"/tmp/db2icrt.log.382674"`.
 DBI1070I Program db2icrt completed successfully.

Setting up the Db2 registry profile variables.

Example 3-9 shows how we set up Db2 in our lab environment and set our registry profile variables.

- ▶ The DB2COMM registry variable allows you to set communication protocols for the current Db2 instance. If the DB2COMM registry variable is undefined or set to null, no protocol connection managers are started.
- ▶ The DB2AUTOSTART configuration parameter can restart the Db2 process if Db2 was shutdown accidentally.
- ▶ The DB2SYSTEM configuration parameter indicates that Db2 is installed and on which hosts.
- ▶ DB2INSTDEF is the default instance variable that is specific to the current Db2 copy in use.

Note that in Example 3-9, we set the DB2COMM variable to TCPIP and used the `db2set -a11` command to display all defined variables in all registry levels. In our example, the [i] means that the variable is set on the instance level and the [g] means that the variable is set globally.

Example 3-9 Add Db2 registry profile variables.

```
db2inst1@rdbk1103:~> db2set DB2COMM=tcpip
db2inst1@rdbk1103:~> db2set -a11
[i] DB2COMM=TCPIP
[i] DB2AUTOSTART=NO
[g] DB2SYSTEM=rdbk1103
[g] DB2INSTDEF=db2inst1
```

Update the services file and specify the ports that you want the server to listen on for incoming client requests. If you want to make changes to the `/etc/services` file, the instance must be fully offline, and you must change the `/etc/services` file on all hosts in the cluster. Example 3-10 shows our Linux port configuration to ensure that the Db2 instance manager is able to run. You would use a text editor to add the connection entries to the services file.

From our example, `db2c_db2inst1` represents the connection service name, `20016` represents the connection port number and `tcp` represents the communication protocol that you are using. `DB2_db2inst1_END20021/tcp` indicates that this is a port range, which we have set up to enable the communication between Db2 partitions using fast communication manager (FCM). By default, the first port (20016) is reserved for client to server connection requests, and the first available four ports above 20021 are reserved for FCM communication.

Example 3-10 Add Linux ports

```
[root@rdbk1103 ~]# cat /etc/services | grep db2
```

```
DB2_db2inst120016/tcp # Db2 connection service port
DB2_db2inst1_120017/tcp
DB2_db2inst1_220018/tcp
DB2_db2inst1_320019/tcp
DB2_db2inst1_420020/tcp
DB2_db2inst1_END20021/tcp
```

Next, you will need to change the Linux kernel parameters based on the memory in the machine. For more information on kernel parameter requirements on Linux, see: <https://www.ibm.com/docs/en/db2/11.5?topic=unix-kernel-parameter-requirements-linux>

Our kernel parameters are shown in Example 3-11.

Example 3-11 Kernel parameters for a machine with 32 GB RAM

```
vi /etc/sysctl.conf
kernel.shmni=8192
kernel.shmmax=34359738368
kernel.shmall=8388608
kernel.semmsl=250
kernel.semni=8192
kernel.semms=256000
kernel.semopm=32
kernel.sem=32767
kernel.msgmni=32768
kernel.msgmax=65536
kernel.msgmnb=65536
```

3.2.3 Db2 DPF install on LinuxONE

Most of the Db2 install steps and pre-requisites to use the Db2 Database Partitioning Feature (DPF) on LinuxONE are similar to those listed in section 3.2.2, “Db2 installation on LinuxONE” on page 51. You will need to repeat the same steps on one or more Db2 systems. Also, you will require a common mount point to be shared between the two or more servers by using either the GPFS or NFS technology. In our lab environment, we used the NFS mount point between two servers.

In our lab environment, we installed Db2 on two systems - rdbkl03 and rdbkl04. These will be a part of the single Db2 DPF cluster instance.

We set up the Db2 NFS server between the two nodes (with rdbkl03 as the NFS server, also known as the coordinator node, and rdbkl04 as the NFS client) following the instructions found at:

<https://earihos.medium.com/how-to-install-and-configure-nfs-storage-server-in-red-hat-enterprise-linux-9-389379cbb4ee>

The NFS mount can be automated to failover from one machine to the other by using TSA software. To do this, we followed the instructions found in Chapter 2 of the IBM Redbooks publication, *High Availability and Disaster Recovery Options for DB2 for Linux, UNIX, and Windows, SG24-7363*.

We changed our Linux kernel parameters based on the memory of our IBM LinuxONE. The minimum kernel parameter requirements for Linux can be found on the following website: <https://www.ibm.com/docs/en/db2/11.5?topic=unix-kernel-parameter-requirements-linux>

You could also install GPFS sharing if needed but we did not do that for our example.

The install command and our output are shown in Example 3-12.

Example 3-12 Command and output of NFS installation

```
[root@rdbk1103 ~]# dnf install nfs-utils
Updating Subscription Management repositories.
```

This system is registered with an entitlement server, but is not receiving updates. You can use subscription-manager to assign subscriptions.

Last metadata expiration check: 2:52:55 ago on Wed 07 Aug 2024 11:51:10 PM EDT.
Dependencies resolved.

```
=====
Package
Architecture                               Version
Repository
Size
=====
Installing:
nfs-utils                                   s390x
1:2.5.4-25.e19
rhel-9-for-s390x-baseos-rpms
455 k
Installing dependencies:
gssproxy                                   s390x
0.8.4-6.e19
rhel-9-for-s390x-baseos-rpms
109 k
libev                                       s390x
4.33-5.e19
rhel-9-for-s390x-baseos-rpms
54 k
libnfsidmap                                s390x
1:2.5.4-25.e19
rhel-9-for-s390x-baseos-rpms
64 k
libverto-libev                             s390x
0.3.2-3.e19
rhel-9-for-s390x-baseos-rpms
15 k
rpcbind                                    s390x
1.2.6-7.e19
rhel-9-for-s390x-baseos-rpms
60 k
sssd-nfs-idmap                              s390x
2.9.4-6.e19_4
rhel-9-for-s390x-baseos-rpms
46 k
=====
```

Transaction Summary

=====

Install 7 Packages

```
Total download size: 804 k
Installed size: 1.8 M
Is this ok [y/N]: y
Downloading Packages:
(1/7):libverto-libev-0.3.2-3.e19.s390x.rpm
63 kB/s | 15 kB    00:00
(2/7):gssproxy-0.8.4-6.e19.s390x.rpm
397 kB/s | 109 kB  00:00
(3/7):libev-4.33-5.e19.s390x.rpm
114 kB/s | 54 kB   00:00
(4/7):libnfsidmap-2.5.4-25.e19.s390x.rpm
178 kB/s | 64 kB   00:00
(5/7):rpcbind-1.2.6-7.e19.s390x.rpm
122 kB/s | 60 kB   00:00
(6/7):sssd-nfs-idmap-2.9.4-6.e19_4.s390x.rpm
89 kB/s | 46 kB    00:00
(7/7):nfs-utils-2.5.4-25.e19.s390x.rpm
499 kB/s | 455 kB  00:00
```

```
Total
675 kB/s | 804 kB    00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing  :
 1/1
Installing :libnfsidmap-1:2.5.4-25.e19.s390x
 1/7
Runningscriptlet:rpcbind-1.2.6-7.e19.s390x
 2/7
Installing :rpcbind-1.2.6-7.e19.s390x
 2/7
Runningscriptlet:rpcbind-1.2.6-7.e19.s390x
 2/7
Created symlink /etc/systemd/system/multi-user.target.wants/rpcbind.service ?
/usr/lib/systemd/system/rpcbind.service.
Created symlink /etc/systemd/system/sockets.target.wants/rpcbind.socket ?
/usr/lib/systemd/system/rpcbind.socket.

Installing :libev-4.33-5.e19.s390x
 3/7
Installing :libverto-libev-0.3.2-3.e19.s390x
 4/7
Installing :gssproxy-0.8.4-6.e19.s390x
 5/7
Runningscriptlet:gssproxy-0.8.4-6.e19.s390x
 5/7
Runningscriptlet:nfs-utils-1:2.5.4-25.e19.s390x
 6/7
```

```

Installing :nfs-utils-1:2.5.4-25.e19.s390x
6/7
Runningscriptlet:nfs-utils-1:2.5.4-25.e19.s390x
6/7
Installing :sssd-nfs-idmap-2.9.4-6.e19_4.s390x
7/7
Runningscriptlet:sssd-nfs-idmap-2.9.4-6.e19_4.s390x
7/7
Verifying  :libev-4.33-5.e19.s390x
1/7
Verifying  :libverto-libev-0.3.2-3.e19.s390x
2/7
Verifying  :gssproxy-0.8.4-6.e19.s390x
3/7
Verifying  :libnfsidmap-1:2.5.4-25.e19.s390x
4/7
Verifying  :nfs-utils-1:2.5.4-25.e19.s390x
5/7
Verifying  :rpcbind-1.2.6-7.e19.s390x
6/7
Verifying  :sssd-nfs-idmap-2.9.4-6.e19_4.s390x
7/7
Installed products updated.

```

Installed:

```

gssproxy-0.8.4-6.e19.s390x      libev-4.33-5.e19.s390x
libnfsidmap-1:2.5.4-25.e19.s390x  libverto-libev-0.3.2-3.e19.s390x
nfs-utils-1:2.5.4-25.e19.s390x    rpcbind-1.2.6-7.e19.s390x
sssd-nfs-idmap-2.9.4-6.e19_4.s390x

```

Complete!

We used the commands shown in Example 3-13 to start the RPC bind services on Red Hat Enterprise Linux (RHEL).

Example 3-13 Start RPC bind services

```

[root@rdbk1103 ~]# systemctl start rpcbind
[root@rdbk1103 ~]# systemctl status rpcbind.service
? rpcbind.service - RPC Bind
   Loaded: loaded (/usr/lib/systemd/system/rpcbind.service; enabled; preset:
enabled)
   Active: active (running) since Thu 2024-08-08 02:47:29 EDT; 11s ago
 TriggeredBy: ? rpcbind.socket
   Docs: man:rpcbind(8)
  Main PID: 1452710 (rpcbind)
    Tasks: 1 (limit: 203912)
   Memory: 860.0K
     CPU: 16ms
   CGroup: /system.slice/rpcbind.service
           ??1452710 /usr/bin/rpcbind -w -f

```

```

Aug 08 02:47:29 rdbk1103.cpolab.ibm.com systemd[1]: Starting RPC Bind...
Aug 08 02:47:29 rdbk1103.cpolab.ibm.com systemd[1]: Started RPC Bind.

```

We use the command found in Example 3-14 to start the NFS server.

Example 3-14 Start NFS server services

```
[root@rdbk1103 ~]# systemctl enable nfs-server.service
Created symlink /etc/systemd/system/multi-user.target.wants/nfs-server.service ?
/usr/lib/systemd/system/nfs-server.service.
[root@rdbk1103 ~]# systemctl start nfs-server.service
[root@rdbk1103 ~]# systemctl status nfs-server.service
? nfs-server.service - NFS server and services
   Loaded: loaded (/usr/lib/systemd/system/nfs-server.service; enabled; preset:
disabled)
   Active: active (exited) since Thu 2024-08-08 02:49:22 EDT; 7s ago
     Docs: man:rpc.nfsd(8)
          man:exportfs(8)
   Process: 1452854 ExecStartPre=/usr/sbin/exportfs -r (code=exited,
status=0/SUCCESS)
   Process: 1452855 ExecStart=/usr/sbin/rpc.nfsd (code=exited, status=0/SUCCESS)
   Process: 1452877 ExecStart=/bin/sh -c if systemctl -q is-active gssproxy; then
systemctl reload gssproxy ; fi (code=exited, status=0/SUCCESS)
   Main PID: 1452877 (code=exited, status=0/SUCCESS)
     CPU: 11ms

Aug 08 02:49:22 rdbk1103.cpolab.ibm.com systemd[1]: Starting NFS server and
services...
Aug 08 02:49:22 rdbk1103.cpolab.ibm.com systemd[1]: Finished NFS server and
services.
```

We used the commands found in Example 3-15 to create the NFS folder and give the mount folder permissions.

Example 3-15 Create NFS folder and set permissions

```
[root@rdbk1103 ~]# mkdir /nfs_storage
[root@rdbk1103 ~]# chmod 777 -R /nfs_storage/
```

Example 3-16 provides an example of the commands we used to export the NFS to our client, stop the firewall and check the mount list.

Example 3-16 Export NFS to client, stop the firewall and review mount information

```
[root@rdbk1103 ~]# cat <<EOF | sudo tee -a /etc/exports
> /nfs_storage
9.76.61.0/24(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)
EOF
[root@rdbk1103 ~]# mkdir /db2home
[root@rdbk1103 ~]# chmod 777 /db2home

[root@rdbk1103 ~]# cat /etc/exports
/nfs_storage
9.76.61.0/24(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)

[root@rdbk1103 ~]# exportfs -rav
exporting 9.76.61.0/24:/nfs_storage
[root@rdbk1103 ~]# showmount -e
```

```
Export list for rdbk1103.cpolab.ibm.com:
/nfs_storage 9.76.61.0/24
```

```
[root@rdbk1103 ~]# systemctl stop firewalld
[root@rdbk1103 ~]# systemctl disable firewalld
Removed "/etc/systemd/system/multi-user.target.wants/firewalld.service".
Removed "/etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service".
```

```
[root@rdbk1104 ~]# mount -t nfs4 9.76.61.102:/nfs_storage /db2home/
[root@rdbk1104 ~]# df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
devtmpfs	4.0M	0	4.0M	0%	/dev
tmpfs	16G	4.0K	16G	1%	/dev/shm
tmpfs	6.3G	17M	6.3G	1%	/run
/dev/dasda2	102G	7.8G	94G	8%	/
/dev/dasda1	456M	284M	173M	63%	/boot
/dev/mapper/rhel_rdbk1104-data	306G	6.9G	299G	3%	/data
tmpfs	3.2G	88K	3.2G	1%	/run/user/0
9.76.61.102:/nfs_storage	102G	8.4G	94G	9%	/db2home/

```
vi /etc/fstab
```

```
9.76.61.102:/nfs_storage /db2home/ nfs defaults 0 0
systemctl daemon-reload
```

Create the Db2 instance to use this NFS folder as a shared directory by using the commands shown in Example 3-17.

Example 3-17 Create the instance using the NFS directory and set the permissions

```
[root@rdbk1103 ~]# mkdir /db2home/db2inst2
[root@rdbk1103 ~]# chmod 777 /db2home/db2inst2
[root@rdbk1104 ~]# mkdir /db2home/db2inst2
[root@rdbk1104 ~]# chmod 777 /db2home/db2inst2
```

Create users db2inst2 and db2fenc2 on both machines with the same user ID (UID) and group ID (GID). Run the commands on both machines, in our case, rdbk1103 and rdbk1104, as shown in Example 3-18.

Example 3-18 Create user IDs and group IDs

```
groupadd -g 1999 db2igrp2
groupadd -g 1998 db2fgrp2

useradd -u 2004 -g db2igrp2 -m -d /db2home/db2inst2 db2inst2
useradd -u 2003 -g db2fgrp2 -m -d /db2home/db2fenc2 db2fenc2
```

```
[root@rdbk1103 ~]# cat /etc/passwd|grep -i db2
db2inst1:x:1001:1003::/home/db2inst1:/bin/bash
db2fenc1:x:1002:1004::/home/db2fenc1:/bin/bash
db2inst2:x:2004:1999::/db2home/db2inst2:/bin/bash
db2fenc2:x:2003:1998::/db2home/db2fenc2:/bin/bash
```

```
[root@rdbk1104 ~]# cat /etc/passwd|grep -i db2
db2inst1:x:1001:1003::/home/db2inst1:/bin/bash
db2fenc1:x:1002:1004::/home/db2fenc1:/bin/bash
db2inst2:x:2004:1999::/db2home/db2inst2:/bin/bash
```

```
db2fenc2:x:2003:1998:./db2home/db2fenc2:/bin/bash
```

```
[root@rdbk1103 ~]# cat /etc/group|grep -i db2
db2grp1:x:1001:db2inst1
db2fgrp1:x:1002:db2fenc1
```

```
db2igrp2:x:1999:db2inst2
db2fgrp2:x:1998:db2fenc2
```

```
[root@rdbk1104 ~]# cat /etc/group|grep -i db2
db2grp1:x:1001:db2inst1
db2fgrp1:x:1002:db2fenc1
```

```
db2igrp2:x:1999:db2inst2
db2fgrp2:x:1998:db2fenc2
```

Verify the Db2 install and list of which products are installed, as shown in Example 3-19.

Example 3-19 Verify Db2 install

```
[root@rdbk1103 universal]# ./db2ls
Install Path          Level  Fix Pack  Special Install Number
Install Date          Installer UID
-----
/data/db2_binary      11.5.9.0    0                Wed
Jul 24 02:42:42 2024 EDT    0
[root@rdbk1103 universal]# cd /data/db2_binary/instance

[root@rdbk1104 universal]# ./db2ls
Install Path          Level  Fix Pack  Special Install Number
Install Date          Installer UID
-----
/data/db2_binary      11.5.9.0    0                Wed
Jul 24 02:42:42 2024 EDT    0
[root@rdbk1104 universal]# cd /data/db2_binary/instance
```

Create the Db2 instance and use /db2home/db2inst2 as the Db2 SQLLIB directory on both servers. In our example, once we logged into the rdbk1103 machine as root, we created the db2inst2 instance in the /db2home/db2inst2 directory. For more details on setting up a partitioned database environment, see the following website:

<https://www.ibm.com/docs/en/db2/11.5?topic=environment-setting-up-partitioned-database>

Example 3-20 shows logging in as root to our lab environment, and creating the instance in the /db2home/db2inst2 directory.

Example 3-20 Create the instance

```
[root@rdbk1103 instance]# ./db2icrt -u db2fenc2 db2inst2
DBI1446I The db2icrt command is running.
```

```
DB2 installation is being initialized.
```

```
Total number of tasks to be performed: 4
```

```
Total estimated time for all tasks to be performed: 309 second(s)
```


Task #1 start
 Description: Setting default global profile registry variables
 Estimated time 1 second(s)
 Task #1 end

Task #2 start
 Description: Initializing instance list
 Estimated time 5 second(s)
 Task #2 end
 Task #3 start
 Description: Configuring DB2 instances
 Estimated time 300 second(s)
 Task #3 end

Task #4 start
 Description: Updating global profile registry
 Estimated time 3 second(s)
 Task #4 end

The execution completed successfully.

For more information see the DB2 installation log at
 "/tmp/db2icrt.log.1999281".
 DBI1070I Program db2icrt completed successfully.

Edit the /etc/services file on both machines to include the Db2 DPF instance, as shown in Example 3-21.

Example 3-21 Edit the services file

```
[root@rdbk1104 instance]# vi /etc/services
DB2_db2inst2      20022/tcp
DB2_db2inst2_1   20023/tcp
DB2_db2inst2_2   20024/tcp
DB2_db2inst2_3   20025/tcp
DB2_db2inst2_4   20026/tcp
DB2_db2inst2_END 20027/tcp
db2c_db2inst2    25011/tcp
```

The node configuration file (db2nodes.cfg), located in the instance owner's home directory, contains configuration information that tells the Db2 database system which servers participate in an instance of the partitioned database environment.

Edit the db2nodes.cfg file to indicate that both servers are a part the Db2 DPF instance. Example 3-22 provides an example of doing this in our lab environment.

Example 3-22 Configure db2nodes.cfg

```
[root@rdbk1103 instance]# su - db2inst2

[db2inst2@rdbk1104 ~]$ vi sqllib/db2nodes.cfg

0 rdbk1103.cpolab.ibm.com 0
1 rdbk1103.cpolab.ibm.com 1
2 rdbk1104.cpolab.ibm.com 0
```

3 rdbk1104.cpolab.ibm.com 1

Example 3-23 shows starting the Db2 instance from the coordinator node, in our lab environment, the NFS server, rdbk1103, of the Db2 DPF instance.

Example 3-23 Start Db2 from rdbk1103

```
[root@rdbk1103 instance]# su - db2inst2
[db2inst2@rdbk1103 sql1lib]$ db2start
db2inst2@rdbk1103's password:
db2inst2@rdbk1103's password:
db2inst2@rdbk1104's password:
db2inst2@rdbk1104's password:
08/13/2024 06:43:50    0  0  SQL1063N  DB2START processing was successful.
08/13/2024 06:43:50    1  0  SQL1063N  DB2START processing was successful.
08/13/2024 06:43:52    3  0  SQL1063N  DB2START processing was successful.
08/13/2024 06:43:52    2  0  SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
```

From the coordinator node (rdbk1103), we check the database configuration to ensure it is using the correct default database path(Example 3-24).

Example 3-24 Verify default database path

```
[root@rdbk1103 instance]# su - db2inst2
[db2inst2@rdbk1103 sql1lib]$ db2 get dbm cfg |grep DFTDBPATH
Default database path                (DFTDBPATH) = /db2home/db2inst2
```

Next, from the coordinator node (rdbk1103), we create the Db2 database, verify the files in the default database path and test our connection to the newly created database, as shown in Example 3-25.

Example 3-25 Create database

```
[db2inst2@rdbk1103 ~]$ db2 create database TEST AUTOMATIC STORAGE YES ON
/db2home/db2inst2 DBPATH ON /db2home/db2inst2
DB20000I The CREATE DATABASE command completed successfully.
```

```
[db2inst2@rdbk1104 ~]$ ls -l /db2home/db2inst2/db2inst2/
total 0
drwxrwxr-x 5 db2inst2 db2igrp2 50 Aug 13 06:59 NODE0000
drwxrwxr-x 5 db2inst2 db2igrp2 50 Aug 13 06:59 NODE0001
drwxrwxr-x 5 db2inst2 db2igrp2 50 Aug 13 06:59 NODE0002
drwxrwxr-x 5 db2inst2 db2igrp2 50 Aug 13 06:59 NODE0003
```

```
[db2inst2@rdbk1104 ~]$ db2 connect to TEST
```

Database Connection Information

```
Database server      = DB2/LINUXZ64 11.5.9.0
SQL authorization ID = DB2INST2
Local database alias = TEST
```

3.2.4 Db2 HADR on LinuxONE

High Availability Disaster Recovery (HADR) provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the primary database, to the target databases, called the standby databases. The HADR feature provides a high availability solution for both partial and complete site failures. HADR protects against data loss by replicating data changes from a source database, called the primary database, to one or more target databases, called the standby databases.

For more information on Db2 HADR on LinuxONE, see chapters 5,6,7,8 in the following IBM Redbooks publication: [High Availability and Disaster Recovery Options for DB2 for Linux, UNIX, and Windows, SG24-7363](#).

3.2.5 Db2 install on Red Hat OpenShift on LinuxONE

IBM announced the availability of Red Hat OpenShift on LinuxONE in 2020, making the integration of applications running on LinuxONE with applications running elsewhere in the cloud much easier. Red Hat OpenShift supports cloud-native applications being built once and deployed anywhere. Application developers and operations teams use Red Hat OpenShift to easily and efficiently manage their container environment.

The minimum Red Hat OpenShift architecture consists of five Linux guests (bootstrap, three control nodes, and one worker node) that are deployed on top of IBM z/VM 7.1 or later. Red Hat OpenShift on LinuxONE users can use the IBM Cloud® Infrastructure Center to manage the underlying cluster infrastructure.

IBM Db2 Universal container database

IBM Db2 Universal (Db2U) is a containerized version of IBM Db2 which has been designed for deployment in cloud and on-premises environments. Db2U is a reference to containerized deployment of Db2 and it is still the same Db2. The following are some key features and aspects of IBM Db2U:

- ▶ Containerized database

Db2U is packaged in containers, making it easy to deploy and manage in cloud-native environments. It leverages Kubernetes and Red Hat OpenShift for orchestration and scalability.

- ▶ Cloud and on-premises compatibility

Db2U is designed to work seamlessly in both public clouds (like IBM Cloud, AWS, Azure) and private data centers. This flexibility makes it suitable for hybrid cloud architectures.

- ▶ Ease of deployment

With Db2U, deploying a database is simplified. Users can quickly spin up Db2 instances using predefined Helm charts (the package manager for Kubernetes), which automate the setup process.

- ▶ Scalability

Db2U can scale out horizontally, allowing you to add more resources as needed without significant downtime. This is particularly useful for handling varying workloads.

- ▶ Automated Updates and Management

Db2U comes with automated patching, upgrades, and management capabilities, reducing the operational overhead of maintaining the database.

- ▶ Integrated with AI and Analytics

Db2U integrates well with AI and analytics tools, making it easier to perform complex data analysis and leverage machine learning models directly within the database environment.

- ▶ **High Availability and Disaster Recovery**

Db2U includes features for high availability (HA) and disaster recovery (DR), ensuring that your data is protected and that your database can continue to operate in the event of failures.

- ▶ **Support for Multiple Data Types**

Like the traditional Db2, Db2U supports a wide range of data types, including structured, semi-structured, and unstructured data, making it versatile for different kinds of applications.

- ▶ **Compliance and Security**

IBM Db2U includes robust security features and is compliant with various industry standards, which is critical for enterprises that handle sensitive data.

See [Modernizing Db2 Containerization Footprint with Db2U](#) for more information on the Db2U architecture.

3.2.6 Installing Db2 on Red Hat OpenShift on LinuxONE

In this section, we demonstrate how to install a Db2 database on Red Hat OpenShift (RHOCP) by first preparing our Red Hat OpenShift cluster and then installing and deploying the database.

As of Db2 11.5.5, Red Hat OpenShift has operator-enabled installations, allowing you more control over your deployment. You deploy Db2 to your Red Hat OpenShift cluster through a series of API calls to the Db2 Operator.

The Db2 Operator is acquired from either the IBM Operator Catalog or the Red Hat Marketplace. In our lab environment, we used the IBM Operator Catalog, which is accessible through the OpenShift user interface (UI) console, while the Red Hat Marketplace is a web site.

In our lab environment, we chose to install Db2 directly onto the Red Hat OpenShift Container Platform using the [IBM Cloud Pak® for Data](#) command line interface (cpd-cli). You do not need to install the complete IBM Cloud Pak for data, nor do you need a license for the cpd-cli.

Installing Db2U Operator on Red Hat OpenShift on LinuxONE

Before doing any actual installation work, verify the Red Hat OpenShift Container Platform (RHOCP) cluster nodes to ensure that their status is READY, as shown in Example 3-26. Run this command from the bastion node.

Example 3-26 Verify the RHOCP cluster nodes

```
[root@rdbko7b1 ibm]# oc get node
NAME                                     STATUS    ROLES
AGE   VERSION
rdbko7m1.ocp-dev.cpolab.ibm.com        Ready    control-plane,master
37h   v1.29.6+aba1e8d
rdbko7m2.ocp-dev.cpolab.ibm.com        Ready    control-plane,master
37h   v1.29.6+aba1e8d
rdbko7m3.ocp-dev.cpolab.ibm.com        Ready    control-plane,master
37h   v1.29.6+aba1e8d
```

```

rdbko7w1.ocp-dev.cp0lab.ibm.com Ready worker
37h v1.29.6+aba1e8d
rdbko7w2.ocp-dev.cp0lab.ibm.com Ready,
worker 37h v1.29.6+aba1e8d
rdbko7w3.ocp-dev.cp0lab.ibm.com Ready worker
36h v1.29.6+aba1e8d

```

Next, from the RHOCP command line, create a new project. A project is essentially the same as a namespace, but Red Hat OpenShift provides additional administrative controls for projects. In our lab environment, we used the following command:

```
oc new-project db2
```

When you create a new project by using the **oc new-project** command, the new project will automatically be set as the current project. To ensure your new project has been created, you can run the following command:

```
oc project db2
```

To view IBM offerings in the Red Hat OpenShift Operator catalog, the catalog index image needs to be enabled. To do this, we first created a Db2 catalog source YAML configuration file, as shown in Example 3-27.

Example 3-27 Db2 catalog source YAML file

```

vi db2u.yaml
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: ibm-operator-catalog
  namespace: openshift-marketplace
spec:
  displayName: "IBM Operator Catalog"
  publisher: IBM
  sourceType: grpc
  image: icr.io/cpopen/ibm-operator-catalog
  updateStrategy:
    registryPoll:
      interval: 45m

```

Next, we applied the YAML file to create the Db2 Operator catalog resource, as shown in Example 3-28.

Example 3-28 Apply the YAML file

```

[root@rdbko7b1 ibm]# oc apply -f db2u.yaml
catalogsource.operators.coreos.com/ibm-operator-catalog created

```

To verify the installation, use the command shown in Example 3-29 from the command line, where **-n openshift-marketplace** is the catalog source namespace, as highlighted in Example 3-27 in your YAML file.

Example 3-29 Verify the Db2 catalog resource

```

[root@rdbko7b1 ibm]# oc get CatalogSources ibm-operator-catalog -n
openshift-marketplace

```

```

NAME                                DISPLAY                                TYPE  PUBLISHER  AGE
ibm-operator-catalog               IBM Operator Catalog                 grpc  IBM        4m53s
[root@rdbko7b1 ibm]# oc get catalogsource,pods -n openshift-marketplace
NAME                                DISPLAY
TYPE  PUBLISHER  AGE
catalogsource.operators.coreos.com/certified-operators   Certified Operators
grpc  Red Hat    37h
catalogsource.operators.coreos.com/community-operators   Community Operators
grpc  Red Hat    37h
catalogsource.operators.coreos.com/ibm-operator-catalog  IBM Operator Catalog
grpc  IBM        5m3s
catalogsource.operators.coreos.com/redhat-marketplace     Red Hat Marketplace
grpc  Red Hat    37h
catalogsource.operators.coreos.com/redhat-operators       Red Hat Operators
grpc  Red Hat    37h

NAME                                READY  STATUS  RESTARTS  AGE
pod/certified-operators-4bvhs       1/1    Running  0          11h
pod/community-operators-pqv65       1/1    Running  0          33h
pod/ibm-operator-catalog-mcskv      1/1    Running  0          5m2s
pod/marketplace-operator-6d694c4984-6751d  1/1    Running  0          33h
pod/redhat-marketplace-5j6wc        1/1    Running  0          33h
pod/redhat-operators-pd22m          1/1    Running  0          33h

```

For workloads that rely heavily on multi-threading, such as databases, the default `pids-limit` of 1024 is insufficient. This setting governs not only the number of processes but also the number of threads, as a thread is essentially a process that shares memory. You will need to change the worker node process ID limit.

To change the worker node process ID limit, create a custom `ContainerRuntimeConfig` resource (in a YAML file) for configuring the CRI-O PID limit. CRI-O is an implementation of the Kubernetes CRI (Container Runtime Interface) to enable using OCI (Open Container Initiative) compatible runtimes. By using a `ContainerRuntimeConfig` custom resource (CR), you can set the configuration values and add a label to match the Machine Config Pool (MCP). The Machine Config Operator (MCO) then rebuilds the `crio.conf` and `storage.conf` configuration files on the associated nodes with the updated values. The MCO manages the operating system and keeps the cluster up to date and configured. See [Creating a ContainerRuntimeConfig CR to edit CRI-O parameters](#) for more details on this process.

As the MCO updates machines in each MCP, it reboots each node one by one.

Example 3-30 provides the command to create the file that will accomplish this task. We set our `pidsLimit` to **65536**.

Example 3-30 Change the worker node PID limit at the CRI-O level

```

[root@rdbko7b1 ibm]# vi pids-limit.yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
  name: crio-pids-limit
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ''

```

```
containerRuntimeConfig:
  pidLimit: 65536
```

Apply the changes by using the following command:

```
oc apply -f pids-limit.yaml
```

To verify the changes have been successful on the MCP, run the command shown in Example 3-31.

Example 3-31 Verify the CRI-O configuration changes were successful on the MCP

```
[root@rdbko7b1 ibm]# oc get mcp
```

NAME	CONFIG	UPDATED	UPDATING	DEGRADED
MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT	DEGRADEDMACHINECOUNT	AGE
master	rendered-master-26c1e0d1f03394f2e2e53287cefee8fb	True	False	False 3
3	3	0	37h	
worker	rendered-worker-c94d116d73511b4aea20b9ec0490001d	True	False	False 3
3	3	0	37h	

A successfully updated node has the following status: UPDATED=true, UPDATING=false, DEGRADED=false.

Once the worker nodes are rebooted, you can login and confirm the health of the nodes by running the command shown in Example 3-32. Healthy nodes show a status of Ready.

Example 3-32 Verify the CRI-O configuration changes were successful on the worker nodes

```
[root@rdbko7b1 ibm]# oc get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
rdbko7m1.ocp-dev.cpolar.ibm.com	Ready			control-plane,master 37h
v1.29.6+aba1e8d				
rdbko7m2.ocp-dev.cpolar.ibm.com	Ready			control-plane,master 37h
v1.29.6+aba1e8d				
rdbko7m3.ocp-dev.cpolar.ibm.com	Ready			control-plane,master 37h
v1.29.6+aba1e8d				
rdbko7w1.ocp-dev.cpolar.ibm.com	Ready			worker 37h
v1.29.6+aba1e8d				
rdbko7w2.ocp-dev.cpolar.ibm.com	Ready,SchedulingDisabled			worker 37h
v1.29.6+aba1e8d				
rdbko7w3.ocp-dev.cpolar.ibm.com	Ready			worker 36h
v1.29.6+aba1e8d				

See [About listing all the nodes in a cluster](#) for more details on nodes status.

Verify that the CRI-O PIDs limit has been applied, as shown in Example 3-33

Example 3-33 CRI-O PIDs limit verification

```
oc get containerruntimeconfigs crio-pids-limit -o yaml
apiVersion: machineconfiguration.openshift.io/v1
kind: ContainerRuntimeConfig
metadata:
  annotations:
    machineconfiguration.openshift.io/mc-name-suffix: ""
    creationTimestamp: "2024-08-18T05:26:14Z"
```

```

finalizers:
- 99-worker-generated-containerruntime
generation: 1
name: crio-pids-limit
resourceVersion: "688733"
uid: 3ae2cd55-b060-4b40-b734-011d19fba551
spec:
  containerRuntimeConfig:
    pidsLimit: 65536
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
status:
  conditions:
  - lastTransitionTime: "2024-08-18T05:26:15Z"
    message: Success
    status: "True"
    type: Success
  observedGeneration: 1

```

Perform the following tasks in order to create the Db2 Operator on the Red Hat OpenShift Container Platform cluster in the *db2* namespace:

1. Log into the Red Hat OpenShift Container Platform console as *kubeadmin*.
2. From the “hamburger” menu icon, select Operator Hub.
3. In the search bar, enter the keyword *db2*.
4. Select IBM Db2, provided by IBM to install the Operator.
5. Select *Install*.
6. Confirm you have the latest version - in our case, it was 110509.02.
7. Set the installation mode to a specific namespace on the cluster.
8. Select Install Operator in the *db2* namespace.
9. Select Update approval to Manual.
10. Select Install.
11. When prompted, select Approve.

Verify the Db2U pods in the *db2* namespace are running. Example 3-34 shows the command we used to verify, and its output.

Example 3-34 Verify the Db2U pods are running

```

[root@rdbko7b1 ibm]# oc get pod

```

NAME	READY	STATUS	RESTARTS	AGE
db2u-day2-ops-controller-manager-585bc88df8-cmwn6	1/1	Running	0	13m
db2u-operator-manager-5546bf8496-j62fr	1/1	Running	0	13m

We will use the NFS storage we set up in section 3.2.3, “Db2 DPF install on LinuxONE” on page 58 as our storage layer. We verify our pre-requisites before we install the Db2 instance.

Example 3-35 shows that all worker nodes are listed in our hosts file.

Example 3-35 Verify worker nodes in hosts file

```

[root@rdbko7b1 ibm]# cat /etc/hosts

```



```

127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6
9.76.61.174 rdbko7b1.ocp-dev.cpolab.ibm.com
9.76.61.178 rdbko7w1.ocp-dev.cpolab.ibm.com
9.76.61.179 rdbko7w2.ocp-dev.cpolab.ibm.com
9.76.61.180 rdbko7w3.ocp-dev.cpolab.ibm.com
9.76.61.181 rdbko7w4.ocp-dev.cpolab.ibm.com
9.76.61.175 rdbko7m1.ocp-dev.cpolab.ibm.com
9.76.61.176 rdbko7m2.ocp-dev.cpolab.ibm.com
9.76.61.177 rdbko7m3.ocp-dev.cpolab.ibm.com

```

Next we will verify that our hosts (listed in Example 3-35) have access to the NFS storage directory by examining the `/etc/exports` configuration file (Example 3-36). This configuration file controls which file systems are exported to remote hosts and specifies options. For more details on the options, see the following website:

https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/storage_administration_guide/nfs-serverconfig#nfs-serverconfig-exports

Example 3-36 Verify hosts are permitted to `/data/nfs_storage`

```

[root@rdbko7b1 ibm]# cat /etc/exports
/data/nfs_storage
9.76.61.174(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)
/data/nfs_storage
9.76.61.175(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)
/data/nfs_storage
9.76.61.176(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)
/data/nfs_storage
9.76.61.177(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)
/data/nfs_storage
9.76.61.178(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)
/data/nfs_storage
9.76.61.179(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)
/data/nfs_storage
9.76.61.180(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)
/data/nfs_storage
9.76.61.181(rw,no_root_squash,insecure,async,no_subtree_check,anonuid=5001,anongid=5001)

```

Finally, we use the command shown in Example 3-37 to query the mount daemon for information about the state of the NFS server on our machine. With no options, **showmount** lists the set of clients who are mounted from that host. In this example, we see the IP addresses of each of our hosts and can verify that they have the NFS storage file (`/data/nfs_storage`) mounted.

Example 3-37 Query the mount daemon for the mount list

```

[root@rdbko7b1 ibm]# showmount -e

```

```
Export list for rdbko7b1.ocp-dev.cpolab.ibm.com:
/data/nfs_storage
9.76.61.181,9.76.61.180,9.76.61.179,9.76.61.178,9.76.61.177,9.76.61.176,9.76.61.175,9.76.61.174
```

For the following activities, log into Red Hat OpenShift Container Platform. In order to install Db2 with a dependency on Db2U, you will need to create a *db2u-product-cm ConfigMap* to specify whether Db2U runs with limited privileges or elevated privileges. In our example, we chose to run Db2U with elevated privileges. We created the YAML file shown in Example 3-38.

Example 3-38 Create the configuration file

```
[root@rdbko7b1 ibm]# vi db2u_cm.yaml
apiVersion: v1
data:
  DB2U_RUN_WITH_LIMITED_PRIVS: "false"
kind: ConfigMap
metadata:
  name: db2u-product-cm
  namespace: db2

[root@rdbko7b1 ibm]# oc create -f db2u_cm.yaml
configmap/db2u-product-cm configured
```

Apply the resource by using the following command:
oc apply -f db2u_cm.yaml

From the bastion node (in our case, rdbko7b1 with an IP address of 9.76.61.174) on the Red Hat OpenShift Container Platform, use the commands shown in Example 3-39 to export the NFS storage.

Example 3-39 Export NFS storage

```
export NFS_SERVER_LOCATION=9.76.61.174
export NFS_PATH=/data/nfs_storage
export PROJECT_NFS_PROVISIONER=nfs-provisioner
export NFS_STORAGE_CLASS=managed-nfs-storage
export
NFS_IMAGE=registry.k8s.io/sig-storage/nfs-subdir-external-provisioner:v4.0.2
```

You can now log into the cluster on the Red Hat OpenShift Container Platform. Our command to login is shown in Example 3-40. The OpenShift Container Platform master includes a built-in OAuth server. Users obtain OAuth access tokens to authenticate themselves to the API. For more information on retrieving this token, see [Describe the details of a user-owned OAuth access token](#).

Example 3-40 Log into the RHOCPC cluster

```
[root@rdbko7b1 ibm]# cpd-cli manage login-to-ocp
--token=sha256~J1ovvH0bzpis-RWE5Bvka-6RiJrZkoJtd541wT0XrZg
--server=https://api.ocp-dev.cpolab.ibm.com:6443
[INFO] 2024-08-18T02:49:01.416704Z Checking architecture: s390x
[INFO] 2024-08-18T02:49:01.416740Z Checking podman or docker
[INFO] 2024-08-18T02:49:01.445593Z Dockerexe: podman
```

```
[INFO] 2024-08-18T02:49:01.478506Z Container olm-utils-play-v3 is running already.
Image: icr.io/cpopen/cpd/olm-utils-v3:latest.s390x
[INFO] 2024-08-18T02:49:01.510250Z Processing subcommand login-to-ocp
KUBECONFIG is /opt/ansible/.kubeconfig
WARNING: Using insecure TLS client config. Setting this option is not supported!
Logged into "https://api.ocp-dev.cpolab.ibm.com:6443" as "kube:admin" using the
token provided.
```

You have access to 70 projects, the list has been suppressed. You can list all projects with 'oc projects'

```
Using project "default".
Using project "default" on server "https://api.ocp-dev.cpolab.ibm.com:6443".
[SUCCESS] 2024-08-18T02:49:01.936354Z You may find output and logs in the
/root/ibm/cpd-cli-workspace/olm-utils-workspace/work directory.
[SUCCESS] 2024-08-18T02:49:01.936392Z The login-to-ocp command ran successfully.
```

Our next objective will be to create the NFS storage class for the Db2 database on RHOC. Since Db2 supports only NFS version 3, we must first change the NFS protocol to NFS3. To do this, you will edit the NFS mount configuration file using the following command:

```
vi /etc/nfsmount.conf
```

In that file, change Defalutvers=4 to Defaultvers=3 and uncomment vers3=y.

Restart the server with the command found in Example 3-41.

Example 3-41 Restart the NFS server

```
[root@rdbko7b1 ibm]# systemctl restart nfs-server rpc-statd.service
rpcbind.service rpcbind.socket
[SUCCESS] 2024-08-18T02:49:01.936392Z The login-to-ocp command ran successfully.
```

Next we will create the storage class by using the command found in Example 3-42.

Example 3-42 Create the storage class for NFS

```
[root@rdbko7b1 ibm]# cpd-cli manage setup-nfs-provisioner
--nfs_server=${NFS_SERVER_LOCATION} --nfs_path=${NFS_PATH}
--nfs_provisioner_ns=${PROJECT_NFS_PROVISIONER}
--nfs_storageclass_name=${NFS_STORAGE_CLASS} --nfs_provisioner_image=${NFS_IMAGE}
[INFO] 2024-08-18T03:23:50.108073Z Checking architecture: s390x
[INFO] 2024-08-18T03:23:50.108110Z Checking podman or docker
[INFO] 2024-08-18T03:23:50.135679Z Dockerexe: podman
[INFO] 2024-08-18T03:23:50.170786Z Container olm-utils-play-v3 is running already.
Image: icr.io/cpopen/cpd/olm-utils-v3:latest.s390x
[INFO] 2024-08-18T03:23:50.202662Z Processing subcommand setup-nfs-provisioner
[INFO] 2024-08-18T03:23:50.202708Z Run command: podman exec -it olm-utils-play-v3
setup-nfs-provisioner --nfs_server=9.76.61.174 --nfs_path=/data/nfs_storage
--nfs_provisioner_ns=nfs-provisioner --nfs_storageclass_name=managed-nfs-storage
--nfs_provisioner_image=registry.k8s.io/sig-storage/nfs-subdir-external-provisioner:v4.0.2
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that
the implicit localhost does not match 'all'
```

```
PLAY [localhost]
*****
TASK [include_role : utils]
*****

TASK [utils : apply scc to nfs provisioner service account]
*****
changed: [localhost]

TASK [utils : create nfs provisioner namespace]
*****
changed: [localhost]

TASK [utils : apply nfs-provisioner deployment]
*****
changed: [localhost]

TASK [utils : wait until nfs provisioner is up and running]
*****
ok: [localhost]

TASK [utils : create nfs-provisioner storage class "managed-nfs-storage"]
*****
changed: [localhost]

TASK [utils : creating test pvc using managed-nfs-storage sc]
*****
changed: [localhost]

TASK [utils : wait until nfs test pvc is bound]
*****
ok: [localhost] => (item=sc-test-pvc)

TASK [utils : remove scc from nfs provisioner service account]
*****
skipping: [localhost]

TASK [utils : remove nfs-provisioner deployment]
*****
skipping: [localhost]

TASK [utils : remove nfs provisioner namespace]
*****
skipping: [localhost]

TASK [utils : remove nfs-provisioner storage class "managed-nfs-storage"]
*****
skipping: [localhost]

TASK [include_role : utils]
*****
skipping: [localhost]

PLAY RECAP
*****
```

```
localhost : ok=7 changed=5 unreachable=0 failed=0
skipped=5 rescued=0 ignored=0
```

[SUCCESS] 2024-08-18T03:24:03.567674Z You may find output and logs in the /root/ibm/cpd-cli-workspace/olm-utils-workspace/work directory.

[SUCCESS] 2024-08-18T03:24:03.567871Z The setup-nfs-provisioner command ran successfully.

To check the NFS storage class, you will run the command shown in Example 3-43.

Example 3-43 Verify the NFS storage class

```
[root@rdbko7b1 ibm]# oc get sc
NAME                                PROVISIONER                                RECLAIMPOLICY
VOLUMEBINDINGMODE ALLOWVOLUMEEXPANSION AGE
managed-nfs-storage k8s-sigs.io/nfs-subdir-external-provisioner Delete
Immediate          false                                38s
```

Note that in Example 3-42, the `nfs_storageclass_name=managed-nfs-storage` and the name shown in Example 3-43 is the same.

You can now proceed to create the Db2 instance and database. To do this, you will create a YAML file, as shown in Example 3-44. This example reflects the configuration parameters for our lab environment. You can change the configuration parameters to suit your own environment.

Example 3-44 YAML file to create Db2 instance and database

```
[root@rdbko7b1 ibm]# vi db2_inst.yam
apiVersion: db2u.databases.ibm.com/v1
kind: Db2uInstance
metadata:
  name: db2u-cr
  namespace: db2
spec:
  account:
    securityConfig:
      privilegedSysctlInit: true
  environment:
    authentication:
      ldap:
        enabled: true
  databases:
  - dbConfig:
    LOGPRIMARY: "20"
    LOGSECOND: "10"
    LOGFILSIZ: "8000"
    NUM_DB_BACKUPS: "2"
    TRACKMOD: "YES"
    name: BLUDB
    dbType: db2wh
    instance:
      registry:
        DB2_4K_DEVICE_SUPPORT: "ON"
        DB2COMPOPT: "LOCKAVOID_EXT_CATSCANS"
    partitionConfig:
```

```
    dataOnMln0: true
    total: 12
    volumePerPartition: true
license:
  accept: true
nodes: 2
podTemplate:
  db2u:
    resource:
      db2u:
        limits:
          cpu: 8
          memory: 60Gi
advOpts:
  memoryPercent: 99
storage:
- name: meta
  spec:
    accessModes:
    - ReadWriteMany
    resources:
      requests:
        storage: 40Gi
    storageClassName: managed-nfs-storage
    type: create
- name: data
  spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 40Gi
    storageClassName: managed-nfs-storage
    type: template
- name: archiveLogs
  spec:
    accessModes:
    - ReadWriteMany
    resources:
      requests:
        storage: 40Gi
    storageClassName: managed-nfs-storage
    type: create
- name: backup
  spec:
    accessModes:
    - ReadWriteMany
    resources:
      requests:
        storage: 80Gi
    storageClassName: managed-nfs-storage
    type: create
- name: tempts
  spec:
    accessModes:
```

```

- ReadWriteOnce
resources:
  requests:
    storage: 40Gi
  storageClassName: managed-nfs-storage
type: template
version: s11.5.9.0-cn2

```

Ensure you are in the correct RHOCP project (namespace) by running the command shown in Example 3-45. Our project is “db2”.

Example 3-45 Verify the project

```

[root@rdbko7b1 ibm]# oc project db2
Now using project "db2" on server "https://api.ocp-dev.cpolab.ibm.com:6443".

```

While the command shown in Example 3-45 only shows a subset of resources, it is sufficient for our purposes. Basically, we are just reviewing the status of our pods.

Example 3-46 Check pods

```

[root@rdbko7b1 ibm]# oc get all
Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+, unavailable in v4.10000+
NAME                                READY   STATUS    RESTARTS   AGE
pod/db2u-day2-ops-controller-manager-585bc88df8-cmwn6  1/1     Running   0           118m
pod/db2u-operator-manager-5546bf8496-j62fr             1/1     Running   0           118m

NAME                                TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/db2u-day2-ops-controller-manager-metrics-service  ClusterIP    172.30.82.205 <none>        8443/TCP    127m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/db2u-day2-ops-controller-manager           1/1     1             1           127m
deployment.apps/db2u-operator-manager                     1/1     1             1           127m

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/db2u-day2-ops-controller-manager-585bc88df8  1         1         1       127m
replicaset.apps/db2u-operator-manager-5546bf8496             1         1         1       127m

```

At this point of the process, now that you have ensured everything is in a good state, you can either apply these configuration details by using the following command:

```
oc apply -f db2_inst.yaml
```

Or, you can run the command shown in Example 3-47 to create the instance using the YAML file you created in Example 3-44.

Example 3-47 Create the instance

```

[root@rdbko7b1 ibm]# oc create -f db2_inst.yaml
db2uinstance.db2u.databases.ibm.com/db2u-cr created

```

You will now verify the Db2 persistent volume claims (PVC) are bound, as shown in Example 3-48. Note that the STORAGECLASS listed in this example is the storage class you exported in Example 3-39, managed-nfs-storage.

Example 3-48 Verify the Db2 PVC

```
[root@rdbko7b1 ibm]# oc get pvc
NAME          STATUS    VOLUME          CAPACITY   ACCESS MODES   STORAGECLASS          VOLUMEATTRIBUTESCLASS
AGE
c-db2u-cr-archivelogs      Bound    pvc-1de77549-eca6-4742-b6e3-a7e17b1e200e  40Gi    RWX
managed-nfs-storage        <unset>                2m37s
c-db2u-cr-backup           Bound    pvc-f468a2d2-afed-4698-9506-1af357e3cef3  80Gi    RWX
managed-nfs-storage        <unset>                2m37s
c-db2u-cr-meta             Bound    pvc-6ea4aec9-633b-41d0-b2f3-e8f0f1775e8a  40Gi    RWX
managed-nfs-storage        <unset>                2m37s
data-c-db2u-cr-db2u-mln-0  Bound    pvc-efc819c1-76dc-46e2-a72a-dc54f1e07f15  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-1  Bound    pvc-f198f2ac-0bc0-4b2e-98fe-2aaf4515c61c  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-10 Bound    pvc-f94199ce-ca71-4368-af96-586886003772  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-11 Bound    pvc-b1fc0ef4-0bf5-4fb9-ab9f-36751c1a6589  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-12 Bound    pvc-8dfbe92b-14dd-40d7-b360-f305948cc553  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-13 Bound    pvc-8171bd92-4513-405f-aabe-3f0e9a8cd8c0  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-14 Bound    pvc-4e401c18-d0c2-4cef-87ae-ca8fb3987cd8  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-15 Bound    pvc-dfdbcf3f-3687-49cb-9167-0bad9386a804  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-2  Bound    pvc-6232ebdf-4e13-4000-bef0-553e3b17b6c7  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-3  Bound    pvc-b9e6c580-eb31-4129-8cbb-328f8f88b5a7  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-4  Bound    pvc-1f7fc8aa-53b9-4b06-8a40-ad7df0556af1  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-5  Bound    pvc-61fc1d33-9909-4966-b3e9-d8f7298ac4e7  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-6  Bound    pvc-00d5fd80-9da4-443f-9026-370ff4420881  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-7  Bound    pvc-9df18faa-22df-4c39-aca8-500713ff76dc  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-8  Bound    pvc-8fc60af2-9155-439f-be8d-63aa0c315397  40Gi    RWO
managed-nfs-storage        <unset>                31s
data-c-db2u-cr-db2u-mln-9  Bound    pvc-1c279dea-f4ee-4086-a63b-edc49602db99  40Gi    RWO
managed-nfs-storage        <unset>                31s
tempt-c-db2u-cr-db2u-0     Bound    pvc-3a180f71-13c8-441a-9db2-aeb99870fddd  40Gi    RWO
managed-nfs-storage        <unset>                31s
tempt-c-db2u-cr-db2u-1     Bound    pvc-40175107-8a66-4ad9-a252-10a0d37597d6  40Gi    RWO
managed-nfs-storage        <unset>                38s
```

Check the pods once again to ensure they are running, as shown in Example 3-49.

Example 3-49 Verify the Db2 pods are running

```
[root@rdbko7b1 ~]# oc get pod
NAME                READY   STATUS    RESTARTS   AGE
c-db2u-cr-db2u-0    1/1     Running   0           22m
c-db2u-cr-db2u-1    1/1     Running   0           22m
```


c-db2u-cr-etcd-0	1/1	Running	0	23m
c-db2u-cr-ldap-756f677897-m7f5r	1/1	Running	0	23m
c-db2u-cr-tools-94d49565b-bmdvw	1/1	Running	0	23m
db2u-day2-ops-controller-manager-585bc88df8-m7j5j	0/1	Running	0	30s
db2u-operator-manager-5546bf8496-jc8bw	1/1	Running	0	30s

Verify the newly created Db2U instance is in a ready state, as shown in Example 3-50.

Example 3-50 Check the Db2U instance on RHOCP

```
[root@rdbko7b1 ibm]# oc get Db2uInstance
NAME      STATE      MAINTENANCESTATE  AGE
db2u-cr   Ready      None                6m9s
```

Run the command shown in Example 3-51 to verify the Db2 database service port.

Example 3-51 Check the Db2 database service port on RHOCP

```
[root@rdbko7b1 ibm]# oc get svc c-db2u-cr-db2u-engn-svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
c-db2u-cr-db2u-engn-svc  NodePort  172.30.203.9    <none>
50001:32051/TCP,50000:32217/TCP  4m3s
```

We next want to set up basic load balancing by using the HAProxy configuration file. First, we edit the `/etc/haproxy/haproxy.cfg` configuration file, as shown in Example 3-52.

Example 3-52 Edit the HAProxy configuration file

```
[root@rdbko7b1 ibm]# vi /etc/haproxy/haproxy.cfg
frontend db2
    bind *:32217
    default_backend db2u
    mode tcp
    option tcplog
backend db2u
    balance source
    mode tcp
    server rdbko7m1.ocp-dev.cpolab.ibm.com 9.76.61.175:32217 check
    server rdbko7m2.ocp-dev.cpolab.ibm.com 9.76.61.176:32217 check
    server rdbko7m3.ocp-dev.cpolab.ibm.com 9.76.61.177:32217 check
frontend db2ssl
    bind *:32051
    default_backend db2u
    mode tcp
    option tcplog
backend db2ussl
    balance source
    mode tcp
    server rdbko7m1.ocp-dev.cpolab.ibm.com 9.76.61.175:32051 check
    server rdbko7m2.ocp-dev.cpolab.ibm.com 9.76.61.176:32051 check
    server rdbko7m3.ocp-dev.cpolab.ibm.com 9.76.61.177:32051 check
```

Restart HAProxy by using the following command:

```
[root@rdbko7b1 ibm]#systemctl restart haproxy
```

Validate the Db2 instance and database on the Red Hat OpenShift Container Platform, switch to the Db2 instance userID and connect to the database, by using the commands shown in Example 3-53. The `oc rsh` command allows you to locally access and manage tools that are on the system. In our example, we are opening a remote shell session to our container (pod) before we try to connect to our database.

Example 3-53 Validate the Db2 instance and database

```
[root@rdbko7b1 ibm]# oc rsh c-db2u-cr-db2u-0 bash -l

su - db2inst1
[db2inst1@c-db2u-cr-db2u-0 - Db2U ~]$ db2 connect to bludb
```

3.3 Db2 migration to LinuxONE

In this section, we provide an overview of the migration steps that are needed to migrate from a Db2 system on one architecture to a Db2 system on LinuxONE.

You also can review the migration steps in section 6.2.2 of the IBM Redbooks publication, [Practical Migration from x86 to LinuxONE](#), SG24-8377.

3.3.1 Database migration planning

The migration process requires several steps, and anticipating how much time is needed for each step is crucial. In general, plan for the proof of concept, associated testing, and final implementation to take the most time for a migration.

The following are the phases of migration at the highest level:

- ▶ **Planning:** After you have decided what will be migrated and how, the plan must specify the time, risks, and owner for each migration task.
- ▶ **PoC and Test:** Proof of concept to check the compatibilities between the x86 and LinuxONE environment and give special focus to performance.
- ▶ **Education:** The technical staff needs the correct skills to work on a LinuxONE migration and maintain the new environment.
- ▶ **Build Environment:** In this phase, the new infrastructure is readied for migration.
- ▶ **Implementation:** The actual migration. Communication between stakeholders is important during this process. All involved people must know and approve the migration and receive follow up reporting on the progress.
- ▶ **Post Migration:** After implementation, documentation must be created that further references the project and documents all necessary maintenance and care procedures. Additionally, the project manager must have a signed acceptance agreement.

3.3.2 Database workloads considerations before you migrate

When you make the decision to migrate and consolidate, the next step is to examine which workloads would be good candidates to be migrated. Start with a fairly simple application that has a low service level agreement (SLA) and a staff that has the associated skills. Migrate a single instance first before attempt to migrate database with a lot of instances.

For more details on application analysis, see section 5.3 of the IBM Redbooks publication, [Practical Migration from x86 to LinuxONE](#), SG24-8377.

For applications developed within the company, ensure that you have the source code available. Regarding the operating system platform, even a workload from a different platform can be migrated but start with servers running Linux. This process will substantially increase the success criteria of the migration effort. Applications that require proximity to corporate data stored on IBM LinuxONE are also ideal candidates, as are applications that have high I/O rates because I/O workloads are off-loaded from general-purpose processors onto specialized I/O processors.

IBM LinuxONE III has a powerful processor with a clock speed of 5.2 GHz. Because IBM LinuxONE is designed to concurrently run disparate workloads, remember that some workloads that required dedicated physical processors designed to run at high sustained CPU utilization rates might not be optimal candidates for migration to a virtualized Linux environment. This is because workloads that require dedicated processors do not take advantage of the virtualization and hardware sharing capabilities. An example of such an application might include video rendering, which requires specialized video hardware.

Table 3-1 Summary of Migration considerations

Variables to be considered	General Migration Strategy requirements
Associated costs	Proper size the workload
Application complexity	Start with a fairly simple application
Service level agreements	Start with a application with non-critical SLAs
Skills and abilities of your support staff	Migrate a single instance

3.3.3 Database migration from Db2 on x86 to Db2 on LinuxONE

There are many valid reasons for considering a migration to IBM LinuxONE.

- ▶ IBM LinuxONE is designed to support scalability and include high availability, high I/O bandwidth capabilities, the flexibility to run disparate workloads concurrently, and excellent disaster recovery capabilities.
- ▶ LinuxONE offers a sustainable and cyber-resilient platform for hybrid cloud and AI applications, which can also help reduce total cost of ownership through workload consolidation.
- ▶ LinuxONE supports built-in data encryption capabilities using Central Processor Assist for Cryptographic Function (CPACF)
- ▶ Another key element in choosing the appropriate applications for migration is whether they are supported on LinuxONE
- ▶ In most cases, a migration to LinuxONE will help an organization realize significant cost savings over three to five years.
- ▶ You can boost the performance and speed of your Linux applications by putting them on the same physical server as their data source.
- ▶ Applications with high I/O or transactional I/O. Because of its design, LinuxONE excels at handling sustained high I/O rates.
- ▶ Applications with lower sustained CPU peaks and average memory needs. These are ideal workloads for LinuxONE. The platform has been designed to run multiple workloads at a consistently high CPU and memory utilization.

When choosing an application for a proof-of-concept (POC), keep it as simple as possible. For more detailed information, see the IBM Redbooks publication, [Practical Migration from x86 to LinuxONE](#), SG24-8377.

A brief summary of different migration strategies along with their pros and cons are summarized in Table 3-2.

Table 3-2 Summary of migration strategies with pros and cons

Migration Strategy	Pros	Cons
Linux backup and restore. Migrate backed-up data from an operating environment to IBM LinuxONE.	<ul style="list-style-type: none"> ▶ Relatively Simple ▶ Low amount of downtime 	<ul style="list-style-type: none"> ▶ This will work only if the old and new servers have same endian
Db2 backup and restore.	<ul style="list-style-type: none"> ▶ Relatively Simple ▶ Low amount of downtime 	<ul style="list-style-type: none"> ▶ This will work only if the old and new servers have same endian
Tableby Tablemigration using LOAD FROM CURSOR^a .	<ul style="list-style-type: none"> ▶ Does not require a lot of space to keep the unload files 	<ul style="list-style-type: none"> ▶ Takes time to migrate the data ▶ Requires direct connection on a good network bandwidth between the old and the new Db2 databases
Tableby Tablemigration using db2look and db2move utilities ^b .	<ul style="list-style-type: none"> ▶ Automated in a way that it is single command to dump the data from a database with large numbers of tables 	<ul style="list-style-type: none"> ▶ Takes time to migrate the data ▶ Require more temporary storage space to hold the unloaded data
Tableby Tablemigration using Db2 HPU utility. IBM Db2 High Performance Unload (Db2 HPU) is a high-speed Db2 utility for unloading Db2 tables from either a table space or from an image copy. Tables are unloaded to one or more files based on a format that you specify ^c .	<ul style="list-style-type: none"> ▶ Much faster for large sized tables compare to the above 2 techniques ▶ Restore is from a backup image and hence much faster and easier to schedule 	<ul style="list-style-type: none"> ▶ Takes time to migrate the data

a. For more information on this command, see:

<https://www.ibm.com/docs/en/db2/11.5?topic=data-moving-using-cursor-file-type>

b. See section 6.2.2 of the IBM Redbooks publication, Practical Migration from x86 to LinuxONE, SG24-8377.

c. For more information on this utility, see:

<https://www.ibm.com/docs/en/dhpufz/5.2.0?topic=documentation-db2-high-performance-unload-overview>

Tips for a successful migration

Almost all databases use buffer pools in the shared memory area to manage the database memory context. Avoid the use of any automatic memory management systems to allocate shared memory. For example, if 6 GB of shared memory must be allocated to the database, force the database to allocate all memory at the system start.

If the database is not using all of the available memory, reduce the server memory until it starts paging. A system that is constantly swapping is the first indication of insufficient memory.

High load averages are not always an indication of CPU bottlenecks. Monitor the LPAR performance and determine if any other process or server that is running in the same LPAR is competing for CPU time when the problem occurs.

Database data files and log files must be in different file systems and should be striped across the storage hardware. Have multiple paths to the data to ensure availability.

The systems administrator and the database administrator must work together during the sizing process. Database servers typically require adjustments at the Linux and database levels



Postgres and LinuxONE

PostgreSQL (often referred to as Postgres) is an advanced open source object-relational database system that uses and extends the SQL language as well as JSON (non-relational) querying. Originally started in 1981 as the Ingres project at the University of California at Berkeley and was later named Postgres as it became "post Ingres" and the project was renamed as PostgreSQL to reflect its support for SQL in 1996. As of August 2024, PostgreSQL ranked #4 in popularity according to the [DB-Engines Ranking](#) website.

The first PostgreSQL release known as version 6.0 was released on January 29, 1997 and since then PostgreSQL has continued to be developed by the PostgreSQL Global Development Group, a diverse group of companies and many thousands of individual contributors. PostgreSQL version 16 released on September 14, 2023, is the 33rd major release in over 37 years of development. PostgreSQL is available from Linux distributions like Red Hat, SLES and Ubuntu and as a Docker image.

The PostgreSQL open source-based database has gained significant adoption in the last few years. With a vibrant and engaged community, PostgreSQL has become a viable alternative for enterprises trying to replace and modernize databases that support their legacy systems of record. There are vendors like Fujitsu and EnterpriseDB (EDB) that enhance the capabilities of open-source PostgreSQL and provide value added services.

In this chapter we will discuss a specific distribution of PostgreSQL provided by Fujitsu that augments the capabilities of the open source edition to provide an enterprise-grade experience and support. This distribution, which is combined with the IBM LinuxONE server, delivers a robust solution.

Additionally, we discuss EDB Postgres Advanced Server, an enterprise-ready database purpose-built that leverages innovations of the open-source Postgres community, bundled with advanced tools for administration, legacy database migration, storage optimization and monitoring, and backed by the support in the industry.

In this chapter, we discuss the following topics:

- ▶ "Fujitsu Enterprise Postgres on IBM LinuxONE" on page 88
- ▶ "EnterpriseDB Postgres Advanced Server on IBM LinuxONE" on page 109

4.1 Fujitsu Enterprise Postgres on IBM LinuxONE

Fujitsu Enterprise Postgres is a feature-rich, enterprise-grade relational database management system (RDBMS) and is fully compatible with PostgreSQL. As an enterprise-grade RDBMS, Fujitsu Enterprise Postgres offers open-source value and enterprise quality for mission-critical systems. It is ANSI SQL compliant and shares open-source software (OSS) PostgreSQL features, including ACID compliance, extensibility, inheritance, clustering, Unicode, and MultiVersion Concurrency Control (MVCC). Fujitsu extends these features with database multiplexing, high availability (HA) clustering, in-memory columnar store (Vertical Clustered Index (VCI)), Transparent Data Encryption (TDE) integration with IBM LinuxONE CryptoCard (provides encryption with FIPS 140-2 Level 4 certification), and PCI-DSS-compliant audit log and policy-based data masking. Fujitsu Enterprise Postgres adds enterprise-grade features for security, HTAP workloads, high availability, and mission-critical systems.

Fujitsu Enterprise Postgres version 17 fully supports IBM LinuxONE and leverages the IBM LinuxONE virtualization capabilities to provide a highly flexible, scalable, and resilient data serving platform that enables organizations to adopt various data service architectures to meet the needs of any business transformation initiative. Fujitsu Enterprise Postgres on IBM LinuxONE embraces open-source innovation with the improved sustainability, performance, scalability, and resiliency that IBM LinuxONE delivers.

Figure 4-1 provides a view of all the enterprise features covering development, operations, and management of Fujitsu Enterprise Postgres.

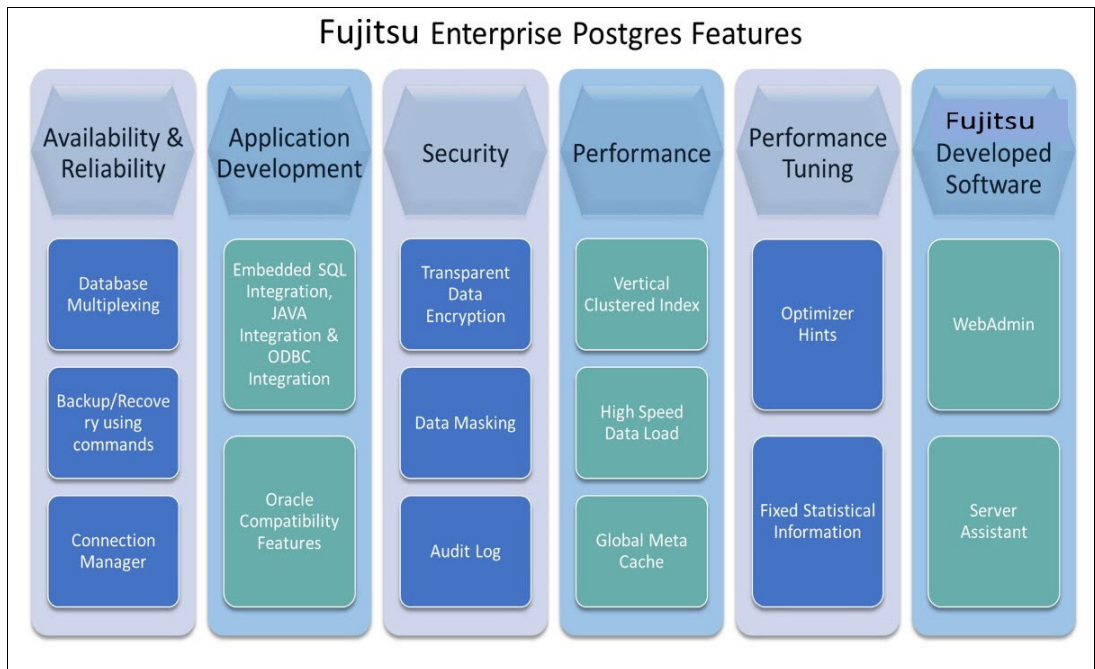


Figure 4-1 Fujitsu Enterprise Postgres features

4.1.1 Availability and reliability features

There are several types of HA modes that can be achieved with Fujitsu Enterprise Postgres Advanced Edition. In general, a complete HA solution for databases should be highly resilient and have instant availability through automatic failover for business continuity. A robust and fault-tolerant HA implementation should meet the following four major requirements:

- ▶ Load sharing and load balancing
- ▶ Instant failover
- ▶ Controlled switchover
- ▶ Disaster recovery (DR)

All the above HA requirements are possible through various data replication modes that are available with Fujitsu Enterprise Postgres. In this section, we describe how Fujitsu Enterprise Postgres provides multiple ways to reliably back up and recover operational and business-critical data in multiple scenarios.

Database multiplexing

Database multiplexing (also known as database mirroring) is a mode of operation that enables HA by creating copies of the database running on other similar systems by using the Postgres streaming replication technology. With the Fujitsu database multiplexing feature, the database is mirrored and the capability of the databases to switch over from the primary database to the standby database automatically is enhanced. The database multiplexing mode of operation is managed by the software component that is called the Mirroring Controller (MC)

Figure 4-2 shows a high-level architecture with switchover and failover capabilities that uses a Mirroring Controller (MC).

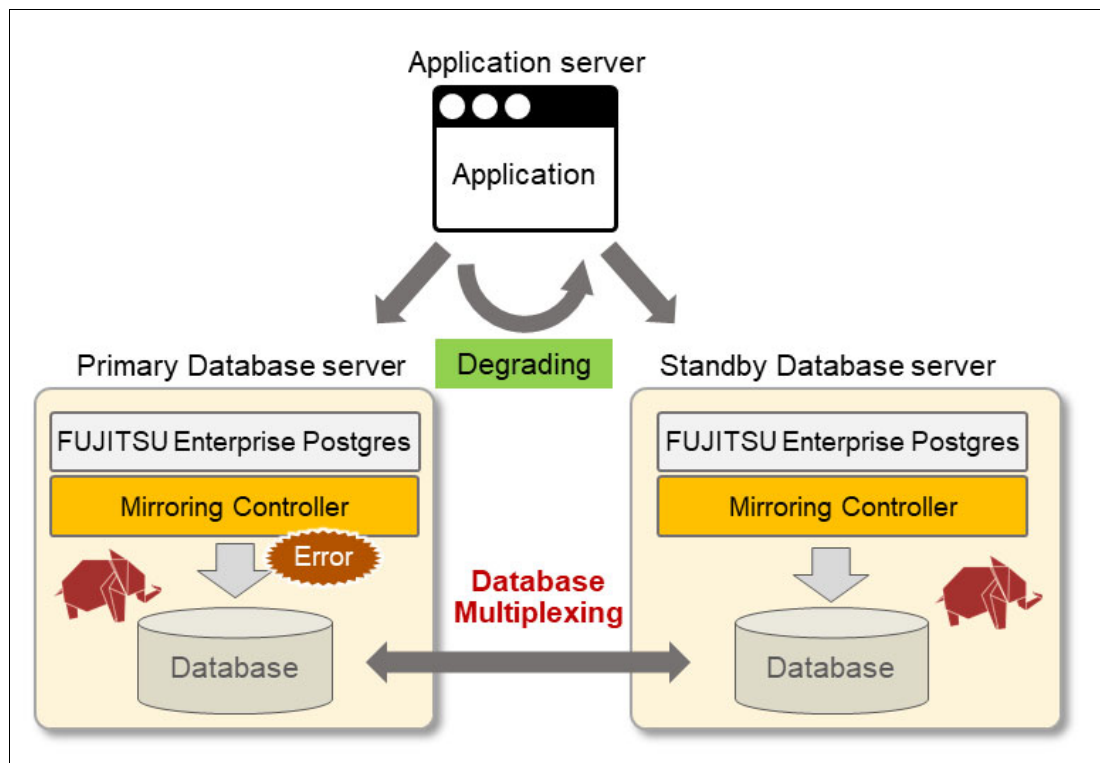


Figure 4-2 Fujitsu Enterprise Postgres Database Multiplexing

The MC feature enables the primary server (the database server that is used for the main jobs) to be switched automatically to the standby server if an error occurs in the former. In addition, data on the standby server can be made available for read access, allowing the standby server to be used for tasks such as data analysis and reporting in parallel to the primary server workload.

The MC feature provides the following benefits:

- ▶ Constantly monitor the operating system (OS), server, disk, network, and database, and notify you if something is amiss.
- ▶ If the primary server fails, MC performs an automatic switch/disconnect to the standby server to ensure operational continuity, and the primary server is disconnected.
- ▶ Detects streaming replication issues (log transfer network and Write-Ahead-Log (WAL) send/receive processes) by periodically accessing the PostgreSQL system views.
- ▶ Provides HA with minimal disruption (in seconds).
- ▶ Packaged as part of Fujitsu Enterprise Postgres Advanced Edition and comes with no additional cost over the annual subscription license.

Note: For more information, see the "Database Multiplexing Mode" section in the [Fujitsu Enterprise Postgres Cluster Operation Guide](#).

The MC is effective in performing failover and controlled switchover when any issues occur in the primary database server. To resolve a split-brain scenario in data multiplexing mode, Fujitsu Enterprise Postgres uses the Server Assistant program, which is described in the next section, "Server Assistant: Avoiding split-brain scenarios" on page 90.

Server Assistant: Avoiding split-brain scenarios

Server Assistant is a feature that objectively checks the status of database servers as a third party and isolates (fences) unstable servers in scenarios like network issues between the database servers. In such cases, when the communication link between the database servers is broken, MC might not be able to correctly determine the status of the other server, which might result in a situation where both the primary and standby servers might temporarily operate as primary servers, so it might be possible to perform updates on either servers that might lead to data corruption issues. This situation is known as split-brain scenario, which Server Assistant prevents from happening.

Figure 4-3 shows a high-level architecture with Server Assistant providing the quorum service (also known as arbitration service) between the primary and standby server and fencing the primary database server when the heartbeat fails between the primary and the standby database servers.

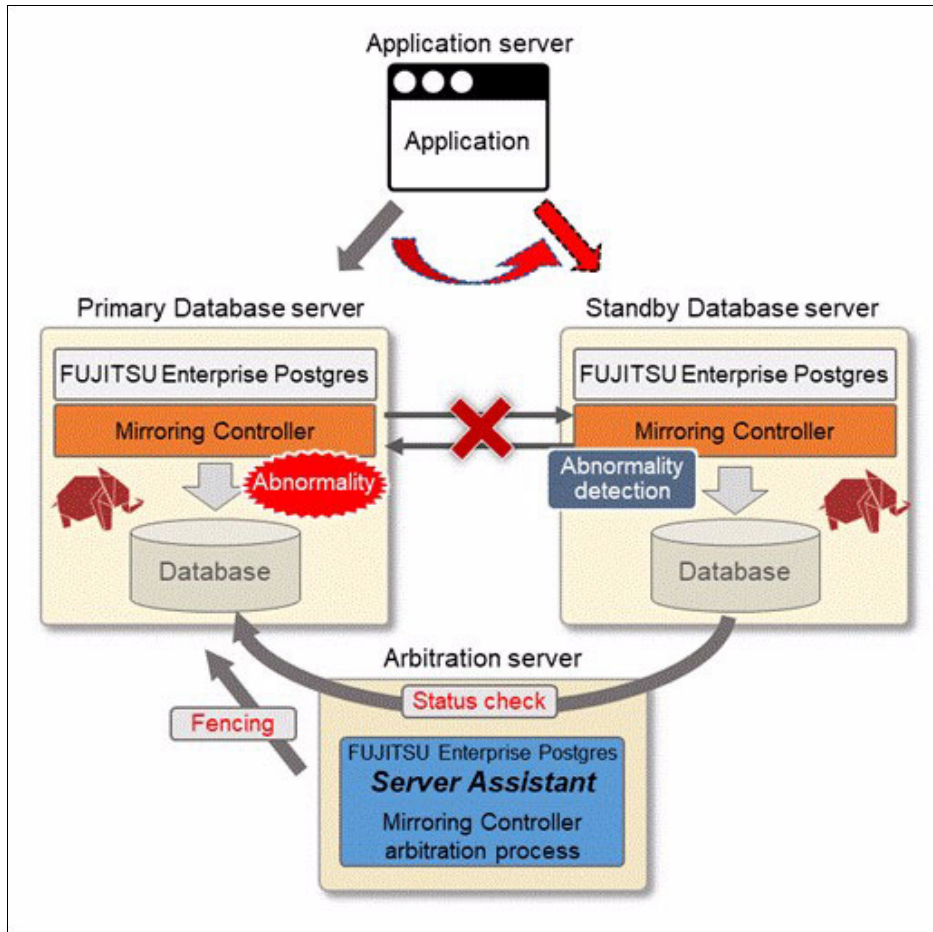


Figure 4-3 Fujitsu Enterprise Postgres Server Assistant arbitration process

Server Assistant is packaged with Fujitsu Enterprise Postgres Advanced Edition and provided at no additional cost. Server Assistant can be installed on a different server that is also known as an arbitration server.

Backup and recovery

Many factors, including hardware failure, cyberattack, natural disasters, and human error can put data at risk, and that may result in an impact to both business opportunities and the reputation of an organization. Implementing a robust backup/recovery plan to ensure that your business is back up and running with minimal revenue and data loss should be the top priority.

Fujitsu Enterprise Postgres supports many backup and recovery methods that are available through the GUI and command-line interface (CLI). Although the Fujitsu WebAdmin GUI tool provides a one-click backup and recovery feature, the CLI utilities `pgx_dmpa11` and `pgx_rcva11` provide a more granular and automated backup and recovery capability.

Backup customization flexibility

Fujitsu Enterprise Postgres extends the PostgreSQL backup/recovery utilities by allowing automation of the full and incremental backup as well as physical and logical backups, tasks through scripts, where you can use the copy technology of your choice.

Encryption

Files that are backed up by Fujitsu Enterprise Postgres are encrypted and automatically protected even if the backup medium is lost or somehow accessed by unauthorized users. In contrast, files that are generated by the equivalent PostgreSQL commands `pg_dump` and `pg_dumpall` are not encrypted and must be encrypted by using OpenSSL commands or other means before they are saved.

Types of backup methods that are supported

Figure 4-4 shows the many utilities that Fujitsu Enterprise Postgres supports for backup.

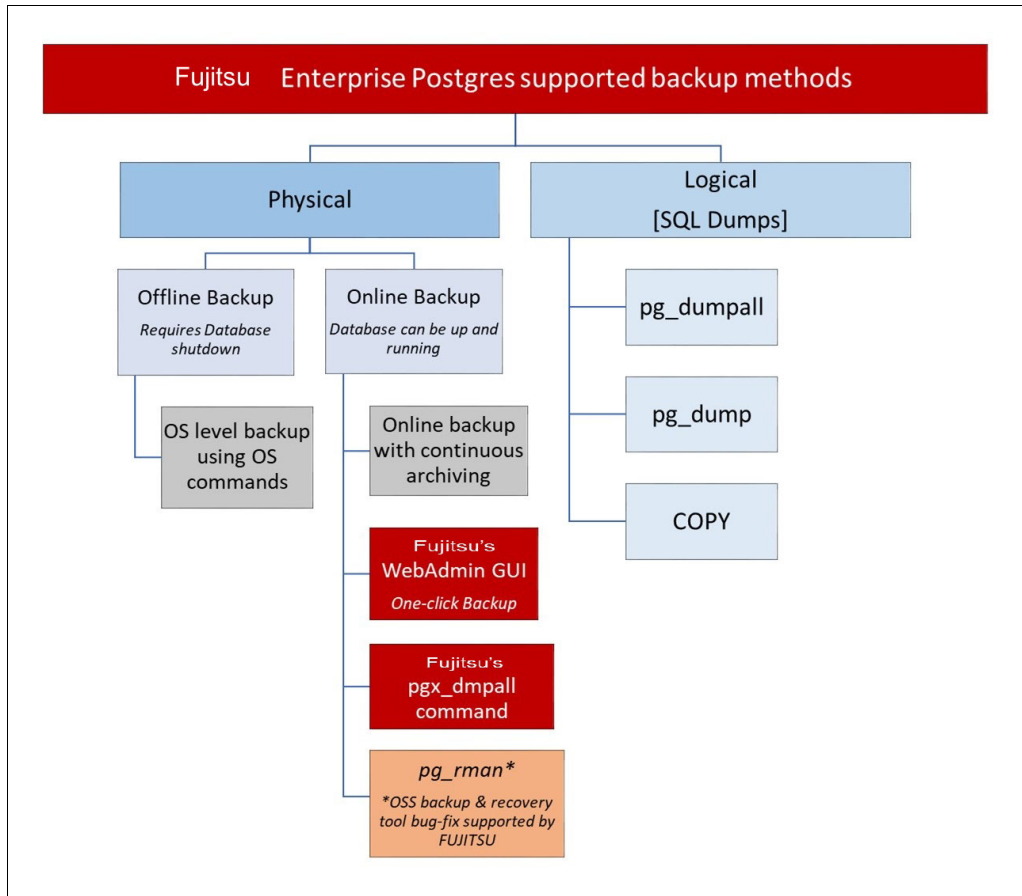


Figure 4-4 Fujitsu Enterprise Postgres supported backup methods

Connection Manager

Fujitsu Enterprise Postgres Connection Manager is a HA feature that provides transparent connectivity to the Fujitsu Enterprise Postgres HA database cluster. Unlike MC and Server Assistant, which are part of the database layer, Connection Manager is part of the application layer. Connection Manager is configured on the application and database servers.

Figure 4-5 shows the operation of Connection Manager. Connection Manager monitors the application server and Fujitsu Enterprise Postgres Database instances running on the database servers, which are primary and standby database servers.

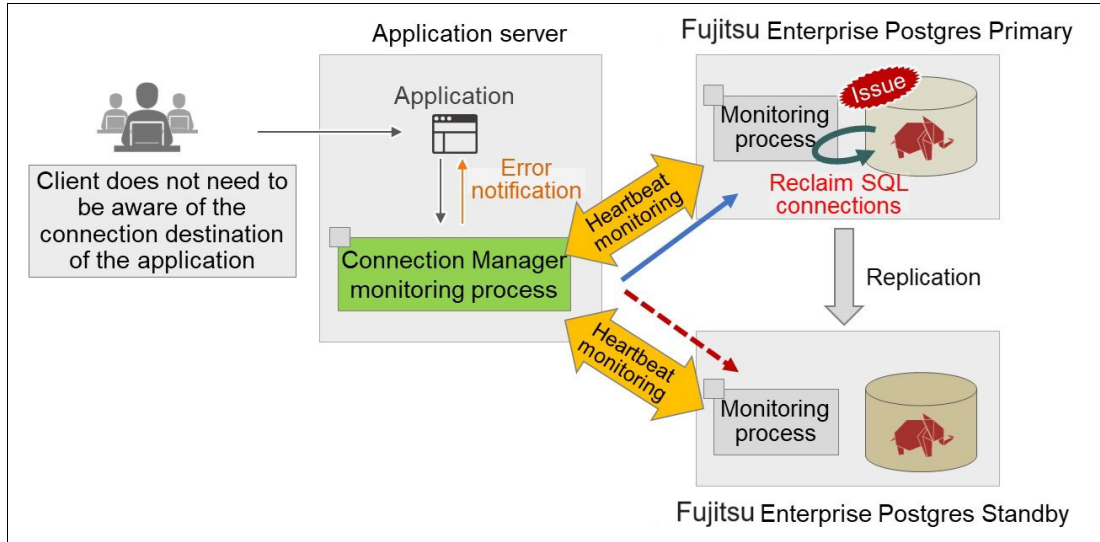


Figure 4-5 Connection Manager heartbeat monitoring and transparent connection support features

Connection Manager features

Connection Manager provides "Heartbeat monitoring" and "Transparent connection support" features, described in Table 4-1, which further improve business application availability.

Table 4-1 Features and their description

Feature	Description
Heartbeat monitoring	<p>Heartbeat monitoring is a keep-alive equivalent timeout function that is implemented at the application layer. This feature detects kernel panics between the server running the application and the server running the Fujitsu Enterprise Postgres instance. It also detects and reports physical server failures and downed inter-server network links between the client and the database instance.</p> <p>The client is notified of an error event through the SQL connection, and the instance is alerted in the form of a force collection of SQL connections with clients that are out of service.</p>
Transparent connection support	<p>As the name implies, the transparent connection support feature makes the connection between the application and the database instances transparent. When an application wants to connect to an instance of an attribute (Primary/Standby) configured with replication, it can connect without knowing which server the instance is running on.</p> <p>Although it might seem to be a single-step process, it involves two steps. In the first step, the application connects to the client driver (libpq), and in the next step, the client driver connects to the Connection Manager process known as the conmgr process. Thereafter, the client driver and the database instance send and receive the query execution data, which does not impact the performance of the SQL query execution.</p>

4.1.2 Application development features

In this section, we describe some of the following application development features:

- ▶ Embedded SQL, Java, and Open Database Connectivity integration.
- ▶ Oracle compatible features.
- ▶ Fujitsu Enterprise Postgres supported open-source software peripheral devices

Embedded SQL, Java, and Open Database Connectivity integration

Encryption Fujitsu Enterprise Postgres supports the JDBC driver, Open Database Connectivity (ODBC) Driver, C library (libpq), and embedded SQL in C on the IBM LinuxONE platform.

The application development interface that is provided by Fujitsu Enterprise Postgres is compatible with PostgreSQL. The Fujitsu Enterprise Postgres client installation provides JDBC driver files for Java SE Development Kit or JRE 6 onwards and ODBC 3.5.

For more information on this topic, see chapter 7 in the IBM Redbooks publication, [Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE, SG24-8499](#), where we described the setup of JDBC, ODBC drivers, examples of using the libpq library and compiling a sample code, and embedded SQL in C.

Note: For more information about application development instructions, see the [Fujitsu Enterprise Postgres Application Development Guide](#)

Oracle compatible features

Table 4-2 lists the features that are compatible with Oracle databases. These features ensure that any application that uses an Oracle database also can use Fujitsu Enterprise Postgres.

Table 4-2 Oracle compatible features

Category		Feature	
		Item	Overview
SQL	Queries	Outer join operator (+)	Operator for outer joins.
		DUAL table	Table provided by the system.
	Functions	DECODE	Compares values, and if they match, returns a corresponding value.
		SUBSTR	Extracts part of a string by using characters to specify position and length.
		NVL	Returns a substitute value when a value is NULL.
Package	DBMS_OUTPUT	Sends messages to clients.	
	UTL_FILE	Enables text file operations.	
	DBMS_SQL	Enables dynamic SQL execution.	

Note: For more information about the Oracle compatibility feature usage and precautions, see the [Fujitsu Enterprise Postgres Installation and Setup Guide](#).

Fujitsu Enterprise Postgres supported open-source software peripheral devices

Fujitsu Enterprise Postgres supports PostgreSQL **contrib** modules, and extensions that are provided by external projects. These **contrib** modules are packaged with Fujitsu Enterprise Postgres so that users get the database management software and the peripheral tools for development, management, and operations needed for Fujitsu Enterprise Postgres.

The **contrib** modules that are listed in “OSS software peripheral devices that are supported by Fujitsu” on page 95 are supported by the Fujitsu development team for resolving issues, which means that customers do not need to wait for the community to provide a solution. Instead, customers get enterprise-grade 24x7 support for the database software and database management tools.

Table 4-3 OSS software peripheral devices that are supported by Fujitsu

Description	Open-Source software name	Version and level
Oracle-compatible SQL features	Orafce	3.8.0
Failover, connection pooling, and load balancing	Pgpool-II	4.1
Connection to the Oracle database server	oracle_fdw	2.2.0
Collection and accumulation of statistics	pg_statsinfo	12.0
Performance tuning (statistics management, and query tuning)	pg_hint_plan	12.1.3.5
	pg_dbms_stats	1.3.11
Table reorganization	pg_repack	1.4.5
Backup and restore management	pg_rman	1.3.9
Log analysis	pgBadger	11.1
Full-text search (multibyte)	pg_bigm	1.2
JDBC driver	PostgreSQL JDBC driver	42.2.8
ODBC driver	psqlODBC	12.01.0000

Note: For more information about the ORAFCE function within Fujitsu Enterprise Postgres, see the following file:
fujitsuEnterprisePostgresInstallDir/share/doc/extension/README.asciidoc

For OSS peripheral tools version information, see the [Fujitsu Enterprise Postgres General Description Guide](#).

4.1.3 Security

Data security is an important area of focus for organizations. Organizations must ensure that the confidentiality and safety of the data that is available in electronic and physical form. Organizations with even the most sophisticated technology have faced data breaches that have resulted in expensive consequences of a legal nature. More importantly, damage might

be done to their reputation, which has a long-term impact on customer trust and diminished confidence and morale in doing business or using services.

Data security generally follows a layered framework that covers every touch point that is vulnerable to security threats. These layers include human, physical, application, databases, and other detection technologies (see Figure 4-6). The aim of a layered security framework is to make sure that a breach in one layer does not compromise the other layers, and in the worst case, the entire data security.

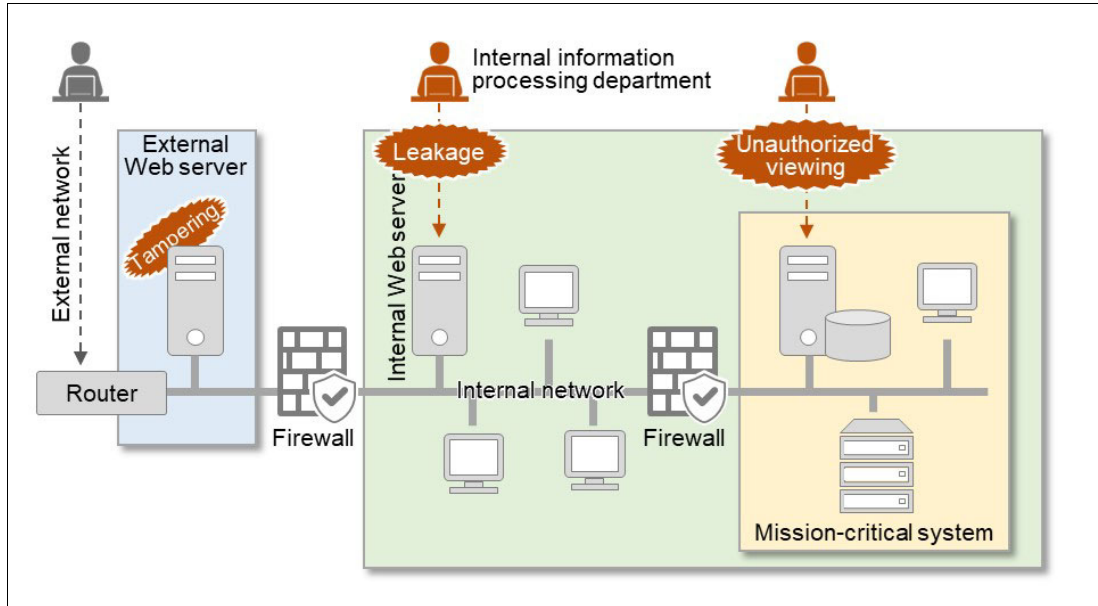


Figure 4-6 Layered security architecture and common vulnerable data security breach points

The layered security framework secures the three states of data through the implementation of security policies. These three data states can be classified as:

- ▶ Data at rest
- ▶ Data in motion
- ▶ Data in use

Fujitsu Transparent Data Encryption (TDE) and Data Masking are the Fujitsu solution for securing data at rest and data in use. TDE and Data Masking are included in Fujitsu Enterprise Postgres Advanced Edition. In Fujitsu Enterprise Postgres on IBM LinuxONE, TDE is further enhanced through integration with a Central Processor Assist for Cryptographic Function (CPACF) instructions set on IBM LinuxONE to make Fujitsu Enterprise Postgres the most secure enterprise-grade Postgres on the planet.

Fujitsu Transparent Data Encryption

Encryption is regarded as one of the best data protection techniques that are available. Through encryption, the user can scramble the data by using encryption keys, and then use decryption keys to decipher the data. TDE is an encryption method that is transparent to the user and to the application, which means no change is required in the application or the existing access policies for using TDE.

Fujitsu Enterprise Postgres comes with TDE integrated. Unlike some other proprietary RDBMS products, you are not required to purchase extra products to gain this function.

Fujitsu Enterprise Postgres supports both file-based keystore management and hardware-based keystore management.

Figure 4-7 shows Fujitsu Enterprise Postgres integrated to use CryptoCard based encryption and keystore management.

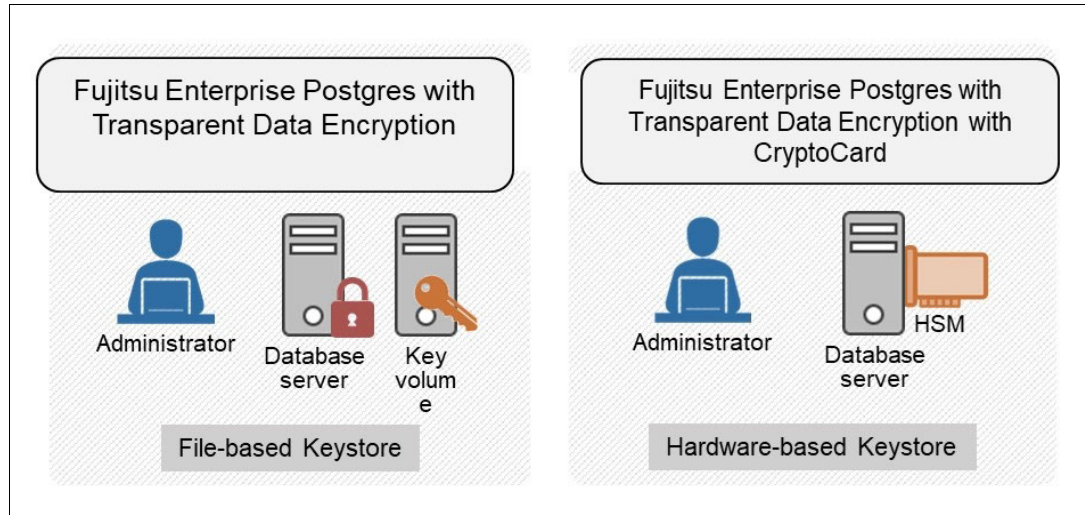


Figure 4-7 Transparent Data Encryption: File-based and hardware security module-based keystore management

TDE key features

The following are the key features of TDE:

- ▶ TDE uses the Advanced Encryption Standard (AES) as its encryption algorithm. AES was adopted as a standard in 2002 by the United States Federal Government and is used throughout the world.
- ▶ TDE provides encryption at the file level (AES-128 and AES-256), essentially solving the issue of protecting data at rest.
- ▶ TDE fulfills requirements for the Payment Card Industry Data Security Standard (PCI DSS) and allows confidential information such as credit card numbers to be made unrecognizable on disk.
- ▶ Data is automatically encrypted and decrypted when it is written and read, so manual key management is not required. Even if an attacker gets through all the access controls and connects to the server, they cannot access the data because the OS file is encrypted.
- ▶ Encryption is extended to WALs (also known as transaction logs), backups, temporary tables, and temporary indexes, thus providing comprehensive security.
- ▶ The encryption algorithm does not alter the size of the object that is being encrypted, so there is no storage overhead.
- ▶ Flexibility to encrypt subset of the data per an organization's data classification rules and policies.
- ▶ Both file-based and hardware-based keystore management mechanisms are supported.
- ▶ Flexibility of using multiple Data Encryption Keys (DEKs), which are encrypted by the Master Encryption Key (MEK).
- ▶ On IBM LinuxONE, MEK is further secured with an IBM LinuxONE CryptoCard hardware security module (HSM) managed secure key.

- ▶ Take advantage of the CPACF in the IBM Z processor to minimize encryption and decryption overhead so that even in situations where previously the minimum encryption target was selected as a tradeoff between performance and security, it is now possible to encrypt all the data of an application.

Figure 4-8 shows a high-level implementation of Fujitsu TDE with openCryptoki for storing master keys on the CryptoCard (also called HSM). openCryptoki allows Fujitsu Enterprise Postgres to transparently use Common Cryptographic Architecture (CCA) and EP11 (Enterprise PKCS #11) firmware in a CryptoCard.

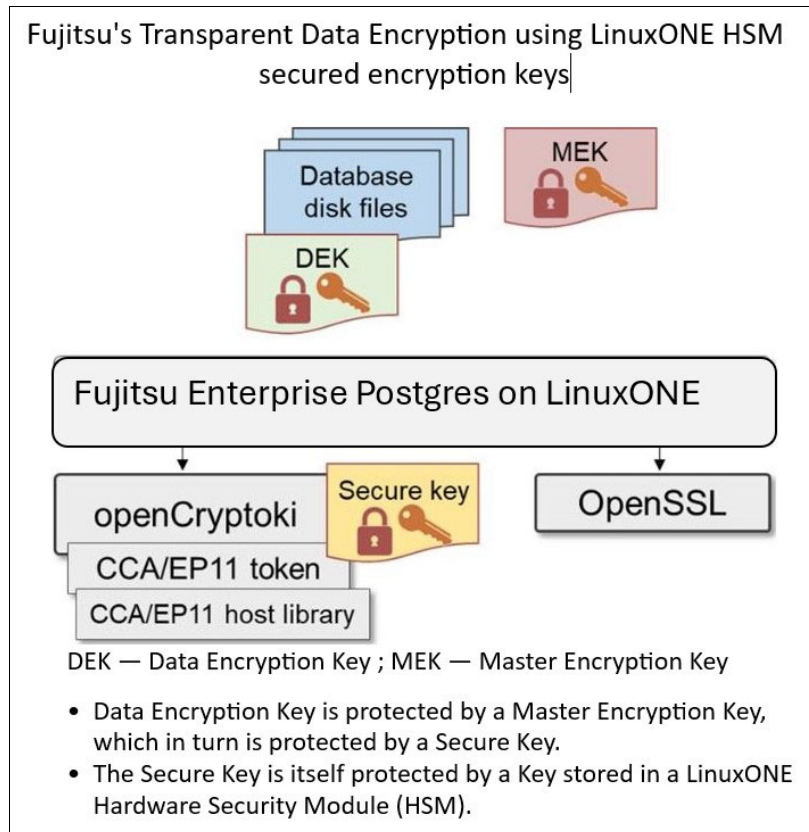


Figure 4-8 Transparent Data Encryption secured by encryption keys

Data masking

Data masking enables data security governance by obfuscating specific columns or part of the columns of tables that store sensitive data while still maintaining the usability of the data.

Here are some common use cases for partial or full obfuscation of information:

- ▶ Test data management

This is the process of sanitizing production data for use in testing of information systems with realistic data without exposing sensitive information to the application development or infrastructure management teams, who might not have the appropriate authority and clearance to access such information. Figure 4-9 shows the data masking use case for test data management.

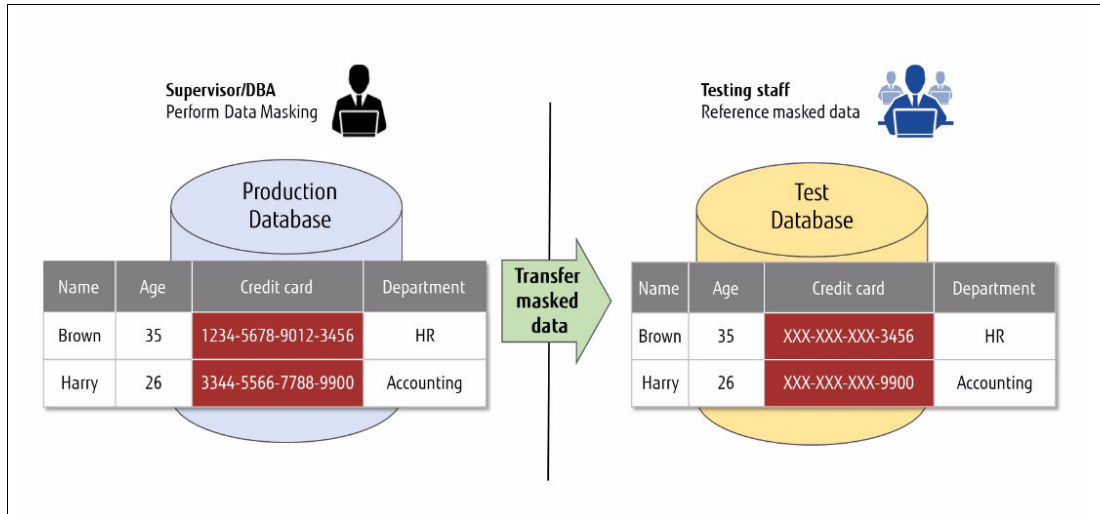


Figure 4-9 Data privacy management by transferring masked production data for testing

► Compliance

Various organizations collect and store large amounts of complex data as part of business operations and use analytics to run their business, which creates data privacy challenges for the organizations to adhere to different data privacy regulations and compliance based on regions, such as General Data Protection Regulation (GDPR), California Consumer Privacy Act (CCPA), PCI-DSS, and HIPAA (Health Insurance Portability and Accountability Act). Data masking is one of the most effective ways to be compliant to various regional and international data privacy legislation and compliance.

► Production data security

While data in production is accessed by people with different roles such as system administrators, customer support, production application maintenance personnel, and business process analysts, it is important that data masking is role-sensitive. Fujitsu Data Masking provides the facility to implement role-sensitive masking. Figure 4-10 shows a production environment example.

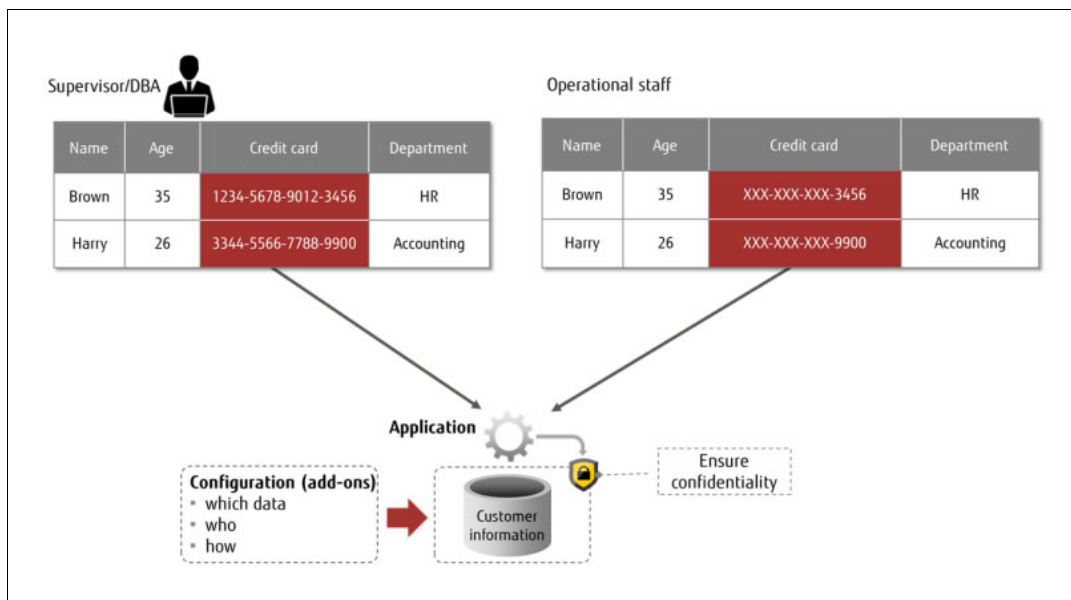


Figure 4-10 Production environment data masking example

The following are some of the features of Fujitsu Data Masking technology:

- ▶ Fujitsu Enterprise Postgres uses a flexible and easy to use policy-based data masking technique. There are advantages of using Fujitsu policy-based Data Masking over other the techniques.
- ▶ Policy-based data masking allows the development of data-sensitive masking policies for different data classifications.
- ▶ Data masking policies can be applied to tables for different columns that come under one data classification or another.
- ▶ After the policy-based data masking is applied, the policies can be disabled or enabled as required without needing to remove or reapply them.
- ▶ The Fujitsu Data Masking feature is irreversible because the relationship between the original data and the masked data is severed during the masking policy implementation to de-risk any reverse engineering to re-create the production data.
- ▶ The data masking policy is applied at the time that the data is accessed, so queried data is modified according to the masking policy before the results are returned in the query chain.
- ▶ Data masking policies can be configured to be applied to specific roles / conditions so that the original data cannot be viewed without the appropriate privileges.
- ▶ Referential integrity is maintained in the source database because the data masking policy is applied during the data access process.
- ▶ Masking policies generate data that can be same every time masked data is accessed, which maintains test data consistency across multiple iterations of accessing original data.
- ▶ Availability of catalog tables for querying data masking policy information to assess the current sensitive data policy state of a database.

Types of data masking

There are three different types of data masking that are available:

- ▶ Full masking: A whole column value can be obfuscated with alternative values.
- ▶ Partial masking: Part of a column value can be obfuscated with alternative values.
- ▶ Regular expression masking: The value of a column can be obfuscated through a regular expression statement.

Figure 4-11 shows the three different types of data masking techniques that are offered with Fujitsu Enterprise Postgres Advanced Edition

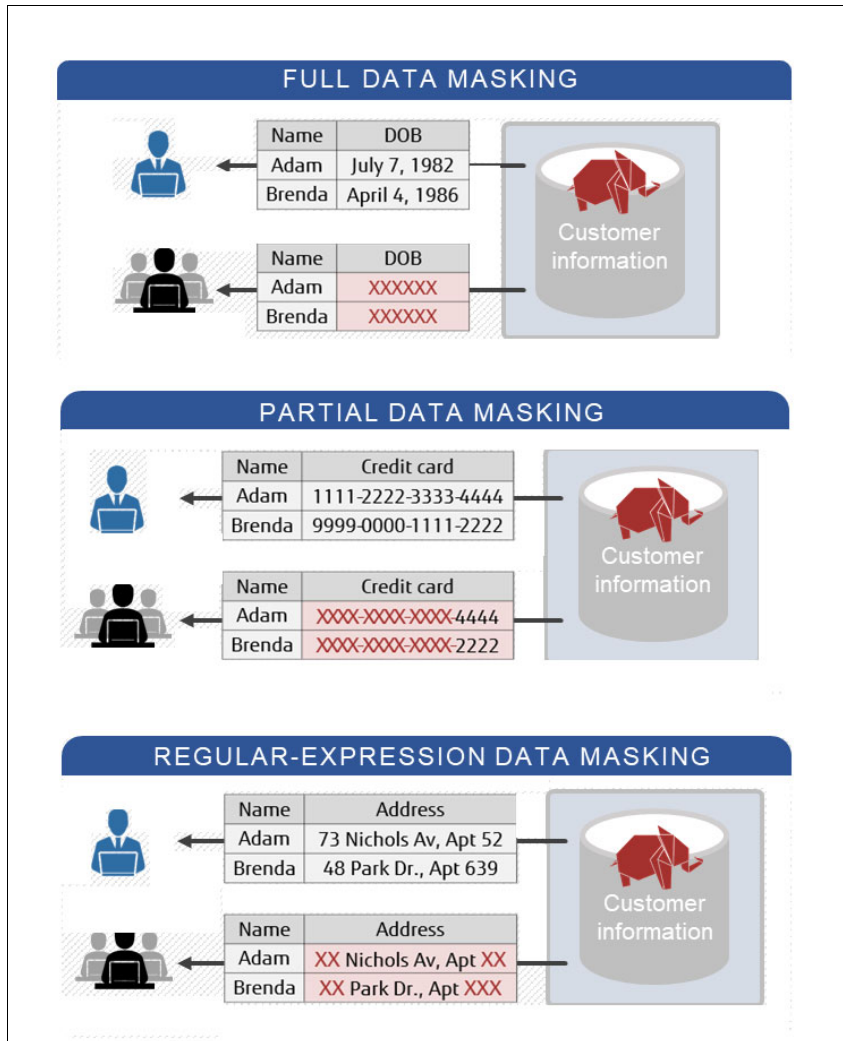


Figure 4-11 Types of data masking that are available with Fujitsu Enterprise Postgres

Data masking policy functions

Fujitsu Enterprise Postgres provides five data masking policy functions for creating, altering, enabling, disabling, and deleting the data masking policies. Figure 4-4 provides the list of these functions.

Table 4-4 Data masking policy functions

Data masking policy function	Description
pgx_create_confidential_policy	Create masking policies.
pgx_alter_confidential_policy	Change masking policies.
pgx_enable_confidential_policy	Enable or disable masking policies.
pgx_update_confidential_values	Changes replacement characters when full masking is specified for masking type.
pgx_drop_confidential_policy	Deletes masking policies.

Audit log

Dedicated audit logging is a unique feature to Fujitsu Enterprise Postgres that helps organizations manage data accountability, traceability, and auditability. **Audit Log** is an important security feature that provides a level of protection while also meeting compliance expectations and protecting the brand and data asset value of an organization.

OSS PostgreSQL implements basic statement logging through the standard logging parameter **log_statement = all**. This implementation is acceptable for monitoring and other use cases, but this method does not provide the level of information that is required by security analysts for an audit. The information that is generated is not enough to list what the user requested. Ideally, the configuration must also establish what happened while the database was satisfying each request.

Fujitsu Enterprise Postgres Audit Logging enables organizations to trace and audit the usage of sensitive data and connection attempts of the database. Audit Logging provides a clear picture of data access by logging the following details:

- ▶ What data is accessed.
- ▶ When the data is accessed.
- ▶ How the data is accessed and who accessed the data.

The information that is provided by audit logs can be used to counter security threats such as:

- ▶ Spoofing
- ▶ Unauthorized database access
- ▶ Privilege misuse

The Audit Log feature uses the **pgaudit** utility, which enables retrieval of details relating to database access as an audit log. Additionally, audit logs are externalized to a dedicated log file or server log to implement an efficient and accurate log monitoring.

Figure 4-12 shows the functioning differences between the server logs and the dedicated audit log (session log). The server log configuration parameters are set in the **postgresql.conf** file, but the audit log configuration is managed in the **pgaudit.conf** configuration file.

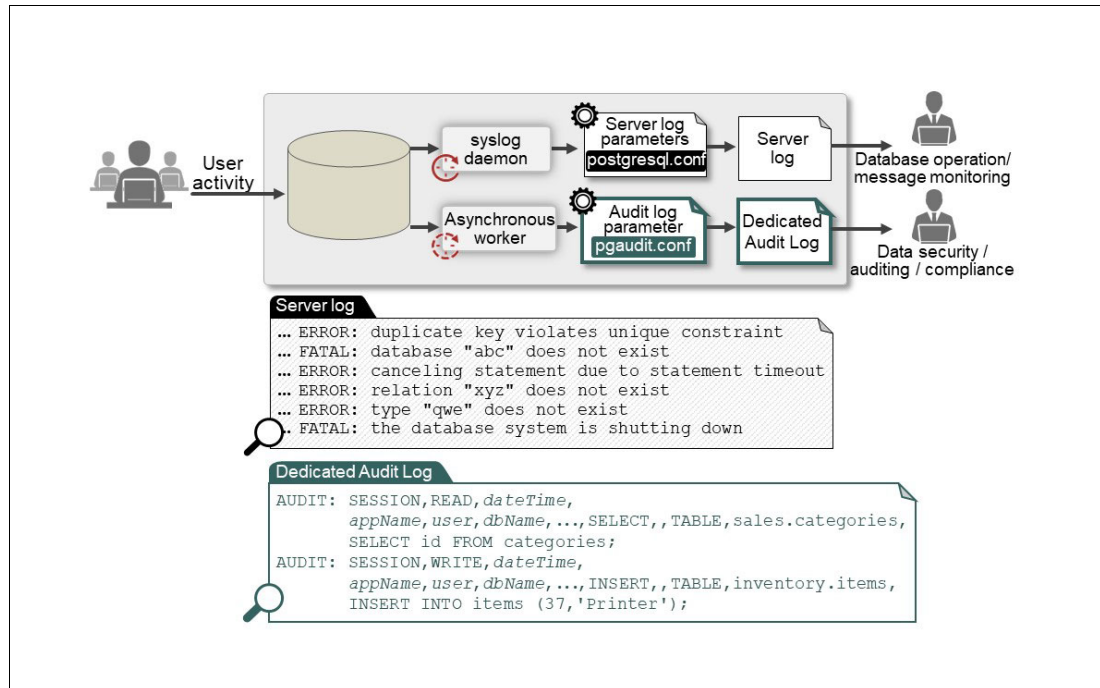


Figure 4-12 Fujitsu Enterprise Postgres Audit Login action

With **pgaudit**, the following two types of audit log can be output:

- ▶ Session Audit Log
- ▶ Object Audit Log

Session Audit Logging

Session Audit Logging provides information about all the SQL statements that are run by the user in the background, information about starting and connecting to databases, and error information.

Object Audit Logging

When SELECT, INSERT, UPDATE, and DELETE are run for specific objects (tables and columns), Object Audit logging outputs them as a log. Object Audit Logging outputs object operations for which privileges are assigned to specified roles, as a log. Object Audit Logging can control log output at an even finer level of granularity than Session Audit Logging.

4.1.4 Performance

One of the driving forces behind business innovation is using information to gain insight about business and respond as needed. This task becomes more challenging because the world generates 10 zettabytes (a trillion gigabytes) of data every year, and this number is projected to increase to 175 zettabytes in 2025.

Regardless of the choices that businesses make, IT systems must adapt to keep pace with exponential data growth and be able to quickly analyze growing amounts of data. Every minute that is saved in ingesting and analyzing data has tremendous value, so the demand to quickly perform analysis puts tremendous pressure on IT systems. Business and data analysts are always looking for faster ways of processing data aggregation and manipulation supporting business leaders to make business decisions.

Fujitsu Enterprise Postgres offers three performance features for enhancing ingesting and analyzing data by fully optimizing the processor, memory, storage, and I/O resources:

- ▶ VCI
- ▶ Fujitsu High-Speed Data Load
- ▶ Global Meta Cache (GMC)

Vertical Clustered Index: Increased aggregation performance

Vertical Clustered Index (VCI) is a proprietary implementation of in-memory columnar storage (IMCS), which provides faster data analysis to support business intelligence. VCI provides faster aggregation performance while reducing the I/O of processing the in-memory data. The design thinking behind VCI is to enable the businesses to perform real-time analytics on changing data sets without needing to Extract, Transform, and Load (ETL) data from Online Transaction Processing (OLTP) to Online Analytical Processing (OLAP) systems, and without any extra cost.

Columnar index

Columnar storage architecture has gained attention with the introduction of Hybrid Transactional Analytical Processing (HTAP), which is an alternative to merging OLTP with OLAP systems. One of the major challenges that exists in various product implementations of HTAP processing is maintaining the ACID compliance of an RDBMS. The goal of HTAP data processing is to remove the ETL processing to an extent that enables real-time analytics on the OLTP systems before they go to OLAP systems through ETL, which traditionally has been the de facto method to connect OLTP and OLAP DBMSs. Although row-oriented data is better suited to OLTP processing, it's the column-oriented data that is best suited for data analysis because it improves aggregation performance.

Aggregations usually process a significant amount of data from a particular column or columns. For such aggregations in the traditional row data structure, columns that are not queried are also fetched, which leads to inefficient memory and CPU cache usage and causing slower query processing. However, columnar storage uses vertical architecture for placing rows adjacently because the query processing does not perform scans on columns that are not in the scope of aggregation computation, which improves aggregation query performance. VCI is the Fujitsu columnar storage implementation offering in the form of an index, hence the name VCI.

VCI has two components to achieve aggregation performance: One is on disk, and another one is in memory. Both the components are managed by the VCI Analysis Engine.

Figure 4-13 shows that the VCI Analysis Engine combines VCI columnar storage on disk by using the `pgx_prewarm_vci` utility to preload VCI pages and make it memory resident. VCI data is stored in the dedicated portion of the shared buffers on memory. VCI memory allocation share is controlled by the configuration parameter `reserve_buffer_ratio`.

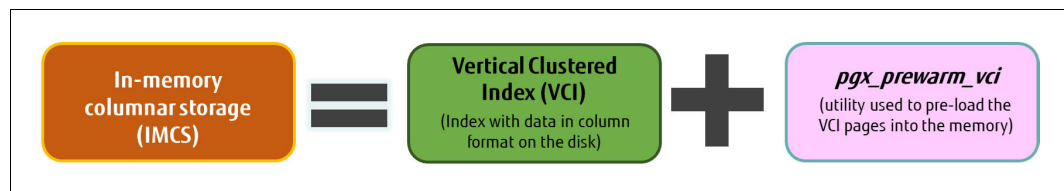


Figure 4-13 VCI disk and memory components

VCI benefits

VCI provides the following benefits:

- ▶ Minimizes the impact on existing jobs and performs aggregation by using job data in real time.
- ▶ VCI is crash-safe. It also stores data on the disk, so aggregation jobs can be quickly resumed by using a VCI even if a failure occurs (when an instance is restarted).
- ▶ If the amount of memory that is used by VCI exceeds the set value, aggregation can continue by using VCI data on the disk to avoid impacting on-going operations.
- ▶ VCI is provided as an index, so no application modification is required.
- ▶ In HA architectures, VCI provides the opportunity to optimize the extra computing power that is available with standby servers for performing resource-intensive data analysis faster.

VCI features

The following are some of the features of VCI:

- ▶ **Disk compression:** Compresses VCI data on the disk, minimizing the required disk space. Even if disk access is required, the read overhead is low.
- ▶ **Parallel scan:** Enhances aggregation performance by distributing aggregation processes to multiple CPU cores and then processing them in parallel.
- ▶ **Preload feature:** Preloads VCI data into memory before an application scans VCI and ensures stable query response times.
- ▶ **Stable buffer feature:** Maintains VCI data in memory, which improves query performance because it reduces disk I/Os by avoiding VCI eviction from memory by other jobs.

Note: The Preload and Stable buffer features keep VCI data in memory and minimize disk I/Os on each aggregation process.

For more information about the evaluation criteria for installing VCI, resource estimation, and configuration steps, see 9.1, “Installing Vertical Clustered Index (VCI)”, of the [Fujitsu Enterprise Postgres Operation Guide](#).

Query planner

VCI optimizes the available CPU and memory within the server to enhance scan performance. It further speeds up execution time for the following use cases:

- ▶ Single table processing
- ▶ Queries processing large aggregations, such as simultaneous sum and average operations
- ▶ Queries fetching and processing many rows in the tables

Before the aggregation is performed, the **query planner** computes the execution cost, and based on an algorithm, it implements the most cost-efficient plan to determine whether VCI processing is regardless of the availability of VCI.

Through VCI, Fujitsu Enterprise Postgres enables organizations to consolidate critical business intelligence analytics queries on the OLTP systems from OLAP systems.

Furthermore, the parallelization of the aggregation process further reduces the query execution time with VCI. Together, both features provide HTAP system design capability to the database designers.

Fujitsu High-Speed Data Load: `pgx_loader`

Fujitsu High-Speed Data Load is designed to perform bulk loading by using multiple parallel processes that depend on the availability of the number of CPU cores or virtual Integrated Facility for Linux (IFL) processors that are available. This approach makes the best use of the available virtual IFL processors or CPU cores without pre-configuring and performance tuning the environment.

Fujitsu High-Speed Data Load distributes the data from an input file to several parallel workers. Each worker performs data conversion, table creation, and index creation, which tremendously reduces the data load time.

Global Meta Cache

Caching is considered an important database performance feature. Database performance experts try to reduce I/O as much as possible by using different techniques at hardware and software layers. Hence, fetching data from storage frequently impacts the performance of workloads that use large amounts of random I/O, such as OLTP and HTAP workloads. This problem is overcome in PostgreSQL by caching data in memory, which tremendously improves performance. PostgreSQL caches table data, indexes, and query execution plans.

Query processing goes through a few steps before a query execution plan is created. The steps include parsing the query, creating a plan, and running the plan. To perform these steps, the PostgreSQL process accesses the system catalogs, during which the system catalog data is cached in memory per process.

Moreover, the metadata that is required to access the tables that are referenced in SQL queries and stored in catalog tables is also aggregated and cached. This cached information in the memory is called a meta cache. The purpose of a meta cache is to improve the query processing performance by providing metadata in memory to avoid metadata searches in the catalog table each time. As the number of transactions and connections and their size increases, the cache size increases because the metadata is cached per process, which increases the system's overall meta cache and causes cache bloat, which impacts the performance. This performance issue is resolved by GMC.

Fujitsu GMC is a method to avoid cache bloat by centrally managing the meta cache in shared memory instead of per process. Because the meta cache is managed centrally, it is called GMC.Fujitsu

4.1.5 Performance tuning

Fujitsu Enterprise Postgres selects the least expensive query plan based on the query planner estimates. The query planner estimates the cost by using algorithms that parse SQL statements and statistical information from catalog tables. For mission-critical systems, it might not be possible to have the most updated statistics. Often, the volatility of the statistics of database objects leads to unstable query plans and impacts performance. To resolve these issues, Fujitsu Enterprise Postgres uses optimizer hints and locked statistics to achieve stable query performance.

Optimizer hints

Optimizer hints enable Fujitsu Enterprise Postgres to stabilize the query plan in mission-critical systems. Users can use `pg_hint_plan` to specify a query plan as a hint in each individual SQL statement.

Fujitsu Enterprise Postgres uses the open-source module `pg_hint_plan` to provide optimizer hints for query planning. The query planner estimates the cost of each possible plan that is available for the SQL statement and picks the low-cost plan because the query planner considers it the best plan, but that plan might not be best because the query planner does not know some details like the association between the columns. Optimizer hints come handy in such situations because it suggests to the planner to choose a specified plan.

Note: For more information about `pg_hint_plan`, see [GitHub](#)

Fixed statistical information

Locked statistics is a feature that locks the various relational statistics in custom views. During query plan generation, the optimizer component of the server calculates the costs based on these locked statistics. This way, query plan generation is stabilized because every time these locked statistics are used to generate the plan, the risk of a sudden execution plan change in the customer environment is reduced. This function is achieved by installing and using `pg_dbms_stats`. Locked statistics also help to reproduce the same query plan in both testing and production environments.

Note: For more information about `pg_dbms_stats`, see [GitHub](#).

Data compression by using IBM zEnterprise Data Compression

Fujitsu Enterprise Postgres provides a unique backup and recovery feature through the `pgx_dmpall` and `pgx_rcvall` commands, which enable administrators to perform online backups with continuous archiving or recover backed up data and archived WALs with a single action.

In addition, Fujitsu Enterprise Postgres on IBM LinuxONE works with the on-chip compression accelerator IBM zEnterprise Data Compression (IBM zEDC) to ensure efficient backup operations in the event of database or hardware failures. IBM zEDC reduces CPU costs by 90% and processing time during data compression by 40% compared to traditional software compression. For more information about IBM zEDC, see [Data compression with the Integrated Accelerator for zEDC](#).

4.1.6 Fujitsu developed database management software

Fujitsu has developed specific software tools to help database management. One such tool is WebAdmin.

WebAdmin is the Fujitsu graphical user interface (GUI) tool for many tasks, such as database installation and database operations management and is used for the following tasks:

- ▶ Fujitsu Enterprise Postgres setup. Instances can be created easily with minimal input because the tool automatically determines the optimal settings for operation.
- ▶ Creating and monitoring a streaming replication cluster.
- ▶ Database backups and recovery.
- ▶ Manage Fujitsu Enterprise Postgres instances in a single server or instances that are spread across multiple servers.

WebAdmin features include:

- ▶ Fast Recovery, which uses WebAdmin as a technology to shorten the rebuild time.

- Recovery that uses WebAdmin requires less time and effort because WebAdmin automatically determines the scope of the operation.
- ▶ Wallet feature.
 - The Wallet feature in WebAdmin is used to conveniently create and store instance username and passwords.
 - This feature can be used to create usernames and passwords when creating remote standby instances through WebAdmin.
 - After the credentials are created in Wallet, they can be used repeatedly.
- ▶ Anomaly Detection and Resolution.

When certain operations are performed through the command line interface (CLI), they can cause anomalies in WebAdmin. These operations are:

- Changes to the **port** and **backup_destination** parameters in the postgresql.conf file.
- Changes to the MC configuration of cluster replication that is added through WebAdmin.

WebAdmin checks for anomalies when an instance is selected for viewing or when any instance operation is performed.

WebAdmin configurations

WebAdmin can be installed as either of two configurations:

- ▶ Single server
- ▶ Multi-server

Note: WebAdmin does not support encrypted communication between a browser and server or between servers. Hence, when using WebAdmin in either configuration, the network communication path should be built on a private network with no external access.

For more information about WebAdmin installation instructions, see [Fujitsu Enterprise Postgres Installation/Setup](#).

4.2 Migrating to Fujitsu Enterprise Postgres

In this section, we describe migration to Fujitsu Enterprise Postgres, the Fujitsu migration tool, and the Fujitsu Enterprise Postgres backup tools that support data migration.

4.2.1 Migrating from OSS PostgreSQL to Fujitsu Enterprise Postgres

OSS PostgreSQL can be migrated to Fujitsu Enterprise Postgres in either an offline or online mode. In an offline migration, the database instance is stopped and a consistent data and schema backup is taken, and in an online migration, PostgreSQL logical replication is used to migrate the changes from OSS PostgreSQL to Fujitsu Enterprise Postgres.

Offline migration is carried out during a migration window and the database is not available for write operations. In an offline migration, the definition of each selected object is read and written in SQL script. This SQL script is used to migrate the Postgres database to Fujitsu Enterprise Postgres.

The total time of offline migration depends on the database size and time that is taken to lift and shift the selected objects. The downtime in the case of an offline backup significantly increases when migrating large databases.

Typical steps to perform an offline migration from OSS PostgreSQL to Fujitsu Enterprise Postgres

To perform an offline migration, typically the following steps are followed:

- ▶ Run database instance in read-only mode.
- ▶ Take a logical backup by using the PostgreSQL utility `pg_dump` to back up a single database, or use `pg_dumpall` to back up global objects that are common to the database cluster
- ▶ Move the backup file to the target server
- ▶ Restore the backup to the Fujitsu Enterprise Postgres instance using `postgres SET`

Typical steps to perform an online migration from OSS PostgreSQL to Fujitsu Enterprise Postgres

Online migration is used when it is not acceptable to have any system downtime.

We can use Postgres logical replication to migrate to Fujitsu Enterprise Postgres. Logical replication is a method of replicating data objects and their changes based on their replication identity (usually a primary key). Logical replication works on the publisher-subscriber model, where the primary server is defined as publisher and the destination server as subscriber. For a detailed example of performing online migration please refer to the Red Book “Data Serving with Fujitsu Enterprise Postgres on IBM LinuxONE”

4.3 EnterpriseDB Postgres Advanced Server on IBM LinuxONE

This section provides information on EnterpriseDB (EDB) Postgres Advanced Server deployments scenarios, failover mechanisms, use-cases and benefits of using EnterpriseDB Postgres on IBM LinuxONE.

4.3.1 Overview

As organizations modernize and consolidate their existing applications, it is essential to have the flexibility to manage and deploy the entire application portfolio on infrastructures that offer flexibility, security and scalability.

4.3.2 EnterpriseDB Postgres with IBM LinuxONE

Deploying EnterpriseDB Postgres on an infrastructure such as IBM LinuxONE can contribute to fewer greenhouse gas emissions and a more environmentally sustainable IT environment. IBM LinuxONE are designed to make a powerful improvement in sustainability by decreasing electricity consumption, reducing the number of standing servers, and enabling high compute and resource utilization.

Some the major advantages of the having EnterpriseDB Postgres deployed on IBM LinuxONE are

- ▶ Performance - Higher response time combined with extreme scalability for data management and analytics workloads.
- ▶ Combining EnterpriseDB Postgres with IBM LinuxONE - offers the high levels of vertical and horizontal scalability, the flexibility and high utilization levels that can't be achieved with distributed infrastructures. Also, with high VM or Container saturation levels on a single LinuxONE server - it offers great economic advantages as well as reduces power and data center footprint from a sustainability perspective.

EnterpriseDB Postgres customers benefit from the most reliable, high-performing, flexible, open, and cost-effective data management platform available.

4.4 EnterpriseDB Postgres deployment scenarios

This section provides detailed information on EnterpriseDB Postgres deployments scenarios, Failover mechanisms, use-cases and benefits of using EnterpriseDB Postgres on IBM LinuxONE.

4.4.1 IBM LinuxONE single box per site: Asynchronous streaming

This deployment option harnesses the inherent IBM LinuxONE reliability features to provide a highly available EDB Postgres deployment architecture. In the production side (DC), the primary and the standby nodes are deployed on different LPARs on the same LinuxONE infrastructure. With EAL5+ LPAR Isolation features, the primary and standby nodes are considered as an air-gapped isolated partitions. The nodes can be allocated with either dedicated or shared mode of CPU resources allocations. Also, since the both the nodes are on the same server, the interconnectivity for the streaming can be achieved using internal high-speed Hipersockets and SMC-D based network connectivity's. This would in-turn provide enhanced performance and security.

From a disaster recovery perspective, standby nodes are deployed on to the Disaster recovery site and a asynchronous streaming is setup between the primary node and the disaster recovery standby nodes.

The EnterpriseDB Failover Manager (EFM) agent manages the nodes health and detects any failures and takes appropriate failover recovery actions. Within the production DC, failover happens to the current standby node, and it attaches the virtual IP to the newly promoted primary node. This automatic failover provides near zero downtime maintenance with controlled switchover with minimal data loss. In case of a disaster recovery, manual failover mechanism is activated for DR switchover. Figure 4-14

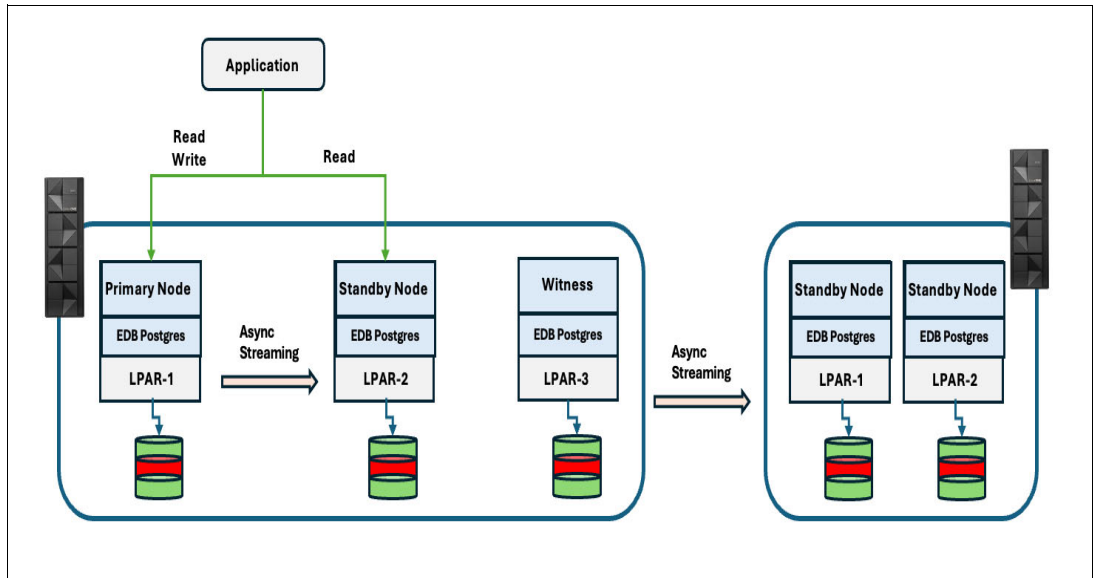
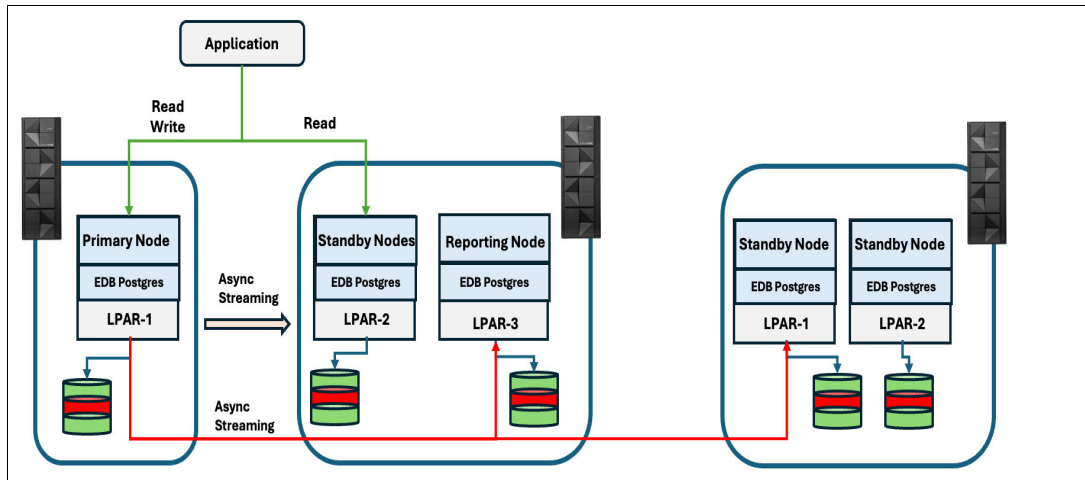


Figure 4-14 Asynchronous streaming

4.4.2 IBM LinuxONE Two Box Option - Asynchronous streaming

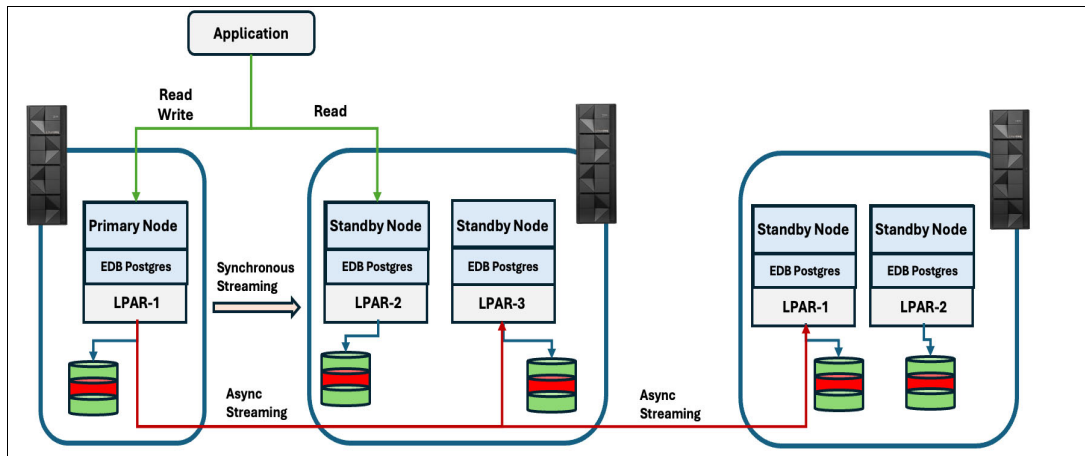
Here the primary difference that of the single system option, is to have the standby nodes on the secondary machine in the production data center. This option is applied only for organizations where there is a specific requirement on having two physical infrastructures hosting workloads on the production data centers. Also to make sure, we utilize the share everything architecture of IBM LinuxONE, we can deploy a separate node for reporting purposes, so that the physical nodes resources can be utilized optimally.



4.4.3 Deployment Option-3 - Synchronous Streaming

This option utilizes synchronous streaming, where-in the primary nodes write-ahead logs (WAL) changes are applied to the standby node and a separate asynchronous streaming is applied to the second standby node. In the event of a crash, we will be able to recover the database using the logs applied to the standby node and any changes that have not been applied to the data pages can be redone from the WAL records.

With this kind of streaming application, the EnterpriseDB deployment can achieve zero RPO.



4.5 EDB Postgres installation and configuration

In this section, we would install and configure EnterpriseDB Postgres v14 on a Red hat Linux guests running on an IBM LinuxONE. The deployment would include the installation of primary, secondary and witness nodes as described in the previous section.

4.5.1 Red Hat Enterprise Linux environmental prerequisites

We have used Red Hat Enterprise Linux v8.10 as the virtual machines for deploying EDB Postgres. As we plan to install EDV v14.10, It's important to verify and validate whether the specific versions are certified on the Red hat Linux v8.10.

Example 4-1 Red hat version

```
[root@rdbk1101 ~]# cat /etc/redhat-release
Red Hat Enterprise Linux release 8.10 (Ootpa)
[root@rdbk1101 ~]# uname -srv
Linux 4.18.0-553.8.1.el8_10.s390x #1 SMP Fri Jun 14 02:46:29 EDT 2024
[root@rdbk1101 ~]#
```

We have downloaded EDB v14.10 and placed in the specific folder "/installables". Before installing, firewall has to be stopped and an EDB Postgres repository has to be created for 'Yum' or 'dnf' commands to be used for installing the binaries.

Example 4-2 Firewall disabling

```
[root@rdbk1101 ~]# systemctl status firewalld
? firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor
   preset: enabled)
   Active: active (running) since Mon 2024-07-22 09:55:40 EDT; 2 weeks 5 days ago
[root@rdbk1101 ~]# systemctl stop firewalld
[root@rdbk1101 ~]# systemctl status firewalld
? firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor
   preset: enabled)
```

Active: inactive (dead) since Sun 2024-08-11 07:02:48 EDT; 3s ago

4.5.2 EnterpriseDB Postgres installation

The next step is to untar the EDB postgres installables, these files would be used for creating the repository

Example 4-3 Untar the EDB Postgres installables.

```
[root@rdbk1101 installables] # tar -zxvf epee_14_rhel8-s390x-20220705.tar.gz
epee_14_rhel8-s390x/
epee_14_rhel8-s390x/edb-xdb-libs-7.2.0-1.rhel8.s390x.rpm
epee_14_rhel8-s390x/edb-odbc-13.02.0000.01-1.rhel8.s390x.rpm
epee_14_rhel8-s390x/edb-as14-mongo_fdw-5.4.0-1.rhel8.s390x.rpm
:
:
epee_14_rhel8-s390x/edb-as14-postgis3-utils-3.1.5-1.rhel8.s390x.rpm
epee_14_rhel8-s390x/repodata/4167d095da5f93ec85be788788d33feff6ecf34b0d4184edca4a4
c02fc1e6170-primary.xml.gz
epee_14_rhel8-s390x/repodata/repomd.xml
:
:
epee_14_rhel8-s390x/edb-as14-pgsnmpd-1.0-1.rhel8.s390x.rpm
[root@rdbk1101 installables] #
```

Once the installables are untared, the next step is to create a repository as below.

Example 4-4 Creating EDB Postgres repository

```
[root@rdbk1101 yum.repos.d]# cat edb.repo
[EDB]
name=EDB repo
baseurl=file:///installables/epee_14_rhel8-s390x
enabled=1
gpgcheck=0
[root@rdbk1101 yum.repos.d]# yum repolist
Updating Subscription Management repositories.
repo id                repo name
EDB                    EDB repo
rhel-8-for-s390x-appstream-rpms  Red Hat Enterprise Linux 8 for IBM z Systems -
AppStream (RPMs)
rhel-8-for-s390x-baseos-rpms     Red Hat Enterprise Linux 8 for IBM z Systems -
BaseOS (RPMs)
[root@rdbk1101 yum.repos.d]#
```

Now that the repository is setup, we can go ahead and install EDB postgres on the virtual machine.

Example 4-5 Installing EDB postgres

```
[root@rdbk1101 ~] # yum install edb-as14-server
Updating Subscription Management repositories.
Last metadata expiration check: 1:03:31 ago on Sun 11 Aug 2024 07:57:52 AM EDT.
Dependencies resolved.
```

```

=====
Package      Arch Version      Repository      Size
=====
Installing:
edb-as14-server      s390x 14.4.0-1.rhel8      EDB
9.6 k
Installing dependencies:
boost-atomic          s390x 1.66.0-13.e18
rhel-8-for-s390x-appstream-rpms 14 k
boost-chrono          s390x 1.66.0-13.e18
:
Transaction Summary
=====
Install      28 Packages
Downgrade   8 Packages
Total size: 91 M
Total download size: 56 M
Is this ok [y/N]: y
Downloading Packages:
(1/16): bcc-0.16.0-3.e18.s390x.rpm      4.2 MB/s | 1.2
MB      00:00
:
(16/16): clang-libs-11.0.1-1.module+e18.4.0+12483+89b287b0.s390x.r 9.9 MB/s | 20
MB      00:02
-----
Total                                          14 MB/s | 56
MB      00:03
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      :
1/1
  Installing     : edb-as14-server-libs-14.4.0-1.rhel8.s390x
1/44
:
:
Installed:
edb-as14-pgagent-4.2.2-2.rhel8.s390x
edb-as14-server-14.4.0-1.rhel8.s390x
edb-as14-server-client-14.4.0-1.rhel8.s390x
:
:
llvm-11.0.0-2.module+e18.4.0+8598+a071fcd5.s390x
Complete!
[root@rdbk1101 ~]#

```

4.5.3 Customizing and starting EDB Postgres Server

Now that we have successfully installed EDB Postgres server on both primary and standby nodes, let's do some environmental customization and then start the DB server. During the installation of the EDB postgres, automatically it creates an 'enterprisedb' user-id and installs

the binaries in '/usr/edb/as14' folder. When we switch user (using 'su' command), the default bash home is "/var/lib/edb".

Example 4-6 Assign password for enterprisedb user-id

```
[root@rdbk1101 ~]# passwd enterprisedb
Changing password for user enterprisedb.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

At this point, we would switch to enterprisedb user-id and set the PGHOME path and INITDB environmental parameters in .bash_profile file. This would enable us to automatically apply the default path and INITDB parameters.

Example 4-7 Setting up environmental parameters.

```
[root@rdbk1101 ~]# su -l enterprisedb
Last login: Sun Aug 11 11:20:55 EDT 2024 on pts/0
[enterprisedb@rdbk1101 ~]$ pwd
/var/lib/edb
[enterprisedb@rdbk1101 ~]$ cat .bash_profile
[ -f /etc/profile ] && source /etc/profile
PGDATA=/var/lib/edb/as14/data
export PGDATA
# If you want to customize your settings,
# Use the file below. This is not overridden
# by the RPMS.
[ -f /var/lib/edb/.enterprisedb_profile ] && source
/var/lib/edb/.enterprisedb_profile
export PGHOME=/usr/edb/as14
PATH=$PATH:$PGHOME/bin
```

Make sure the PGHOME path is applied to the bash shell. From the root user-id initialize the database using the 'initdb' command with the CHAR types.

Example 4-8 Initializing the EDB Postgres.

```
[root@rdbk1101 ~]# PGSETUP_INITDB_OPTIONS="-E UTF-8"
/usr/edb/as14/bin/edb-as-14-setup initdb
Initializing database ... OK
[root@rdbk1101 ~]#
```

Once the database is initialized, now it's time to start the EDB Postgres server and initialize with the database.

Example 4-9 Starting EDB Postgres Server.

```
[root@rdbk1101 ~]# systemctl start edb-as-14.service
[root@rdbk1101 ~]# systemctl status edb-as-14.service
? edb-as-14.service - EDB Postgres Advanced Server 14
   Loaded: loaded (/usr/lib/systemd/system/edb-as-14.service; disabled; vendor
   preset: disabled)
   Active: active (running) since Mon 2024-08-12 07:26:13 EDT; 19s ago
     Process: 2922618 ExecStartPre=/usr/edb/as14/bin/edb-as-14-check-db-dir ${PGDATA}
   (code=exited, status=0/>)
    Main PID: 2922624 (edb-postmaster)
```

```

Tasks: 9 (limit: 204226)
Memory: 178.7M
CGroup: /system.slice/edb-as-14.service
  ..2922624 /usr/edb/as14/bin/edb-postmaster -D /var/lib/edb/as14/data
  ..2922625 postgres: logger
  ..2922627 postgres: checkpointer
  ..2922628 postgres: background writer
  ..2922629 postgres: walwriter
  ..2922630 postgres: autovacuum launcher
  ..2922631 postgres: stats collector
  ..2922632 postgres: dbms_aq launcher
  ..2922633 postgres: logical replication launcher
Aug 12 07:26:13 rdbk1101.cpolab.ibm.com systemd[1]: Starting EDB Postgres Advanced
Server 14...
Aug 12 07:26:13 rdbk1101.cpolab.ibm.com edb-postmaster[2922624]: 2024-08-12
07:26:13 EDT LOG:  redirecting>
Aug 12 07:26:13 rdbk1101.cpolab.ibm.com edb-postmaster[2922624]: 2024-08-12
07:26:13 EDT HINT:  Future log>
Aug 12 07:26:13 rdbk1101.cpolab.ibm.com systemd[1]: Started EDB Postgres Advanced
Server 14.
[root@rdbk1101 ~]#

```

Now that the EDB Postgres server is successfully started, let's verify that the database server process is started. For this we would need to switch user from 'root' to 'enterisedb' and issue 'pg_ctl' command.

Example 4-10 Verify database process using pg_ctl command

```

[root@rdbk1101 ~]# su - enterisedb
Last login: Mon Aug 12 06:58:52 EDT 2024 on pts/0
[enterisedb@rdbk1101 ~]$ pg_ctl status
pg_ctl: server is running (PID: 2922624)
/usr/edb/as14/bin/edb-postgres "-D" "/var/lib/edb/as14/data"

```

Now we know that database process is running, let's login in to the database shell and list down the databases that's available.

Example 4-11 Issue sql statements on the new EDB Postgres Server.

```

[enterisedb@rdbk1101 ~]$ /usr/edb/as14/bin/psql -d edb -U enterisedb
psql (14.4.0, server 14.4.0)
Type "help" for help.
edb=# \l

```

List of databases						
Name	Owner	Encoding	Collate	Ctype	ICU	
Access privileges						
edb	enterisedb	UTF8	en_US.UTF-8	en_US.UTF-8		
postgres	enterisedb	UTF8	en_US.UTF-8	en_US.UTF-8		
template0	enterisedb	UTF8	en_US.UTF-8	en_US.UTF-8		
=c/enterisedb		+				
enterisedb=CTc/enterisedb						

```

template1 | enterprisedb | UTF8      | en_US.UTF-8 | en_US.UTF-8 |
=c/enterprisedb          +
|                         |             |             |             |
enterprisedb=CTc/enterprisedb
(4 rows)
edb=# \q
[enterprisedb@rdbk1101 ~]$

```

Important: Till now, all the above steps are applicable to both the Postgres primary and secondary nodes. From here on, the steps would be differing, as would deploy Enterprise Failover manager for replication streaming purposes.

4.5.4 EDB Streaming Replication between primary and standby node

Configuration on the primary node

The first step is to create a replication user-id as part of the primary node, this user-id would be used by the standby node to connectivity purposes.

Example 4-12 Creating a user for replication in primary node

```

[enterprisedb@rdbk1101 ~]$ /usr/edb/as14/bin/psql -d edb -U
enterprisedb
psql (14.4.0, server 14.4.0)
Type "help" for help.
edb=# CREATE USER repl WITH REPLICATION ENCRYPTED PASSWORD 'repl';
CREATE ROLE
edb=# \password enterprisedb
Enter new password for user "enterprisedb":
Enter it again:
edb=#

```

Next, we would change the connectivity mode, in the `pg_hba.conf` file located under the `'/var/lib/edb/as14/data'` folder. For the easiness of the setup, the authentication method is set as trust. but in the real production setup it should be set to some more secure authentication method.

Example 4-13 Change the connectivity method in pg_hba.conf

```

[enterprisedb@rdbk1101 ~]$ cat /var/lib/edb/as14/data/pg_hba.conf
# PostgreSQL Client Authentication Configuration File
# =====
:
:
# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all trust
# IPv4 local connections:
host all all 127.0.0.1/32 trust
# IPv6 local connections:
host all all ::1/128 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local replication all trust

```

host	replication	all	127.0.0.1/32	trust
host	replication	all	:::1/128	trust

4.5.5 Enterprise Failover Manager (EFM)

EDB Failover Manager (EFM) provides the high availability infrastructure for EDB Postgres. EFM monitors the members of a Postgres cluster, identifies and verifies database failures quickly and reliably, and, if needed, promotes a standby node to become the cluster master and issues alerts. EnterpriseDB has also developed enterprise-class tools for performance, scalability, high availability, and management. These tools enable database administrators to easily manage more and bigger Postgres database deployments.



1. MongoDB and LinuxONE

Data is everywhere, in addition to traditional System of Record transaction data, nowadays a lot of streaming data feed into the system whether it is from Facebook, Instagram or other social media. Also, there are billions of IoT interfaces generating data. In the last couple of years world has generated as much data as in the history of entire human race. Companies are looking to make the most out of the data by organizing and building applications to access the information. And they are looking for ability to change the applications quickly if not dynamically. Robust data analysis needed to gain insights and make informed decisions and scalability and sustainability are another important factor. Enterprises want to use data and AI to optimize their departments like:

- Ø Marketing: Optimize customer reach
- Ø Sales: Uncover client needs.
- Ø Operations: Automate business processes.
- Ø Finance: Accurately forecast performance
- Ø HR: Attract top talent.
- Ø IT: Optimize IT spending.

So, the data must be managed, and the related data are called database and the technology for managing the database is called DBMS. Two prominent DBMS technologies are:

- Ø RDBMS
 - o Supports Relational data model and have the following characteristics
 - § Maturity
 - § Reliability
 - § High availability

- § ACID compliance
- § Security
- § Skills availability
- o Db2, Oracle, PostgreSQL, SQL Server are examples
- ∅ NoSQL Database Systems are an alternative to the RDBMS
 - o They don't use a relational data model and they have “**not only SQL**” and other interfaces
 - o Large volumes of data which changes rapidly, and data can be Structured or Unstructured
 - o No predefined Schema
 - o Supports Agile / DevOps methodologies for quick delivery of applications in just weeks, not in months
 - o The types of NoSQL databases are
 - § **Key-value Stores** -- Store pairs of keys and values, as well as retrieve values when a key is known, **redis** is an example
 - o **Document** -- Extension of key-value stores, schema free to organize data and data is stored in JSON-like format, **MongoDB**, **CouchDB** are examples
 - o **Wide Column Stores** --- Two-dimensional key-value stores and supports very large number of dynamic columns, **cassandra** is an example
 - o **Graph DBMS** -- Represent data in graph structures as nodes and edges, which are relationships between nodes, used for social connections, security, fraud, **neo4j** is an example database

In this chapter we will discuss about one of the NoSQL database MongoDB and how that database fits very well and leverages LinuxONE architecture

1.1 MongoDB features on IBM LinuxONE

MongoDB is a document database designed for ease of application development and scaling. MongoDB stores data in BSON format (like JSON document structure format). This makes it easy to operate with dynamic and unstructured data. MongoDB is an open-source and cross-platform database System and first released in 2009, the current release is 7.0. MongoDB is great in handling large volumes of data and the word Mongo is derived from “Humongous”. One of the LinuxONE architecture's benefit is to efficiently handle large I/O transactions and MongoDB take advantages of that. In the coming sections we will discuss the following important features of MongoDB on LinuxONE.

- ∅ MongoDB architecture
- ∅ MongoDB high availability
- ∅ MongoDB scalability
- ∅ Why MongoDB on LinuxONE

Ø MongoDB as a Service

0.0.1 MongoDB architecture

MongoDB is good at handling both structured and unstructured data. The basic components of MongoDB are

- Ø MongoDB Data layer
 - o Documents
 - o Collections
 - o Databases

- Ø MongoDB Application layer
 - o Application
 - o Drivers

- Ø MongoDB core components
 - o Storage engines
 - o MongoDB Server
 - o mongo and mongosh
 - o mongos

MongoDB Data layer

In MongoDB deployment, data layer is where the data resides. It consists of one or many databases, and each database has a set of collections. Collections are conceptually like tables in an RDBMS, but they're schema-less. Each collection can have one or multiple documents, which are like rows in an RDBMS.

Documents

MongoDB is a document database and stores all the related data together in binary JSON (BSON) documents, offering a schema-less model that provides flexibility in terms of database design. JSON and BSON documents don't require a predefined schema and allows the data structure to change over time.

The following is an example of a MongoDB document:

```
{
  _id: ObjectId("15cf23abcdef11c23a111a5b"),
  string: 'First Record',
  first_name: 'Sam',
  last_name: 'Amsavelu',
  object: { a: 100, b: false },
```

```
    array: [ 1, 2, 3 ]  
  }
```

Collections

Collections are grouping of documents, like tables in RDBMS, but without a fixed schema. Each collection can have one or more documents

Databases

A MongoDB instance can have one or more databases and each database contains one or more collections. There are default databases, **Admin** and **Local** Databases, which are used for system and internal operations.

MongoDB Application layer

In MongoDB deployment, the application layer is responsible for communicating with the database. To ensure secure access to the data, requests are initiated from this tier.

Application

This is the software or program that interacts with the MongoDB database. The application sends queries to the database and receives and processes the results. The application can be any type of software, such as a web application, mobile app, or desktop application. In a modern web application, this would be your API. A back-end application built with Node.js (or any other programming language with a native driver) makes requests to the database and relays the information back to the clients.

Drivers

The driver is a software library that allows the application to interact with the MongoDB database. The driver provides an interface for the application to send queries to the database and receive and process the results. MongoDB supports the driver written in many programming languages, such as Java, Python, Node.js, or C#. Basically, the drivers are client libraries that offer interfaces and methods for applications to communicate with MongoDB databases. Drivers will handle the translation of documents between BSON objects and mapping application structures.

MongoDB core components

Storage engines

MongoDB supports multiple storage engines, including the default WiredTiger storage engine and the older MMAPv1 engine. WiredTiger offers better performance and scalability, while MMAPv1 is simpler and can be a good choice for small deployments.

MongoDB Server

MongoDB Server is responsible for maintaining, storing, and retrieving data from the database through several interfaces. `mongod` is the MongoDB daemon, also known as the server for MongoDB. The MongoDB server listens for connections from clients on port 27017 by default, and stores data in the `/data/db` default directory when you use `mongod`. Each **mongod** server instance is in charge of handling client requests, maintaining data storage,

and performing database operations. Several mongod instances work together to form a cluster in a typical MongoDB setup.

mongo and mongosh

mongo is the shell, it's the client, it's a javascript interface that you can use to interact with the MongoDB server (mongod). However, as of June 2020, it was superseded by the new Mongo Shell, called, **mongosh**. Compared to mongo shell mongosh has improved syntax highlighting, command history and logging. mongosh is used to interact or change the data from MongoDB.

mongos

mongos is a proxy that sits between the client application (mongo/mongosh) and a sharded database cluster, that is multiple mongod replica sets. The mongos proxy is a map that the mongo client can use to query or make changes in the cluster when the data is sharded. This in-between proxy is required because the MongoDB cluster doesn't know which shard(s) that data exists on, but the mongos proxy does. We will discuss more about the sharded databases in the coming section.

The following figure 5-1 provides a graphical view of MongoDB architecture

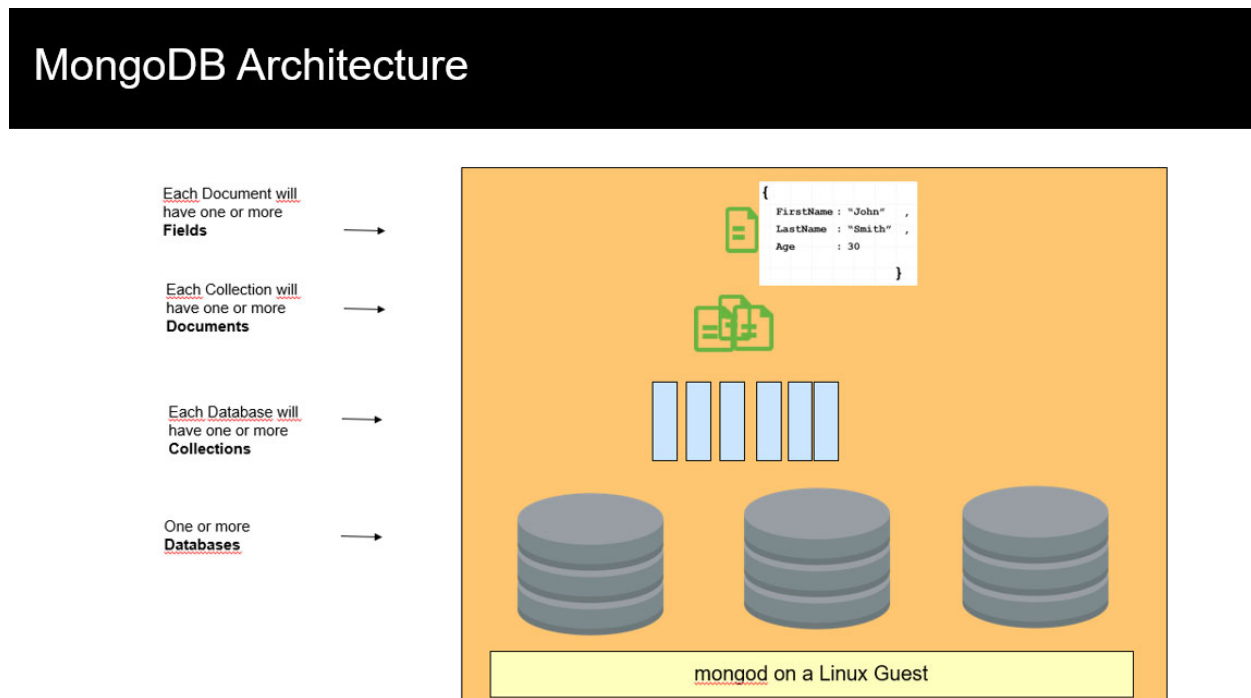


Figure 5-1 MongoDB architecture

0.0.1 MongoDB high availability

Enterprises have critical databases and applications and that require high availability. High availability databases are important to reduce the risk of losing revenue or dissatisfied customers. Highly available databases are designed to operate with no interruptions in service, even if there are hardware outages or network problems and those high available database systems are designed by the following core principles:

- Ø Eliminate single point of failures
- Ø Redundancy at various levels including data
- Ø Ability to detect and act on failures

MongoDB offers high availability through its replication feature, which involves the use of replica sets. A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability. In simple terms, MongoDB replication is the process of creating a copy of the same data set in more than one MongoDB server. MongoDB handles replication through a Replica Set, which consists of multiple MongoDB nodes that are grouped together as a unit. A replica set in MongoDB is a group of **mongod** processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments.

A MongoDB Replica Set requires a minimum of three MongoDB nodes:

- Ø One of the nodes will be considered the primary node that receives all the write operations.
- Ø The others are considered secondary nodes. These secondary nodes will replicate the data from the primary node.
- Ø The primary records all changes to its data sets in its operation log (**oplog**).
- Ø The secondaries replicate the primary's oplog and apply the operations to their data sets asynchronously such that the secondaries' data sets reflect the primary's data set.
- Ø If the primary is unavailable or when a primary does not communicate with the other members of the replica set for more than the configured period (ten seconds by default), an eligible secondary calls for an election to nominate itself as the new primary. The cluster attempts to complete the election of a new primary and resume normal operations.
- Ø The replica set cannot process write operations until the election completes successfully. The replica set can continue to serve read queries if such queries are configured to run on secondaries while the primary is offline.
- Ø Each replica set node must belong to one, and only one, replica set. Replica set nodes cannot belong to more than one replica set.
- Ø Replica-sets are platform independent

In relational databases like Db2, Oracle the data is shared across their instances for high availability however in MongoDB the data itself is replicated. This results in additional storage

and resource requirements. Also, as there is only one node which is primary, is taking all the write operations it usually reduces the throughput and scalability.

0.0.1 MongoDB scalability

Database systems with large data sets or high throughput applications can challenge the capacity of a single server. Very high query rates can exhaust the CPU capacity of the server and if the working set size is larger than the system's memory then it can stress the I/O and reduce the throughput. This problem can be addressed by either vertical scaling or horizontal scaling.

Vertical scaling

Vertical scaling involves increasing the hardware capabilities of the server, such as adding additional CPUs, increasing more RAM, and increasing the amount of storage space. Systems like IBM LinuxONE is a very good example to meet the vertical scalability requirements for applications by allowing to dynamically add CPU, memory and storage. Limitations in other platform technologies or cloud operations may restrict a single machine from being sufficiently powerful for a given workload.

Horizontal scaling

Horizontal Scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. Distributing the load reduces the strain on the required hardware resources, however horizontal scaling increases the complexity of underlying architecture.

MongoDB supports horizontal scaling through **sharding**.

MongoDB sharding

MongoDB sharding works by creating a cluster of MongoDB instances consisting of at least three servers. A MongoDB sharded cluster consists of the following components:

shard:

A **shard** is a single MongoDB instance that holds a subset of the sharded data. To increase availability and provide redundancy shards must be deployed as replica sets. The combination of multiple shards creates a complete data set. For example, a 2 TB data set can be broken down into four shards, each containing 500 GB of data from the original data set.

mongos:

The **mongos** act as the query router providing an interface between the application and the sharded cluster. This MongoDB instance is responsible for routing the client requests to the correct shard. mongos also support hedged reads to minimize latencies.

config servers:

Config servers store metadata and configuration settings for the whole sharded cluster. Config servers must be also deployed as a replica set.

Figure 5-2 shows a high-level architecture of MongoDB sharded cluster

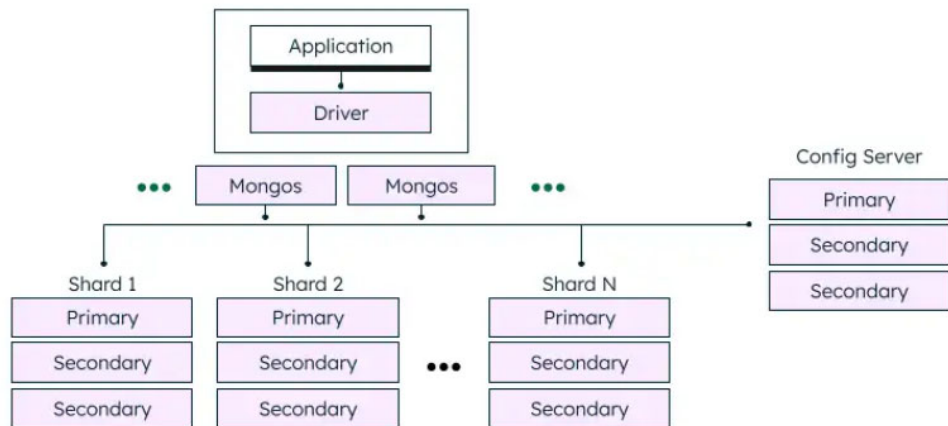


Figure 5-2 High level architecture of MongoDB sharded cluster

- ∅ The application communicates with the routers (**mongos**) about the query to be executed.
- ∅ The mongos instance consults the config servers to check which shard contains the required data set to send the query to that shard.
- ∅ Finally, the result of the query will be returned to the application.
- ∅ MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster.

Shard Keys:

MongoDB uses the shard key to distribute the collection's documents across shards. The shard key consists of a field or multiple fields in the documents. When sharding a MongoDB collection, a shard key gets created as one of the initial steps. The sharded key is immutable and cannot be changed after sharding. A sharded collection only contains a single shard key. The choice of shard key affects the performance, efficiency, and scalability of a sharded cluster.

MongoDB supports two sharding strategies **Hashed Sharding** and **Ranged Sharding** for distributing data across sharded clusters.

Sharding limitations:

- ∅ Sharding requires careful planning and maintenance
- ∅ The shard key directly impacts the overall performance of the underlying cluster
- ∅ The sharded key is immutable and cannot be changed after sharding. Once a collection has been sharded, MongoDB provides no method to unshard a sharded collection. While you can reshard your collection later, it is important to carefully consider your shard key choice to avoid scalability and performance issues
- ∅ Fragmentation, where a sharded collection's data is broken up into an unnecessarily large number of small chunks
- ∅ Also, there are some operational restrictions in sharded clusters like:
 - o The **geoSearch** command is not supported in sharded environments
 - o MongoDB does not support unique indexes across shards
 - o MongoDB does not guarantee consistent indexes across shards

For more detailed explanation about MongoDB high availability, sharding and other operations please refer to the MongoDB documentation.

0.1 Installing MongoDB on LinuxONE

In this section, we discuss the steps we took to install MongoDB Enterprise Server on LinuxONE in our lab environment. MongoDB Enterprise Server is the commercial edition of MongoDB, which includes additional capabilities such as in-memory storage engine for high throughput and low latency, advanced security features like LDAP and Kerberos access controls, and encryption for data at rest. MongoDB Enterprise Server software can be downloaded from MongoDB Download Center

MongoDB Enterprise Edition contains the following officially supported packages:

```
=====
Package
=====
Installing:
mongodb-enterprise
Installing dependencies:
mongodb-database-tools
mongodb-enterprise-cryptd
mongodb-enterprise-database
mongodb-enterprise-database-tools-extra
mongodb-enterprise-mongos
mongodb-enterprise-server
mongodb-enterprise-tools
mongodb-mongosh
```

mongodb-enterprise-7.0.14
mongodb-enterprise-database-7.0.14
mongodb-enterprise-server-7.0.14
mongodb-mongosh-7.0.14
mongodb-enterprise-mongos-7.0.14
mongodb-enterprise-tools-7.0.14

In our lab environment, we were running Red Hat Enterprise Linux release 8.4 guests

We downloaded the MongoDB Enterprise Server current software 7.0.14 for Red Hat s390x version server package from MongoDB Download Center

0.1.1 Install MongoDB Enterprise Edition

We followed these steps in our lab environment to install MongoDB Enterprise Edition using the yum package manager.

Configure the repository

We created an `/etc/yum.repos.d/mongodb-enterprise-7.0.repo` file with the following entries so that MongoDB Enterprise can be installed directly using yum:

```
[mongodb-enterprise-7.0]
name=MongoDB Enterprise Repository
baseurl=https://repo.mongodb.com/yum/redhat/8/mongodb-enterprise/7.0/$basearch/
gpgcheck=1
enabled=1
gpgkey=https://pgp.mongodb.com/server-7.0.asc
```

The MongoDB Enterprise repository contains the following officially supported packages:

mongodb-enterprise

A metapackage that automatically installs the component packages listed below:

- Ø **mongodb-enterprise-database**, a metapackage that automatically installs the component packages listed below:
 - o **mongodb-enterprise-server**
 - § Contains the mongod daemon and associated configuration and init scripts.
 - o **mongodb-enterprise-mongos**
 - § Contains the mongos daemon.
 - o **mongodb-enterprise-cryptd**
 - § Contains the mongocryptd binary
- Ø **mongodb-mongosh**
 - § Contains the MongoDB Shell (mongosh).
- Ø **mongodb-enterprise-tools**, a metapackage that automatically installs the component packages listed below:
 - o **mongodb-database-tools**, Contains the following MongoDB database tools:
 - § mongodump
 - § mongorestore
 - § bsondump
 - § mongoimport
 - § mongoexport
 - § mongostat
 - § mongotop
 - § mongofiles
 - o **mongodb-enterprise-database-tools-extra**, contains the following MongoDB support tools:
 - § mongoldap
 - § mongokerberos
 - § install_compass script
 - § mongodecrypt binary

Install the MongoDB Enterprise packages

Install the MongoDB Enterprise packages using the following command

```
# sudo yum install mongodb-enterprise
```

Sample output from the above command:

```
MongoDB Enterprise Repository                279 kB/s | 33 kB    00:00
Dependencies resolved.
=====
Package                                     Arch   Version           Repository         Size
=====
Installing:
mongodb-enterprise                          s390x  7.0.14-1.el8     mongodb-enterprise-7.0  9.5 k
Installing dependencies:
mongodb-database-tools                     s390x  100.10.0-1       mongodb-enterprise-7.0  45 M
mongodb-enterprise-cryptd                  s390x  7.0.14-1.el8     mongodb-enterprise-7.0  23 M
mongodb-enterprise-database                 s390x  7.0.14-1.el8     mongodb-enterprise-7.0  9.7 k
mongodb-enterprise-database-tools-extra    s390x  7.0.14-1.el8     mongodb-enterprise-7.0  20 M
mongodb-enterprise-mongos                   s390x  7.0.14-1.el8     mongodb-enterprise-7.0  23 M
mongodb-enterprise-server                   s390x  7.0.14-1.el8     mongodb-enterprise-7.0  34 M
mongodb-enterprise-tools                    s390x  7.0.14-1.el8     mongodb-enterprise-7.0  9.4 k
mongodb-mongosh                             s390x  2.3.1-1.el8      mongodb-enterprise-7.0  55 M
=====
Transaction Summary
=====
Install 9 Packages

Total download size: 200 M
Installed size: 878 M
Downloading Packages:
(1/9): mongodb-enterprise-7.0.14-1.el8.s390x.rpm           166 kB/s | 9.5 kB    00:00
(2/9): mongodb-enterprise-database-7.0.14-1.el8.s390x.rpm 341 kB/s | 9.7 kB    00:00
(3/9): mongodb-enterprise-database-tools-extra-7.0.14-1.el8.s390x.r 30 MB/s | 20 MB     00:00
(4/9): mongodb-database-tools-100.10.0-1.s390x.rpm        39 MB/s | 45 MB     00:01
(5/9): mongodb-enterprise-mongos-7.0.14-1.el8.s390x.rpm   27 MB/s | 23 MB     00:00
(6/9): mongodb-enterprise-cryptd-7.0.14-1.el8.s390x.rpm  13 MB/s | 23 MB     00:01
(7/9): mongodb-enterprise-tools-7.0.14-1.el8.s390x.rpm    64 kB/s | 9.4 kB    00:00
(8/9): mongodb-enterprise-server-7.0.14-1.el8.s390x.rpm   30 MB/s | 34 MB     00:01
(9/9): mongodb-mongosh-2.3.1.s390x.rpm                    46 MB/s | 55 MB     00:01
=====
Total                                                    67 MB/s | 200 MB    00:02
MongoDB Enterprise Repository                          16 kB/s | 1.6 kB    00:00
Importing GPG key 0x1785BA38:
  Userid      : "MongoDB 7.0 Release Signing Key <packaging@mongodb.com>"
  Fingerprint: E588 3020 1F7D D82C D808 AA84 160D 26BB 1785 BA38
  From        : https://pqp.mongodb.com/server-7.0.asc
```

```

Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      :                               1/1
  Installing     : mongodb-enterprise-database-tools-extra-7.0.14-1.el8.s390x 1/9
  Installing     : mongodb-mongosh-2.3.1-1.el8.s390x                2/9
  Running scriptlet: mongodb-enterprise-server-7.0.14-1.el8.s390x    3/9
  Installing     : mongodb-enterprise-server-7.0.14-1.el8.s390x    3/9
  Running scriptlet: mongodb-enterprise-server-7.0.14-1.el8.s390x    3/9
Created symlink /etc/systemd/system/multi-user.target.wants/mongod.service → /usr/lib/systemd/system/mongod.service.

  Installing     : mongodb-enterprise-mongos-7.0.14-1.el8.s390x     4/9
  Installing     : mongodb-enterprise-cryptd-7.0.14-1.el8.s390x     5/9
  Installing     : mongodb-enterprise-database-7.0.14-1.el8.s390x   6/9
  Running scriptlet: mongodb-database-tools-100.10.0-1.s390x       7/9
  Installing     : mongodb-database-tools-100.10.0-1.s390x       7/9
  Running scriptlet: mongodb-database-tools-100.10.0-1.s390x       7/9
  Installing     : mongodb-enterprise-tools-7.0.14-1.el8.s390x     8/9
  Installing     : mongodb-enterprise-7.0.14-1.el8.s390x          9/9
  Running scriptlet: mongodb-enterprise-7.0.14-1.el8.s390x        9/9
  Verifying     : mongodb-database-tools-100.10.0-1.s390x       1/9
  Verifying     : mongodb-enterprise-7.0.14-1.el8.s390x         2/9
  Verifying     : mongodb-enterprise-cryptd-7.0.14-1.el8.s390x   3/9
  Verifying     : mongodb-enterprise-database-7.0.14-1.el8.s390x  4/9
  Verifying     : mongodb-enterprise-database-tools-extra-7.0.14-1.el8.s390x 5/9
  Verifying     : mongodb-enterprise-mongos-7.0.14-1.el8.s390x   6/9
  Verifying     : mongodb-enterprise-server-7.0.14-1.el8.s390x   7/9
  Verifying     : mongodb-enterprise-tools-7.0.14-1.el8.s390x   8/9
  Verifying     : mongodb-mongosh-2.3.1-1.el8.s390x             9/9
Installed products updated.

Installed:
mongodb-database-tools-100.10.0-1.s390x
mongodb-enterprise-7.0.14-1.el8.s390x
mongodb-enterprise-cryptd-7.0.14-1.el8.s390x
mongodb-enterprise-database-7.0.14-1.el8.s390x
mongodb-enterprise-database-tools-extra-7.0.14-1.el8.s390x
mongodb-enterprise-mongos-7.0.14-1.el8.s390x
mongodb-enterprise-server-7.0.14-1.el8.s390x
mongodb-enterprise-tools-7.0.14-1.el8.s390x
mongodb-mongosh-2.3.1-1.el8.s390x

Complete!

```

Now we have installed MongoDB enterprise packages and any required dependent modules have also been installed automatically.

Running MongoDB Enterprise Edition

Recommended ulimit Settings

MongoDB recommends the following **ulimit** settings for **mongod** and **mongos** deployments:

- Ø -f (file size): unlimited
- Ø -t (cpu time): unlimited
- Ø -v (virtual memory): unlimited
- Ø -l (locked-in-memory size): unlimited
- Ø -n (open files): 64000
- Ø -m (memory size): unlimited
- Ø -u (processes/threads): 64000

Directory Paths

The package manager creates the default directories for data and log during installation. The owner and group name are mongod. By default, MongoDB runs using the mongod user account and uses the following default directories for data and log.

`/var/lib/mongo` (the data directory)

`/var/log/mongodb` (the log directory)

If you want to use data directory and/or log directory other than the default directories, create them and change the ownership of the directories to mongod. And edit the configuration file `/etc/mongod.conf` and modify the following fields accordingly:

storage.dbPath to specify a new data directory path

systemLog.path to specify a new log file path

Start MongoDB

MongoDB, mongod process can be started by issuing the following command:

```
# sudo systemctl start mongod
```

Verify MongoDB started successfully

You can verify that MongoDB has started successfully by issuing the following command:

```
# sudo systemctl status mongod
```

Sample output from the above commands:

```
[root@citis2 ~]# sudo systemctl start mongod
[root@citis2 ~]# sudo systemctl status mongod
● mongod.service - MongoDB Database Server
   Loaded: loaded (/usr/lib/systemd/system/mongod.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2024-09-11 12:59:14 EDT; 16s ago
     Docs: https://docs.mongodb.org/manual
   Main PID: 3545503 (mongod)
    Memory: 83.6M
    CGroup: /system.slice/mongod.service
            └─3545503 /usr/bin/mongod -f /etc/mongod.conf

Sep 11 12:59:14 citis2.wsclab.washington.ibm.com systemd[1]: Started MongoDB Database Server.
Sep 11 12:59:14 citis2.wsclab.washington.ibm.com mongod[3545503]: {"t":{"$date":"2024-09-11T16:59:14.166Z"}}
[root@citis2 ~]# █
```

Check the MongoDB version

You can check the MongoDB version by issuing the following command:

```
# mongod --version
```

Begin using MongoDB

You can start a mongosh session, the command line utility on the same host machine where the mongod is running.

You can run mongosh without any command-line options to connect to a mongod that is running on your localhost with default port 27017

Enter the following command:

```
# mongosh
```

Sample output from the above command:

```
[root@citis2 ~]# mongod --version
db version v7.0.14
Build Info: {
  "version": "7.0.14",
  "gitVersion": "ce59cfc6a3c5e5c067dca0d30697edd68d4f5188",
  "opensslVersion": "OpenSSL 1.1.1k FIPS 25 Mar 2021",
  "modules": [
    "enterprise"
  ],
  "allocator": "tcmalloc",
  "environment": {
    "distmod": "rhel83",
    "distarch": "s390x",
    "target_arch": "s390x"
  }
}
[root@citis2 ~]# mongosh
Current Mongosh Log ID: 66e1ce019678612898b44462
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.3.1
Using MongoDB:      7.0.14
Using Mongosh:      2.3.1

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2024-09-11T12:59:15.222-04:00: Access control is not enabled for the database. Read and write access to
o data and configuration is unrestricted
2024-09-11T12:59:15.222-04:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest set
ting it to 'never' in this binary version
2024-09-11T12:59:15.222-04:00: vm.max_map_count is too low
-----

Enterprise test> █
```

Type exit to come out of the mongosh shell

Connect to MongoDB from remote

By default, MongoDB launches with `bindIp` set to `127.0.0.1`, which binds to the localhost network interface. This means that the `mongod` can only accept connections from clients that are running on the same machine. Remote clients will not be able to connect to the `mongod`, and the `mongod` will not be able to initialize a replica set unless this value is set to a valid network interface.

This value can be configured in the MongoDB configuration file `/etc/mongod.conf` with `bindIp` value to `0.0.0.0`

Stop MongoDB

You can stop the `mongod` process by issuing the following command:

```
# sudo systemctl stop mongod
```

Restart MongoDB

You can restart the mongod process by issuing the following command:

```
# sudo systemctl restart mongod
```

0.1.2 Why MongoDB on LinuxONE

The IBM LinuxONE runs at the highest utilization level for maximum efficiency which enables businesses to be scalable and sustainable. Also, the lower TCO of LinuxONE The most significant contributor to a lower TCO for the LinuxONE is lower energy consumption and fewer cores needed per workload.

- Ø To run large amount of MongoDB instances at scale in a secure environment.
- Ø To support the required quality of service (QoS)
- Ø To be resilient and secure
- Ø For sustainability
- Ø Server consolidations
- Ø To achieve higher TCO
- Ø Dynamic scalability
- Ø Vertical scalability reduces the need for complex sharding in a MongoDB environment
- Ø Ability to deploy MongoDB as a Service
- Ø Compression to reduce MongoDB backup dump times

0.1.3 MongoDB as a Service on LinuxONE

Traditionally the business applications are developed in silo environments where each application and its database are deployed on its own servers. Most of the times these servers are oversized for the CPU, memory and storage requirements resulting in lot of whitespaces. In addition, each server must stipulate their backup, recovery and DR solutions. Very quickly the number of servers in a datacenter for an enterprise increases exponentially to meet the ever changing requirements from the business owners and their applications. These result in poor administration, less reliability, decreased productivity, increased cost of operations and longer time for development and deployments of new applications. DevOps are the new way


of developing applications efficiently and it mandates simplicity, flexibility, scalability and self-serviceability. The DevOps model of database management, known as Database-as-a-Service, helps break down these silos by creating a streamlined and efficient agile model.

In general, Database as a Service (DBaaS) can be defined as a managed service that gives users the ability to use a database without the complexities of hardware setup, software installation, or intricate configuration. Also, it offloads the responsibility of routine maintenance such as upgrades, backups, and security to the service provider, ensuring the database is operational around the clock.

Customers implement MongoDB Database-as-a-Service in private, public as well as hybrid cloud environments.

Customers choose to run MongoDB on-premises in a private cloud environment implementing as a DBaaS model for security, availability and reliability. LinuxONE is an excellent platform to deploy MongoDB as a service in an on-premises environment and has the following characteristics

- Ø self-service/on-demand database consumption coupled with automation of operations
- Ø scalability
- Ø availability
- Ø security
- Ø simplicity
- Ø flexibility
- Ø monitoring tools
- Ø sustainability
- Ø virtualization
- Ø cataloging
- Ø energy and floor space savings

Please refer to the Chapter 6: MongoDB as a service with Linux on IBM Z in the Redbook "Leveraging LinuxONE to Maximize Your Data Serving Capabilities" for a description of how IBM LinuxONE, combined with IBM Storage, provides high availability (HA), performance, and security by using a sample-anonymized client environment and includes a use case that demonstrates setting up a database as a service that can be replicated on a much larger scale across multiple client sites. 

MongoDB and LinuxONE

Data is everywhere, in addition to traditional System of Record transaction data, nowadays a lot of streaming data feed into the system whether it is from Facebook, Instagram or other social media. Also, there are billions of Internet of Things (IoT) interfaces that generate data. In the last couple of years, the world has generated as much data as in the history of the entire human race. Companies are looking to make the most out of the data by organizing

and building applications to access the information. And they are looking for the ability to change the applications quickly, if not dynamically.

Robust data analysis needed to gain insights and make informed decisions. Scalability and sustainability are other important factors when performing robust data analysis. Enterprises want to use data and AI to optimize their departments, such as:

- ▶ Marketing: Optimize customer reach
- ▶ Sales: Uncover client needs.
- ▶ Operations: Automate business processes.
- ▶ Finance: Accurately forecast performance
- ▶ HR: Attract top talent.
- ▶ IT: Optimize IT spending.

So, the data must be managed. Organized collections of data arranged for ease and speed of search and retrieval are called databases and the technology for managing databases is called a database management system or DBMS. Two prominent DBMS technologies are:

- ▶ Relational database management systems (RDBMS):
 - Supports the relational data model and have the following characteristics
 - Maturity
 - Reliability
 - High availability
 - Atomicity, consistency, isolation, and durability (ACID) compliance
 - Security
 - Skills availability
 - Db2, Oracle, PostgreSQL and SQL Server are examples of RDBMS.
- ▶ NoSQL, an alternative approach to an RDBMS that focuses on providing a way for storage and retrieval of data. NoSQL database designs are an alternative to the RDBMS for the following reasons:
 - They do not use a relational (tabular) data model and they have NoSQL (**not only SQL**) and other interfaces.
 - Handles large volumes of data which changes rapidly, and data can be Structured or Unstructured.
 - There is no predefined Schema.
 - Supports Agile / DevOps methodologies for quick delivery of applications in just weeks, not in months.
 - The following are types of NoSQL databases:
 - Key-value Stores:** Store pairs of keys and values, as well as retrieve values when a key is known, **redis** is an example.
 - Document:** Extension of key-value stores, schema free to organize data and data is stored in JSON-like format, **MongoDB and CouchDB** are examples.
 - Wide Column Stores:** Two-dimensional key-value stores and supports very large number of dynamic columns, **cassandra** is an example.

Graph DBMS

Represent data in graph structures as nodes and edges, which are relationships between nodes, used for social connections, security, fraud, **neo4j** is an example database.

In this chapter we will discuss one of the NoSQL databases, MongoDB, and how that database can satisfy your data serving needs while leveraging the LinuxONE architecture.

5.1 MongoDB on IBM LinuxONE overview

MongoDB is a document database designed for ease of application development and scaling. MongoDB stores data in BSON format (similar to the JSON document structure format). This makes it easy to operate with dynamic and unstructured data. MongoDB is an open-source and cross-platform database and was first released in 2009. As of this publication, the current release is 7.0.

MongoDB is great in handling large volumes of data and the word Mongo is derived from “Humongous”. One of the LinuxONE architecture’s benefit is to efficiently handle large I/O transactions and MongoDB takes advantage of that.

The following are topics discussed in this section:

- ▶ “MongoDB architecture” on page 138
- ▶ “MongoDB high availability” on page 140
- ▶ “MongoDB scalability ” on page 141
- ▶ “Why MongoDB on LinuxONE ” on page 149
- ▶ “MongoDB as a Service on LinuxONE ” on page 149

5.1.1 MongoDB architecture

MongoDB is good at handling both structured and unstructured data. The following are the basic components of MongoDB:

- ▶ MongoDB data layer

In MongoDB deployment, data layer is where the data resides. It consists of one or many databases, and each database has a set of collections. Collections are conceptually like tables in an RDBMS, but they’re schema-less. Each collection can have one or multiple documents, which are like rows in an RDBMS.

- Documents

MongoDB is a document database and stores all the related data together in binary JSON (BSON) documents, offering a schema-less model that provides flexibility in terms of database design. JSON and BSON documents don’t require a predefined schema and allows the data structure to change over time.

An example of a MongoDB document is shown in Example 5-1.

Example 5-1 MongoDB document

```
{
  _id: ObjectId("15cf23abcdef11c23a111a5b"),
  string: 'First Record',
  first_name: 'Sam',
  last_name: 'Amsavelu',
  object: { a: 100, b: false },
```

```
    array: [ 1, 2, 3 ]
  }
```

- Collections

Collections are grouping of documents, like tables in RDBMS, but without a fixed schema. Each collection can have one or more documents.

- Databases

A MongoDB instance can have one or more databases and each database contains one or more collections. There are default databases, Admin and Local Databases, which are used for system and internal operations.

- ▶ MongoDB Application layer

In MongoDB deployment, the application layer is responsible for communicating with the database. To ensure secure access to the data, requests are initiated from this tier.

- Application

This is the software or program that interacts with the MongoDB database. The application sends queries to the database and receives and processes the results. The application can be any type of software, such as a web application, mobile app, or desktop application. In a modern web application, this would be your API. A back-end application built with Node.js (or any other programming language with a native driver) makes requests to the database and relays the information back to the clients.

- Drivers

The driver is a software library that allows the application to interact with the MongoDB database. The driver provides an interface for the application to send queries to the database and receive and process the results. MongoDB supports the driver written in many programming languages, such as Java, Python, Node.js, or C#. Basically, the drivers are client libraries that offer interfaces and methods for applications to communicate with MongoDB databases. Drivers will handle the translation of documents between BSON objects and mapping application structures.

- ▶ MongoDB core components

- Storage engines

MongoDB supports multiple storage engines, including the default WiredTiger storage engine and the older MMAPv1 engine. WiredTiger offers better performance and scalability, while MMAPv1 is simpler and can be a good choice for small deployments.

- MongoDB Server

MongoDB Server is responsible for maintaining, storing, and retrieving data from the database through several interfaces. `mongod` is the MongoDB daemon, also known as the server for MongoDB. The MongoDB server listens for connections from clients on port 27017 by default, and stores data in the `/data/db` default directory when you use `mongod`. Each **mongod** server instance is in charge of handling client requests, maintaining data storage, and performing database operations. Several `mongod` instances work together to form a cluster in a typical MongoDB setup.

- `mongo` and `mongosh`

mongo is the shell, it's the client, it's a javascript interface that you can use to interact with the MongoDB server (`mongod`). However, as of June 2020, it was superseded by the new Mongo Shell, called, **mongosh**. Compared to `mongo` shell `mongosh` has improved syntax highlighting, command history and logging. `mongosh` is used to interact or change the data from MongoDB.

- mongos

mongos is a proxy that sits between the client application (mongo/mongosh) and a sharded database cluster, that is multiple mongod replica sets. The mongos proxy is a map that the mongo client can use to query or make changes in the cluster when the data is sharded. This in-between proxy is required because the MongoDB cluster doesn't know which shard(s) that data exists on, but the mongos proxy does. We will discuss more about the sharded databases in the coming section.

Figure 5-1 provides an overview of the MongoDB architecture.

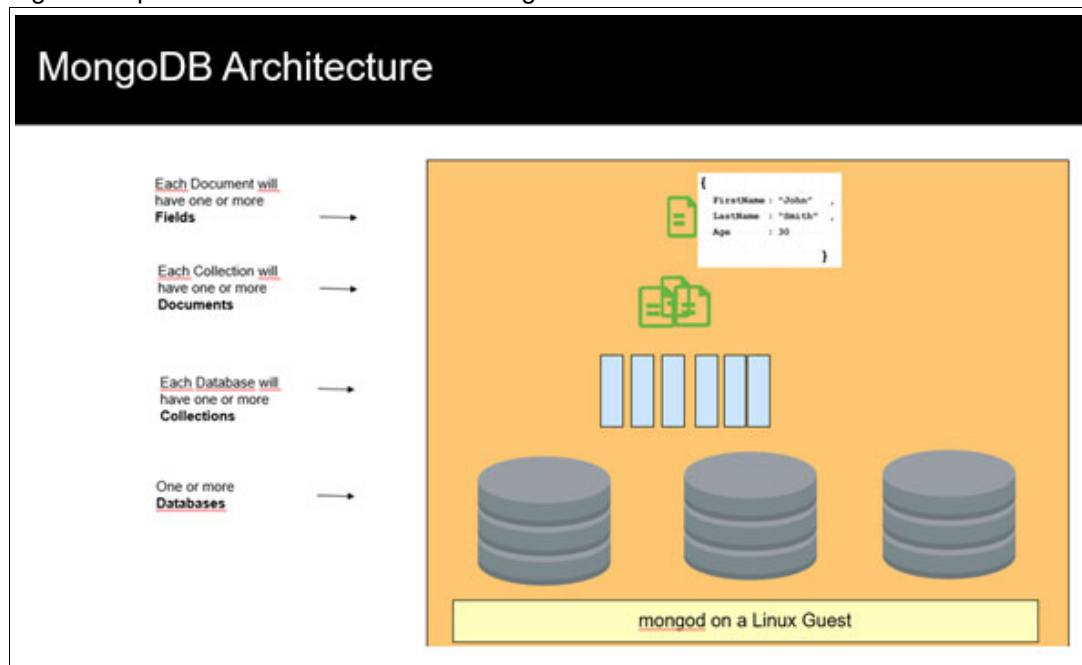


Figure 5-1 MongoDB architecture

5.1.2 MongoDB high availability

Enterprises have critical databases and applications and that require high availability. High availability databases are important to reduce the risk of dissatisfied customers or losing revenue. Highly available databases are designed to operate with no interruptions in service, even if there are hardware outages or network problems and those high availability database systems are designed with the following core principles:

- ▶ Eliminate single points of failure
- ▶ Redundancy at various levels, including data
- ▶ Ability to detect and act on failures

MongoDB offers high availability through its replication feature, which involves the use of replica sets. A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability. In simple terms, MongoDB replication is the process of creating a copy of the same data set in more than one MongoDB server. MongoDB handles replication through a Replica Set, which consists of multiple MongoDB nodes that are grouped together as a unit. A replica set in MongoDB is a group of **mongod** processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments.

A MongoDB Replica Set requires a minimum of three MongoDB nodes:

- ▶ One of the nodes will be considered the primary node that receives all the write operations.
- ▶ The others are considered secondary nodes. These secondary nodes will replicate the data from the primary node.
- ▶ The primary node records all changes to its data sets in its operation log (**oplog**).
- ▶ The secondary nodes replicate the primary node's oplog and apply the operations to their data sets asynchronously, such that the secondary nodes' data sets reflect the primary node's data set.
- ▶ If the primary node is unavailable or when a primary node does not communicate with the other members of the replica set for more than the configured period of time (ten seconds by default), an eligible secondary node calls for an election to nominate itself as the new primary node. The cluster attempts to complete the election of a new primary node and resume normal operations.
- ▶ The replica set cannot process write operations until the election completes successfully. The replica set can continue to serve read queries if such queries are configured to run on secondary nodes while the primary node is offline.
- ▶ Each replica set node must belong to one, and only one, replica set. Replica set nodes cannot belong to more than one replica set.
- ▶ Replica-sets are platform independent.

In relational databases such as Db2 or Oracle, the data is shared across their instances for high availability. However, in MongoDB, the data itself is replicated. This results in additional storage and resource requirements. Also, as there is only one node which is primary, if taking all the write operations, it usually reduces the throughput and scalability.

5.1.3 MongoDB scalability

Database systems with large data sets or high throughput applications can challenge the capacity of a single server. Very high query rates can exhaust the CPU capacity of the server and if the working set size is larger than the system's memory, then it can stress the I/O and reduce the throughput. This problem can be addressed by either vertical scaling or horizontal scaling.

Vertical scaling

Vertical scaling involves increasing the hardware capabilities of the server, such as adding additional CPUs, increasing more RAM, and increasing the amount of storage space. IBM LinuxONE provides a very good example to meet the vertical scalability requirements for applications by allowing dynamic additional CPU, memory and storage. Limitations in other platform technologies or cloud operations may restrict a single machine from being sufficiently powerful for a given workload.

Horizontal scaling

Horizontal scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. Distributing the load reduces the strain on the required hardware resources, however horizontal scaling increases the complexity of underlying architecture.

MongoDB supports horizontal scaling through **sharding**.

MongoDB sharding

MongoDB sharding works by creating a cluster of MongoDB instances consisting of at least three servers.

Figure 5-2 shows a high-level architecture of MongoDB sharded cluster.

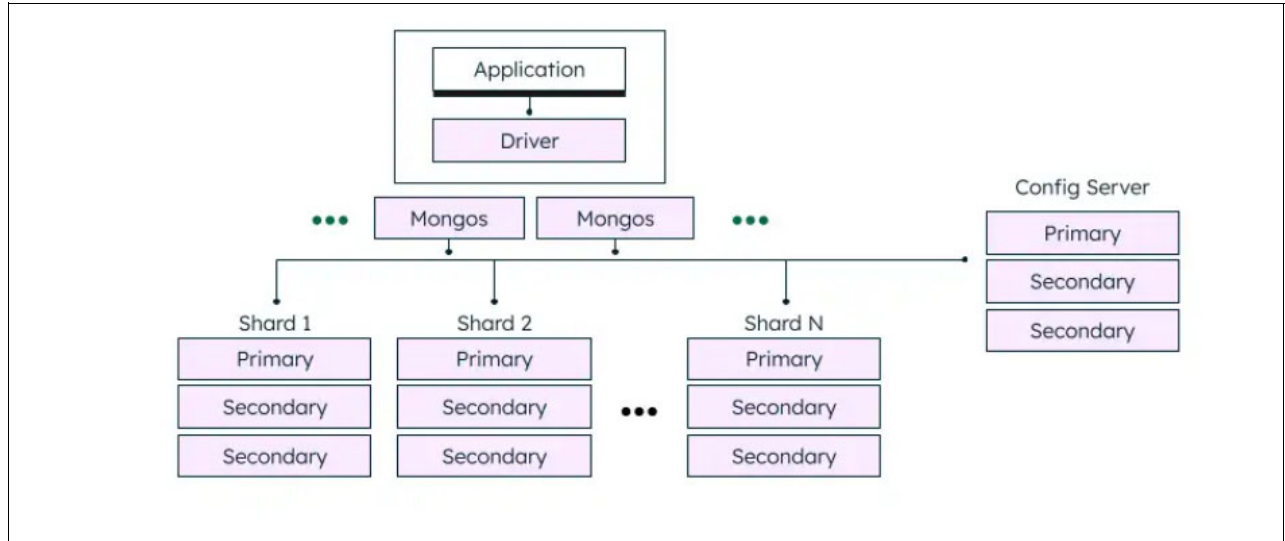


Figure 5-2 High level architecture of MongoDB sharded cluster

A MongoDB sharded cluster consists of the following components:

- shard** A **shard** is a single MongoDB instance that holds a subset of the sharded data. To increase availability and provide redundancy, shards must be deployed as replica sets. The combination of multiple shards creates a complete data set. For example, a 2 TB data set can be broken down into four shards, each containing 500 GB of data from the original data set.
- mongos** The **mongos** act as the query router providing an interface between the application and the sharded cluster. This MongoDB instance is responsible for routing the client requests to the correct shard. mongos also support hedged reads to minimize latencies.
- config servers** **Config servers** store metadata and configuration settings for the whole sharded cluster. Config servers must be also deployed as a replica set.
- Shard Keys** MongoDB uses the shard key to distribute the collection's documents across shards. The shard key consists of a field or multiple fields in the documents. When sharding a MongoDB collection, a shard key gets created as one of the initial steps. The sharded key is immutable and cannot be changed after sharding. A sharded collection only contains a single shard key. The choice of shard key affects the performance, efficiency, and scalability of a sharded cluster.

MongoDB supports two sharding strategies **Hashed Sharding** and **Ranged Sharding** for distributing data across sharded clusters.

The following shows the process of a MongoDB sharded cluster operation:

- ▶ The application communicates with the routers (**mongos**) about the query to be executed.

- ▶ The mongos instance consults the config servers to check which shard contains the required data set to send the query to that shard.
- ▶ Finally, the result of the query will be returned to the application.
- ▶ MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster.

Sharding limitations

The following are some limitations of sharding.

- ▶ Sharding requires careful planning and maintenance.
- ▶ The shard key directly impacts the overall performance of the underlying cluster.
- ▶ The sharded key is immutable and cannot be changed after sharding. Once a collection has been sharded, MongoDB provides no method to unshard a sharded collection. While you can reshard your collection later, it is important to carefully consider your shard key choice to avoid scalability and performance issues.
- ▶ Fragmentation, where a sharded collection's data is broken up into an unnecessarily large number of small chunks.
- ▶ Also, there are some operational restrictions in sharded clusters such as:
 - The **geoSearch** command is not supported in sharded environments
 - MongoDB does not support unique indexes across shards
 - MongoDB does not guarantee consistent indexes across shards

For more detailed explanation about MongoDB high availability, sharding and other operations please refer to the [MongoDB documentation](#).

5.2 Installing MongoDB on LinuxONE

In this section, we discuss the steps we took to install MongoDB Enterprise Server on LinuxONE in our lab environment. MongoDB Enterprise Server is the commercial edition of MongoDB, which includes additional capabilities such as an in-memory storage engine for high throughput and low latency, advanced security features such as LDAP and Kerberos access controls, and encryption for data at rest. MongoDB Enterprise Server software can be downloaded from the [MongoDB Download Center](#). In our lab environment, we were running Red Hat Enterprise Linux release 8.4 guests so we downloaded the MongoDB Enterprise Server software version 7.0.14 for Red Hat s390x server package.

MongoDB Enterprise Edition contains the officially supported packages shown in Figure 5-2.

Example 5-2 MongoDB Enterprise Edition packages

```

=====
Package
=====
Installing:
mongodb-enterprise
Installing dependencies:
mongodb-database-tools
mongodb-enterprise-cryptd
mongodb-enterprise-database
mongodb-enterprise-database-tools-extra
mongodb-enterprise-mongos

```

```
mongodb-enterprise-server
mongodb-enterprise-tools
mongodb-mongosh
```

```
mongodb-enterprise-7.0.14
mongodb-enterprise-database-7.0.14
mongodb-enterprise-server-7.0.14
mongodb-mongosh-7.0.14
mongodb-enterprise-mongos-7.0.14
mongodb-enterprise-tools-7.0.14
```

5.2.1 Install MongoDB Enterprise Edition

In this section, we outline the steps that we followed in our lab environment to install MongoDB Enterprise Edition using the YUM package manager.

1. Configure the repository

We created an `/etc/yum.repos.d/mongodb-enterprise-7.0.repo` file with the entries shown in Example 5-3 so that MongoDB Enterprise can be installed directly using YUM.

Example 5-3 Create the repo file

```
[mongodb-enterprise-7.0]
name=MongoDB Enterprise Repository
baseurl=https://repo.mongodb.com/yum/redhat/8/mongodb-enterprise/7.0/$basearch/
gpgcheck=1
enabled=1
gpgkey=https://pgp.mongodb.com/server-7.0.asc
```

The MongoDB Enterprise repository contains the `mongodb-enterprise` package which is a metapackage that automatically installs the following component packages:

- ▶ **mongodb-enterprise-database**, a metapackage that automatically installs the following component packages:
 - **mongodb-enterprise-server**: Contains the `mongod` daemon and associated configuration and init scripts.
 - **mongodb-enterprise-mongos**: Contains the `mongos` daemon.
 - **mongodb-enterprise-cryptd**: Contains the `mongocryptd` binary
 - **mongodb-mongosh**: Contains the MongoDB Shell (`mongosh`).
- ▶ **mongodb-enterprise-tools**, a metapackage that automatically installs the following component packages:
 - **mongodb-database-tools**, Contains the following MongoDB database tools:
 - `mongodump`
 - `mongorestore`
 - `bsondump`
 - `mongoimport`
 - `mongoexport`
 - `mongostat`
 - `mongotop`

- mongofiles
- **mongodb-enterprise-database-tools-extra**, contains the following MongoDB support tools:
 - mongoldap
 - mongokerberos
 - install_compass script
 - mongodecrypt binary

2. Install the MongoDB Enterprise packages by using the following command:

```
# sudo yum install mongodb-enterprise
```

Sample output from the above command is shown in Example 5-4.

Example 5-4 Output from install command

```
MongoDB Enterprise Repository                279 kB/s | 33 kB    00:00
Dependencies resolved.
=====
Package                                Arch    Version           Repository         Size
=====
Installing:
mongodb-enterprise                      s390x   7.0.14-1.el8     mongodb-enterprise-7.0  9.5 k
Installing dependencies:
mongodb-database-tools                  s390x   100.10.0-1       mongodb-enterprise-7.0  45 M
mongodb-enterprise-cryptd               s390x   7.0.14-1.el8     mongodb-enterprise-7.0  23 M
mongodb-enterprise-database              s390x   7.0.14-1.el8     mongodb-enterprise-7.0  9.7 k
mongodb-enterprise-database-tools-extra s390x   7.0.14-1.el8     mongodb-enterprise-7.0  20 M
mongodb-enterprise-mongos                s390x   7.0.14-1.el8     mongodb-enterprise-7.0  23 M
mongodb-enterprise-server                s390x   7.0.14-1.el8     mongodb-enterprise-7.0  34 M
mongodb-enterprise-tools                 s390x   7.0.14-1.el8     mongodb-enterprise-7.0  9.4 k
mongodb-mongosh                          s390x   2.3.1-1.el8      mongodb-enterprise-7.0  55 M
=====
Transaction Summary
=====
Install 9 Packages

Total download size: 200 M
Installed size: 878 M
Downloading Packages:
(1/9): mongodb-enterprise-7.0.14-1.el8.s390x.rpm                166 kB/s | 9.5 kB    00:00
(2/9): mongodb-enterprise-database-7.0.14-1.el8.s390x.rpm      341 kB/s | 9.7 kB    00:00
(3/9): mongodb-enterprise-database-tools-extra-7.0.14-1.el8.s390x.r  30 MB/s | 20 MB    00:00
(4/9): mongodb-database-tools-100.10.0-1.s390x.rpm             39 MB/s | 45 MB    00:01
(5/9): mongodb-enterprise-mongos-7.0.14-1.el8.s390x.rpm        27 MB/s | 23 MB    00:00
(6/9): mongodb-enterprise-cryptd-7.0.14-1.el8.s390x.rpm        13 MB/s | 23 MB    00:01
(7/9): mongodb-enterprise-tools-7.0.14-1.el8.s390x.rpm         64 kB/s | 9.4 kB    00:00
(8/9): mongodb-enterprise-server-7.0.14-1.el8.s390x.rpm        30 MB/s | 34 MB    00:01
(9/9): mongodb-mongosh-2.3.1.s390x.rpm                          46 MB/s | 55 MB    00:01
-----
Total                                                                67 MB/s | 200 MB    00:02
MongoDB Enterprise Repository                                       16 kB/s | 1.6 kB    00:00
Importing GPG key 0x1785BA38:
  Userid      : "MongoDB 7.0 Release Signing Key <packaging@mongodb.com>"
  Fingerprint: E588 3020 1F7D D82C D808 AA84 160D 26BB 1785 BA38
  From        : https://pqp.mongodb.com/server-7.0.asc
```

```

Key imported successfully
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      :                               1/1
  Installing     : mongodb-enterprise-database-tools-extra-7.0.14-1.el8.s390x 1/9
  Installing     : mongodb-mongosh-2.3.1-1.el8.s390x 2/9
  Running scriptlet: mongodb-enterprise-server-7.0.14-1.el8.s390x 3/9
  Installing     : mongodb-enterprise-server-7.0.14-1.el8.s390x 3/9
  Running scriptlet: mongodb-enterprise-server-7.0.14-1.el8.s390x 3/9
Created symlink /etc/systemd/system/multi-user.target.wants/mongod.service → /usr/lib/systemd/system/mongod.service.

  Installing     : mongodb-enterprise-mongos-7.0.14-1.el8.s390x 4/9
  Installing     : mongodb-enterprise-cryptd-7.0.14-1.el8.s390x 5/9
  Installing     : mongodb-enterprise-database-7.0.14-1.el8.s390x 6/9
  Running scriptlet: mongodb-database-tools-100.10.0-1.s390x 7/9
  Installing     : mongodb-database-tools-100.10.0-1.s390x 7/9
  Running scriptlet: mongodb-database-tools-100.10.0-1.s390x 7/9
  Installing     : mongodb-enterprise-tools-7.0.14-1.el8.s390x 8/9
  Installing     : mongodb-enterprise-7.0.14-1.el8.s390x 9/9
  Running scriptlet: mongodb-enterprise-7.0.14-1.el8.s390x 9/9
  Verifying     : mongodb-database-tools-100.10.0-1.s390x 1/9
  Verifying     : mongodb-enterprise-7.0.14-1.el8.s390x 2/9
  Verifying     : mongodb-enterprise-cryptd-7.0.14-1.el8.s390x 3/9
  Verifying     : mongodb-enterprise-database-7.0.14-1.el8.s390x 4/9
  Verifying     : mongodb-enterprise-database-tools-extra-7.0.14-1.el8.s390x 5/9
  Verifying     : mongodb-enterprise-mongos-7.0.14-1.el8.s390x 6/9
  Verifying     : mongodb-enterprise-server-7.0.14-1.el8.s390x 7/9
  Verifying     : mongodb-enterprise-tools-7.0.14-1.el8.s390x 8/9
  Verifying     : mongodb-mongosh-2.3.1-1.el8.s390x 9/9
Installed products updated.

Installed:
mongodb-database-tools-100.10.0-1.s390x
mongodb-enterprise-7.0.14-1.el8.s390x
mongodb-enterprise-cryptd-7.0.14-1.el8.s390x
mongodb-enterprise-database-7.0.14-1.el8.s390x
mongodb-enterprise-database-tools-extra-7.0.14-1.el8.s390x
mongodb-enterprise-mongos-7.0.14-1.el8.s390x
mongodb-enterprise-server-7.0.14-1.el8.s390x
mongodb-enterprise-tools-7.0.14-1.el8.s390x
mongodb-mongosh-2.3.1-1.el8.s390x

Complete!

```

The MongoDB Enterprise Server packages and any required dependent modules have now been installed automatically.

5.2.2 Running MongoDB Enterprise Edition

In this section, we outline recommended ulimit settings, directory paths that get created, and how to start MongoDB.

ulimit settings

MongoDB recommends the following **ulimit** settings for **mongod** and **mongos** deployments:

- ▶ f (file size): unlimited
- ▶ t (cpu time): unlimited
- ▶ v (virtual memory): unlimited
- ▶ l (locked-in-memory size): unlimited
- ▶ n (open files): 64000

- ▶ m (memory size): unlimited
- ▶ u (processes/threads): 64000

Directory paths

The package manager creates default directories for data and log during installation. The owner and group name are `mongod`. By default, MongoDB runs using the `mongod` user account and uses the following default directories for data and log.

- ▶ `/var/lib/mongo` (the data directory)
- ▶ `/var/log/mongodb` (the log directory)

If you want to use a data directory and/or a log directory other than the default directories, create them and change the ownership of the directories to `mongod`. Edit the configuration file `/etc/mongod.conf` and modify the following fields accordingly:

- ▶ `storage.dbPath` to specify a new data directory path
- ▶ `systemLog.path` to specify a new log file path

Start MongoDB

You can start MongoDB from a command line by issuing the following `mongod` command.
sudo systemctl start mongod

For more information on the `mongod` command as well as a list of options, see the following website:

<https://www.mongodb.com/docs/manual/reference/program/mongod/>

Verify MongoDB started successfully

You can verify that MongoDB has started successfully by issuing the following command:
sudo systemctl status mongod

Sample output from the `status` command is shown in Figure 5-3.

```
[root@citis2 ~]# sudo systemctl start mongod
[root@citis2 ~]# sudo systemctl status mongod
● mongod.service - MongoDB Database Server
   Loaded: loaded (/usr/lib/systemd/system/mongod.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2024-09-11 12:59:14 EDT; 16s ago
     Docs: https://docs.mongodb.org/manual
   Main PID: 3545503 (mongod)
    Memory: 83.6M
    CGroup: /system.slice/mongod.service
           └─3545503 /usr/bin/mongod -f /etc/mongod.conf

Sep 11 12:59:14 citis2.wsclab.washington.ibm.com systemd[1]: Started MongoDB Database Server.
Sep 11 12:59:14 citis2.wsclab.washington.ibm.com mongod[3545503]: {"t":{"$date":"2024-09-11T16:59:14.123Z"}}
[root@citis2 ~]#
```

Figure 5-3 Sample out put from status command

Check the MongoDB version

You can check the MongoDB version by issuing the following command:
mongod --version

Begin using MongoDB

You can start a `mongosh` session, the command line utility, on the same host machine where the `mongod` is running.

You can run `mongosh` without any command-line options to connect to a `mongod` that is running on your localhost with default port 27017 by using the following command:

mongosh

Sample output from the above command is shown in Figure 5-4.

```
[root@citis2 ~]# mongod --version
db version v7.0.14
Build info: {
  "version": "7.0.14",
  "gitVersion": "ce59cfc6a3c5e5c067dca0d30697edd68d4f5188",
  "opensslVersion": "OpenSSL 1.1.1k FIPS 25 Mar 2021",
  "modules": [
    "enterprise"
  ],
  "allocator": "tcmalloc",
  "environment": {
    "distmod": "rhel83",
    "distarch": "s390x",
    "target_arch": "s390x"
  }
}
[root@citis2 ~]# mongosh
Current Mongosh Log ID: 66e1ce019678612898b44462
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.3.1
Using MongoDB:      7.0.14
Using Mongosh:      2.3.1

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2024-09-11T12:59:15.222-04:00: Access control is not enabled for the database. Read and write access to
data and configuration is unrestricted
2024-09-11T12:59:15.222-04:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest set
ting it to 'never' in this binary version
2024-09-11T12:59:15.222-04:00: vm.max_map_count is too low
-----

Enterprise test> █
```

Figure 5-4 Sample output from `mongosh` command.

Type `exit` to leave the `mongosh` shell.

Connect to MongoDB from remote

By default, MongoDB launches with `bindIp` set to 127.0.0.1, which binds to the localhost network interface. This means that the `mongod` can only accept connections from clients that are running on the same machine. Remote clients will not be able to connect to the `mongod`, and the `mongod` will not be able to initialize a replica set unless this value is set to a valid network interface.

This value can be configured in the MongoDB configuration file `/etc/mongod.conf` with `bindIp` value to 0.0.0.0

Stop MongoDB

You can stop the `mongod` process by issuing the following command:

```
sudo systemctl stop mongod
```

Restart MongoDB

You can restart the `mongod` process by issuing the following command:

```
sudo systemctl restart mongod
```

5.2.3 Why MongoDB on LinuxONE

IBM LinuxONE is designed to run at the highest utilization level for maximum efficiency which enables businesses to be scalable and sustainable. Also, the lower TCO of LinuxONE The most significant contributor to a lower TCO for the LinuxONE is lower energy consumption and fewer cores needed per workload.

- ▶ To run large amount of MongoDB instances at scale in a secure environment.
- ▶ To support the required quality of service (QoS)
- ▶ To be resilient and secure
- ▶ For sustainability
- ▶ Server consolidations
- ▶ To achieve higher TCO
- ▶ Dynamic scalability
- ▶ Vertical scalability reduces the need for complex sharding in a MongoDB environment
- ▶ Ability to deploy MongoDB as a Service
- ▶ Compression to reduce MongoDB backup dump times

5.2.4 MongoDB as a Service on LinuxONE

Traditionally, business applications are developed in silo environments where each application and its database are deployed on their own servers. Most of the times these servers are oversized for the CPU, memory and storage requirements resulting in lot of whitespaces. In addition, each server must stipulate their backup, recovery and DR solutions. Very quickly the number of servers in a datacenter for an enterprise increases exponentially to meet the ever changing requirements from the business owners and their applications. These result in poor administration, less reliability, decreased productivity, increased cost of operations and longer time for development and deployments of new applications. DevOps is the new way of developing applications efficiently and it mandates simplicity, flexibility, scalability and self-serviceability. The DevOps model of database management, known as Database-as-a-Service, helps break down these silos by creating a streamlined and efficient agile model.

In general, Database as a Service (DBaaS) can be defined as a managed service that gives users the ability to use a database without the complexities of hardware setup, software installation, or intricate configuration. Also, it offloads the responsibility of routine maintenance such as upgrades, backups, and security to the service provider, ensuring the database is operational around the clock.


Customers implement MongoDB Database-as-a-Service in private, public as well as hybrid cloud environments.

Customers choose to run MongoDB on-premises in a private cloud environment implementing as a DBaaS model for security, availability and reliability. LinuxONE is an excellent platform to deploy MongoDB as a service in an on-premises environment and has the following characteristics:

- ▶ self-service/on-demand database consumption coupled with automation of operations
- ▶ scalability
- ▶ availability
- ▶ security

- ▶ simplicity
- ▶ flexibility
- ▶ monitoring tools
- ▶ sustainability
- ▶ virtualization
- ▶ cataloging
- ▶ energy and floor space savings

See Appendix B, “MongoDB as a service with IBM LinuxONE” on page 351, for a description of how IBM LinuxONE, combined with IBM Storage, provides high availability (HA), performance, and security by using a sample-anonymized client environment and includes a use case that demonstrates setting up a database as a service that can be replicated on a much larger scale across multiple client sites.



Open source data base management systems and LinuxONE

There are additional open source database management systems (DBMS) that can be migrated to the LinuxONE platform. In this chapter, you will learn about migration from relational databases such as MySQL to open-source databases such as PostgreSQL or MongoDB on LinuxONE. This migration will provide advantages such as scalability , agility, cost saving and security enhancements

- ▶ “Migration from MySQL to MongoDB” on page 152
- ▶ “Migration from MySQL to PostgreSQL” on page 152

6.1 Migration from MySQL to MongoDB

Users can migrate from MySQL to open source Database MongoDB by using the following simple steps.

1. First, you will need to prepare the MongoDB data schemes based on your own requirements and follow best practices to improve the data retrieval as MongoDB is a document database includes object and array fields. <<See the following website for some of the best practices to follow:
https://www.mongodb.com/docs/manual/data-modeling/?_ga=2.118818446.642180643.1721974765-1737037843.1721577255 >>.

For example : User contact across multiple tables can be mapped as a one-to-many relationship in a document.

2. Next, you will need to prepare for the crucial step where data can be migrated from an existing MySQL database to the new MongoDB. Use a tool such as MySQL workbench to convert the content to MongoDB readable format, such as comma separated values(CSV) or JavaScript Object Notation (JSON) files and export the resulting files into MongoDB.
3. Transform the data in such a way that the destination database, MongoDB will work in an efficient manner. For example: Single call to MongoDB can be grouped in a single document. To transform the data, use an ETL (Extract Transform Load) tool such as IBM Infosphere Datastage.
4. Finally, perform the last step of migration, importing the existing MySQL Data to the MongoDB instance. A command Line Interface(CLI) tool such as mongoimport or graphical user interface such as MongoDB Compass can be used to import the data.

6.2 Migration from MySQL to PostgreSQL

Users can migrate from MySQL to open source Database Redis by using the following simple steps.

1. First, install Pgloader utility on the system where PostgreSQL Database will be Installed. Refer to https://hevodata.com/learn/migrate-data-from-mysql-to-postgresql/#Step_2_Install_Pgloader_On_Your_System to install Pgloader.
2. Next, build a PostgreSQL Database and Role. For more information on this topic, see:https://hevodata.com/learn/migrate-data-from-mysql-to-postgresql/#Step_3_Build_a_PostgreSQL_Database_and_Role .
3. Next, make sure that MySQL to PostgreSQL communication is over secure channel through establishing SSL connection with certificates. For more information on this topic, see:https://hevodata.com/learn/migrate-data-from-mysql-to-postgresql/#Step_3_Build_a_PostgreSQL_Database_and_Role on how to establish the secure connection. Once the secure connection is established, Pgloader will migrate your data over secure channel.
4. Next, use the pgLoader utility to migrate data from MySQL to local or remote PostgreSQL database. Refer to https://hevodata.com/learn/migrate-data-from-mysql-to-postgresql/#Step_5_Migration_of_Data for more information.



Leveraging containers

With the rapid progress of digital technologies, the needs of the world are constantly changing. Organizations must transform businesses through system modernization to respond quickly and flexibly to this change. This transformation requires a break from the traditional system that takes several years to design and run for a long time. Rather, it is imperative to build and operate database systems quickly and flexibly. To achieve this goal, it is a best practice to use containers that automate deployment and operations.

7.1 Containerized databases

Using containerized business applications has become prominent in recent years. As organizations shift toward a microservices system to remodel their assets to data-driven, cloud-native enterprise systems, database systems are becoming more popular as the candidate for containerization.

However, there are differences between the nature of database management systems (DBMSs) and business applications that kept DBMSs from being containerized in the past. DBMSs are more CPU- and memory-intensive; they are stateful; and they require storage. The rapid progress in container and container management technology along with DBMS software's close integration with those technologies has realized containerized databases. DBMS server software is encapsulated into containers by separating the database engine from the database files storage, and persistent storage volumes are used. Container orchestration frameworks such as Kubernetes provide high-throughput, low-latency networking, built-in high availability (HA) capabilities, and support for stateful container management, which are essential for DBMS.

Building and maintaining DBMSs in containerized environments brings various benefits to organizations:

- ▶ Running business applications and their databases on a common platform lowers maintenance complexity, and also reduces networking issues between the application and databases compared to a system where the application and database are running on separate environments.
- ▶ With flexible scaling, containerized databases can cater better to application elasticity. Scaling flexibility also means that organizations can start small and scale up or out.
- ▶ With the simplicity of a whole system that contains the application and the database, deploying regional services can be done conveniently, which ensures that future reform of your systems can be done without substantial rebuilding and investment.

In short, containerized databases are an essential component of a data modernization platform.

This chapter describes the following concepts and use cases:

- ▶ “Benefits of automation when using containers” on page 155
- ▶ “Oracle containers” on page 156
- ▶ “Db2 containers” on page 156
- ▶ “FUJITSU Enterprise Postgres containers” on page 156
 - Automated deployment by using a standardized template file
 - Automated backup that runs by default, which can be customized easily
 - Autohealing to recover systems without human intervention in a failure
 - Monitoring for stable system operations and continuous system reform
 - Autoscaling for expanding system capacity according to workloads
 - Service expansion by using IBM LinuxONE
 - Quick deployment of new databases for business expansion
- ▶ “MongoDB and containers” on page 209
- ▶ “Open source databases and containers” on page 209

7.2 Benefits of automation when using containers

The needs of consumers are constantly changing, and to respond quickly and flexibly to this change, it is necessary to build and operate databases quickly and flexibly. In the past, it was costly to deploy and operate databases because of such things as designing the HA configuration of databases and responding to abnormal and unexpected fluctuations.

With automation that uses containers, the cost of deployment and operations can be reduced. Organizations can benefit from focusing investments on application development for new services, which empowers and expedites business reform through modernization.

7.2.1 Key qualities of modern database systems

Databases that support today's initiatives have three key qualities:

- ▶ **Agility**

Containerized databases enable accelerated database deployment and setup. Database systems can be deployed in a short period with simple operations. Databases are more responsive to changes.

- ▶ **Flexibility**

Scales according to workload and adapts to changing situations. In addition to the resiliency of container technology, the automation of database failover and recovery operations enables immediate recovery from events such as DB failures. Automated backup operations and declarative restores make it easy to recover from events such as data corruption.

- ▶ **Portability**

The portability of containers makes it possible to deploy databases on various platforms.

As shown in Figure 7-1, when modernizing data and services, organizations move through five elements. These five elements are deployment at the start of reform, operation and fluctuation during the continuation of the reform, success at the completion of the reform, and next steps. The successful outcomes of one reform project build the foundation for the next steps, and reform is delivered continuously.

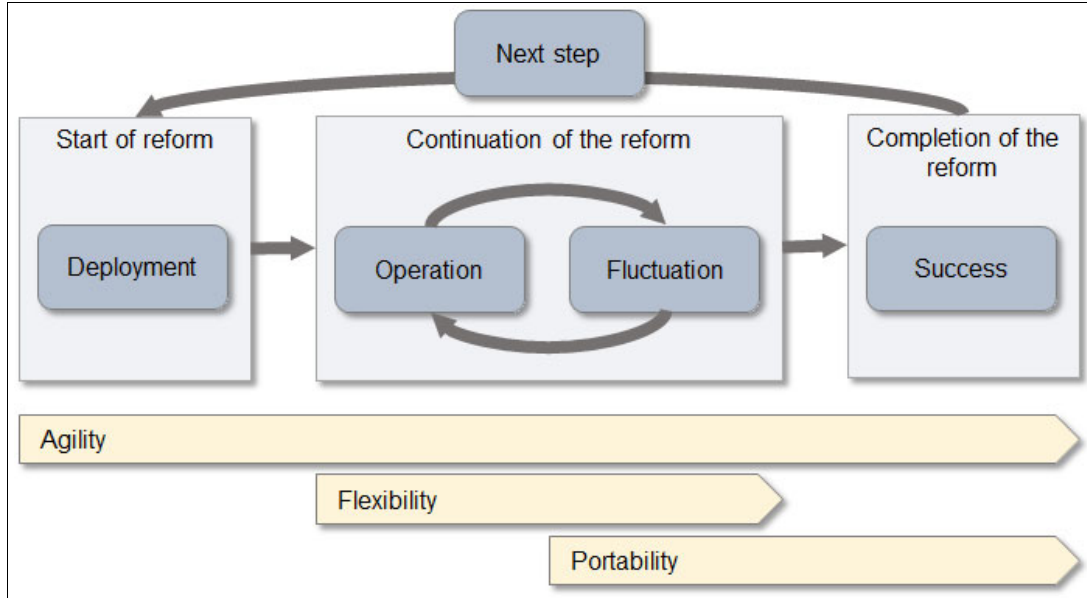


Figure 7-1 Customer journey through data and services modernization

The following sections describe quick deployment, low-cost database operations, maintaining optimal performance against fluctuation, and the next steps.

7.3 Oracle containers

7.4 Db2 containers

IBM Db2 Universal (Db2U) is a containerized version of IBM Db2 which has been designed for deployment in cloud and on-premises environments. Db2U is a reference to containerized deployment of Db2 and it is still the same Db2.

Db2U is packaged in containers, making it easy to deploy and manage in cloud-native environments. It leverages Kubernetes and Red Hat OpenShift for orchestration and scalability.

See [Modernizing Db2 Containerization Footprint with Db2U](#) for more information on the Db2U architecture.

7.5 FUJITSU Enterprise Postgres containers

FUJITSU Enterprise Postgres combines container technology with an open interface database and Fujitsu technology to realize these three qualities.

FUJITSU Enterprise Postgres Operator supports multi-architecture and can be deployed anywhere on LinuxONE and IBM cloud services.

Complex database server deployment and setup can be realized by using a GUI. By modifying the input template file in a declarative way, any configuration can be built in a few steps, making database deployment quick and easy.

This section describes how FUJITSU Enterprise Postgres Operator deploys databases quickly.

Sections 7.5, “FUJITSU Enterprise Postgres containers” on page 156 through 7.5.3, “Fluctuation” on page 190 assume that the configuration that is shown in Figure 7-2 on page 157 is used.

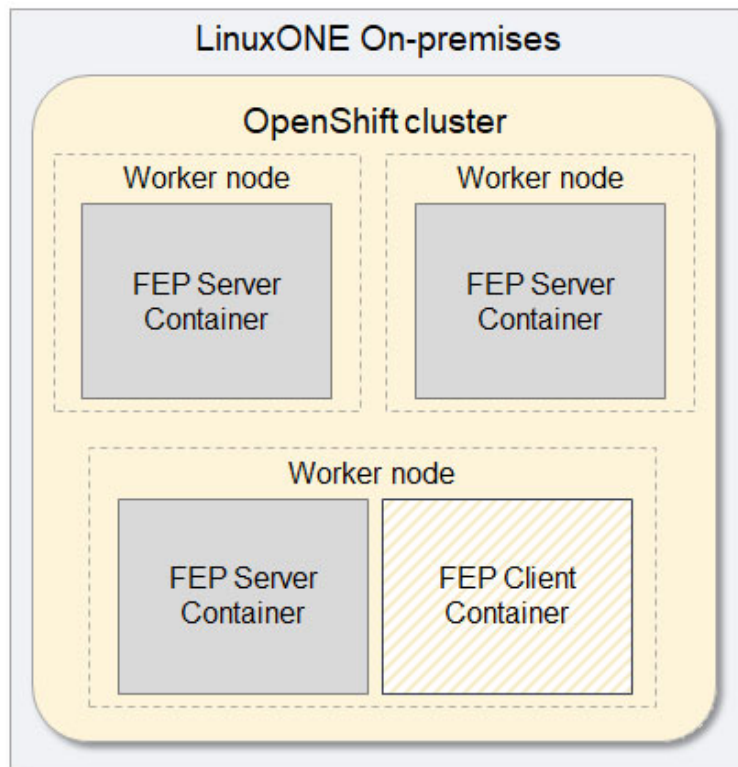


Figure 7-2 System deployment overview

Prerequisites

In this section, we present the prerequisites that are needed to deploy a FUJITSU Enterprise Postgres database on Red Hat OpenShift Container Platform (RHOCP).

The FUJITSU Enterprise Postgres Operator must be installed first. For more information, see 10.4.1 “FUJITSU Enterprise Postgres Operator installation”, in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

Our storage was on IBM Spectrum Virtualize and we used Gold as our storage class. For more information, see 10.2.17 “Installing IBM Spectrum Virtualize and setting up a storage class”, in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

Here are more details about the prerequisites:

- ▶ Fully qualified domain name (FQDN) for logical replication

Prepare the FQDN that will be used when connecting to the FUJITSU Enterprise Postgres server from outside of its RHOCP cluster. For example, when using logical replication between FUJITSU Enterprise Postgres servers on different RHOCP clusters, the FQDNs for both the publisher and subscriber FUJITSU Enterprise Postgres servers are required.

► **Certificates for authentication**

The certificates of the FUJITSU Enterprise Postgres server and user are required for authentication. For example, when using logical replication to use FQDN to connect to the FUJITSU Enterprise Postgres server from outside of its RHOCP cluster, the server certificate for FUJITSU Enterprise Postgres server must include the FQDNs for both the publisher and subscriber FUJITSU Enterprise Postgres servers.

► **FUJITSU Enterprise Postgres client**

For the FUJITSU Enterprise Postgres client, download the rpm from [FUJITSU Enterprise Postgres client - download](#) and install it in the client machine or in a container.

For more information about the setup, see Chapter 3, “Setup”, in *FUJITSU Enterprise Postgres 13 on IBM LinuxONE Installation and Setup Guide for Client*.

Note: Because data is communicated over the internet, a secured network is required, such as mutual authentication by Mutual Transport Layer Security (MTLS). MTLS must be set up before deploying the FUJITSU Enterprise Postgres cluster. For more information about implementing MTLS, see *FUJITSU Enterprise Postgres 13 for Kubernetes User Guide*.

7.5.1 Automatic instance creation

This section describes how to create database instances automatically by using the FUJITSU Enterprise Postgres Operator.

Modernization of data and services requires rapid response to changing consumer needs. Therefore, database deployment requires speed.

However, building a database server in an on-premises environment typically requires obtaining infrastructure resources, which is followed by database server configuration design, installation, and setup. Also, when considering HA configurations for increased reliability, deployments can be more complex.

The FUJITSU Enterprise Postgres Operator, which is available in container environments, automatically performs the work that is required to deploy these databases. Customers can specify the configuration by modifying the template file, which is standardized based on Fujitsu knowledge as needed, and perform simple operations with a GUI. Building a complex database HA configuration can be done with a declarative configuration. Modifying templates require as few as eight parameters, which are described in the following section. If other configurations are required, further tuning and customization is also possible.

Creating 3-node HA FUJITSU Enterprise Postgres cluster instances by using Red Hat OpenShift Console

Here are the step-by-step instructions to build a database cluster in a HA configuration by using a template:

1. In the Red Hat OpenShift Console, click **Installed Operators**.
2. Select the **FUJITSU Enterprise Postgres13 Operator**, as shown in Figure 7-3 on page 159.

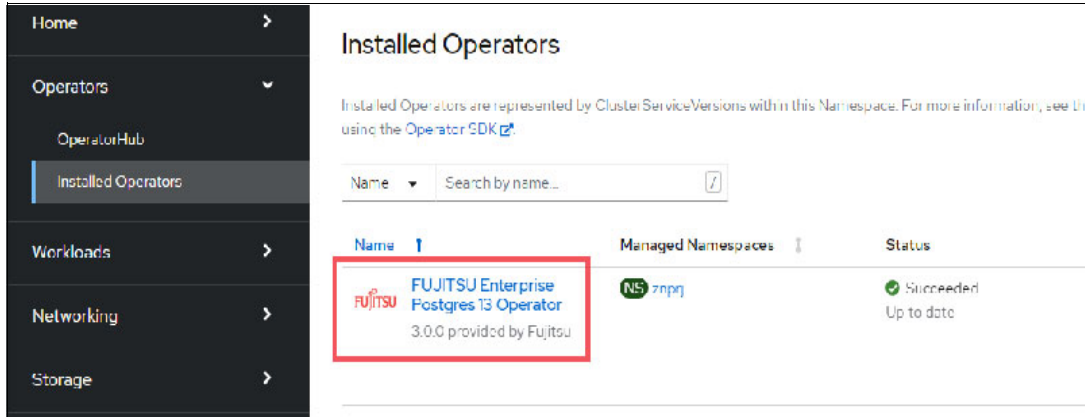


Figure 7-3 Selecting FUJITSU Enterprise Postgres13 Operator

3. In the Operator details window, select **Create Instance**, as shown in Figure 7-4.

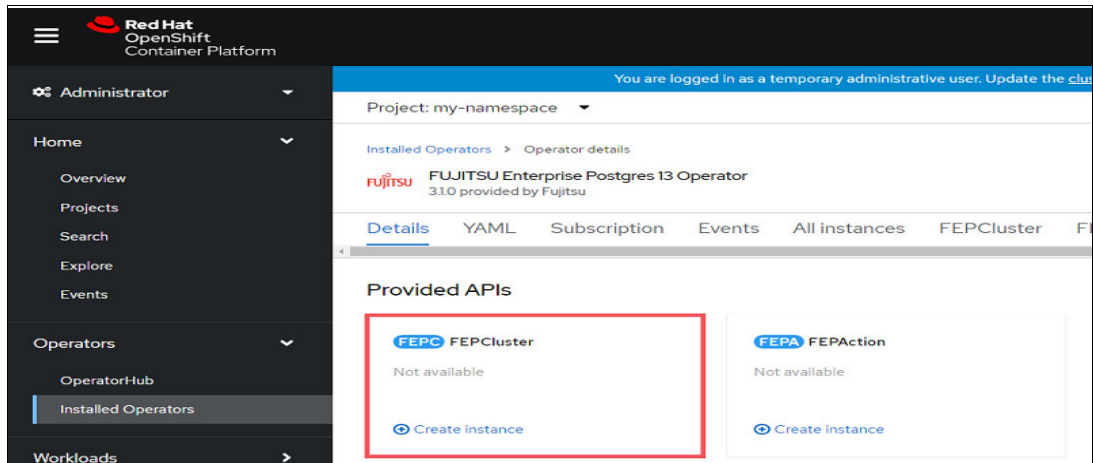


Figure 7-4 Creating a FUJITSU Enterprise Postgres cluster

4. In the Create FEPCluster window, click the **YAML** tab, and update the parameter values as shown in Table 7-1. To create a cluster, click **Create**, as shown in Figure 7-5 on page 161.

Table 7-1 FEPCluster CR configuration file details

Field	Value	Details
metadata: name:	ha-fep	Name of the FUJITSU Enterprise Postgres Cluster. Must be unique within a namespace.
spec: fep: forceSsl:	true	Ensures that communication to the server should be only through SSL. Changes are reflected in pg_hba.conf.
spec: fep: mcSpec:	limits: cpu: 500m memory: 700Mi requests: cpu: 200m memory: 512Mi	Resource that is allocated to each of the FUJITSU Enterprise Postgres pods in the cluster.

Field	Value	Details
spec: fep: instances:	3	Number of Fujitsu Enterprise Postgres pods in the cluster. In this example, we deploy three nodes (one master and two replicas).
spec: fep: syncMode	on	Replication mode. off: Asynchronous replication on: Synchronous replication
spec: fepChildCrVal: customPgHba:	host postgres postgres 10.131.0.213/32 trust	pg_hba custom rules to merge with the default rules. Inserted into pg_hba.conf. Sets the IP address of a trusted client.
spec: fepChildCrVal: sysUsers:	pgAdminPassword: admin-password	Password for postgres superuser.
	pgdb: mydb	Name of the user database to be created.
	pguser: mydbuser	Name of the user for the user database to be created.
	pgpassword: mydbpassword	Password for pguser.
	pgrepluser: repluser	Name of the replication user. This parameter is used for setting up replication between primary and replica in FUJITSU Enterprise Postgres Cluster.
	pgreplpassword: repluserpwd	Password for the replication user.
spec: fepChildCrVal: storage:	dataVol: size: 2Gi storageClass: gold walVol: size: 1200Mi storageClass: gold tablespaceVol: size: 512Mi storageClass: gold archivalVol: size: 1Gi storageClass: nfs-client accessModes: "ReadWriteMany" logVol: size: 1Gi storageClass: gold backupVol: size: 2Gi storageClass: nfs-client accessModes: "ReadWriteMany"	Storage allocation to this container. For each volume, set the disk size to be allocated and the name of the storageClass that corresponds to the pre-provisioned storage. For more information about preparing disks, see 10.2.17 "Installing IBM Spectrum Virtualize and setting up storage class" in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE</i> , SG24-8499.

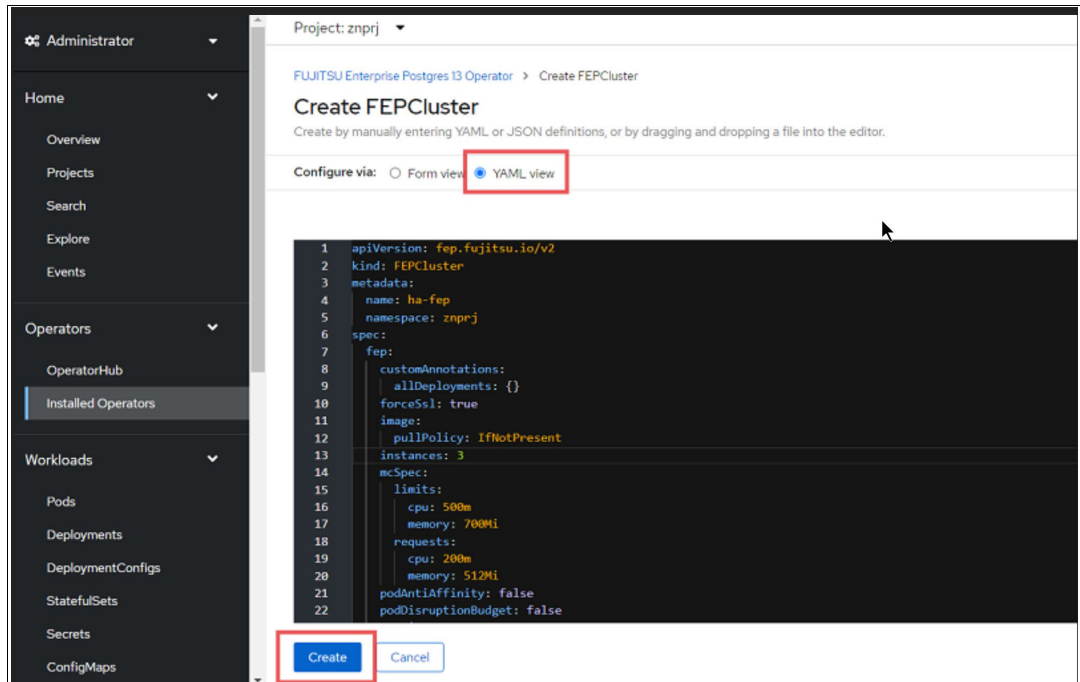


Figure 7-5 Updating parameters for deployment

- The HA cluster is deployed, and the deployment status can be checked by selecting **Workloads** → **Pods**, as shown in Figure 7-6. `ha-fep-sts-0` is the master server, and `ha-fep-sts-1` and `ha-fep-sts-2` are the replica servers against the cluster name `ha-fep` that is specified in the CR configuration file. The status shows *Running* when the cluster is ready.

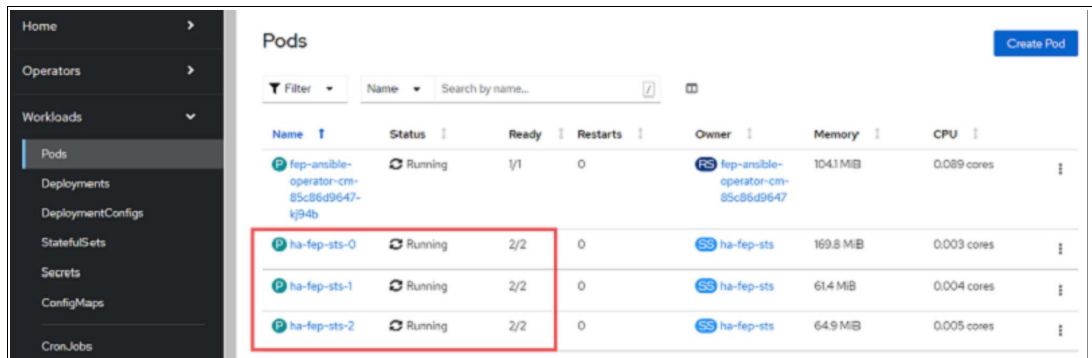


Figure 7-6 HA cluster deployment result

All FUJITSU Enterprise Postgres pods must show the status as *Running*.

You have successfully installed FUJITSU Enterprise Postgres on a Red Hat OpenShift cluster on the IBM LinuxONE server platform.

Note: For more information about the parameters of the FUJITSU Enterprise Postgres cluster CR configuration, see [FUJITSU Enterprise Postgres 13 for Kubernetes Reference Guide](#).

7.5.2 Operation

This section describes how database operations are automated with FUJITSU Enterprise Postgres Operator.

Automated operations, which are set up by declarative configurations, help organizations reduce database management costs so that developers can focus on application development.

Automatic backup

This section describes the automatic backup of FUJITSU Enterprise Postgres Operator.

Data is the lifeline for organizations, and protecting data is critical. Taking a backup of your data periodically and automatically is essential. Restoring data to the latest state is necessary in cases of disk corruption or data corruption. Point-in-time recovery (PITR) to restore data after operational or batch processing errors is also imperative.

For users to deploy and use the database system at ease, automatic backup is enabled by default in FUJITSU Enterprise Postgres Operator. Backup schedules and backup retention periods can be customized with a declarative configuration.

The following steps are demonstrated in this section:

- ▶ Updating the automatic backup definition
- ▶ Verifying the automatic backup
- ▶ Point-in-time recovery by using a backup

Updating the automatic backup definition

To update the automatic backup definition, complete the following steps.

Note: The following examples assume that the system is used in the UTC-5 time zone. FUJITSU Enterprise Postgres Operator accepts Coordinated Universal Time (UTC) time. The UTC-5 time that is used in this example is converted to Coordinated Universal Time time for the parameters.

Note: The FEPCluster monitoring feature must be enabled to view information about the backups that you did. For more information about this procedure, see , “Monitoring” on page 176.

1. In the Red Hat OpenShift Console, select **Operators** → **Installed Operators**, as shown in Figure 7-7 on page 163.

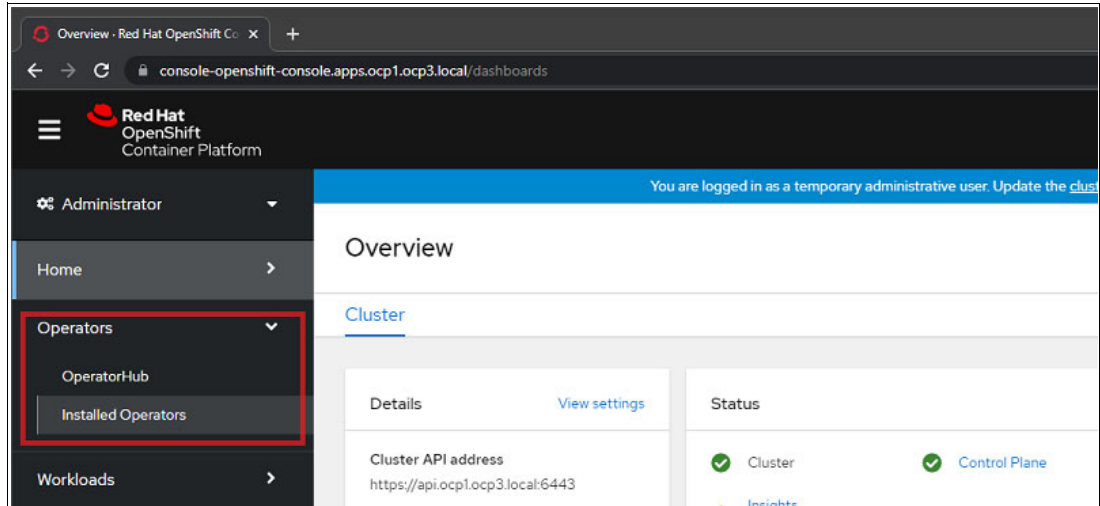


Figure 7-7 Navigating to the Installed Operators window

2. Select **FUJITSU Enterprise Postgres 13 Operator**, as shown in Figure 7-8.

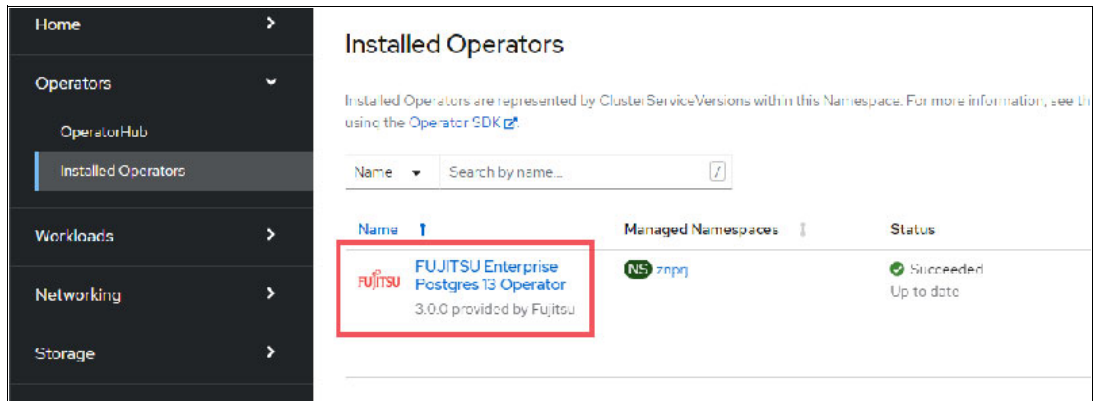


Figure 7-8 Selecting FUJITSU Enterprise Postgres 13 Operator

3. Select the **FEPCluster** tab, and select **ha-fep**, as shown in Figure 7-9.

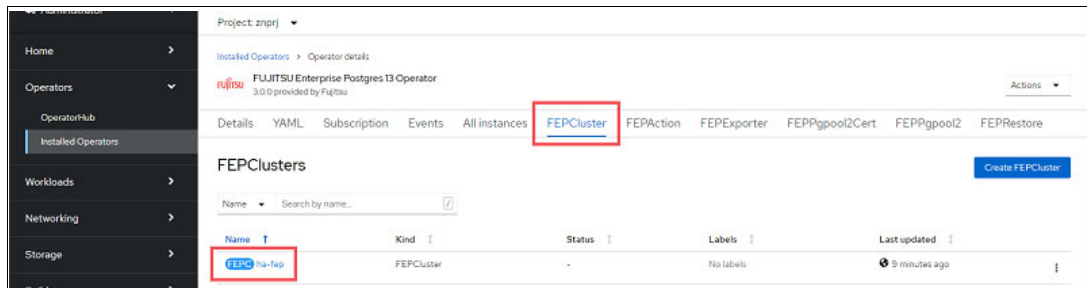


Figure 7-9 Selecting ha-fep on the FEPCluster tab

- In the FEPCluster details window, select the **YAML** tab, as shown in Figure 7-10. Set the parameters as shown in Table 7-2.

In this example, here are the backups that are taken:

- Full backup is taken at midnight every Sunday.
- Incremental backup is taken at 1300 everyday.
- Backups are retained for 5 weeks.

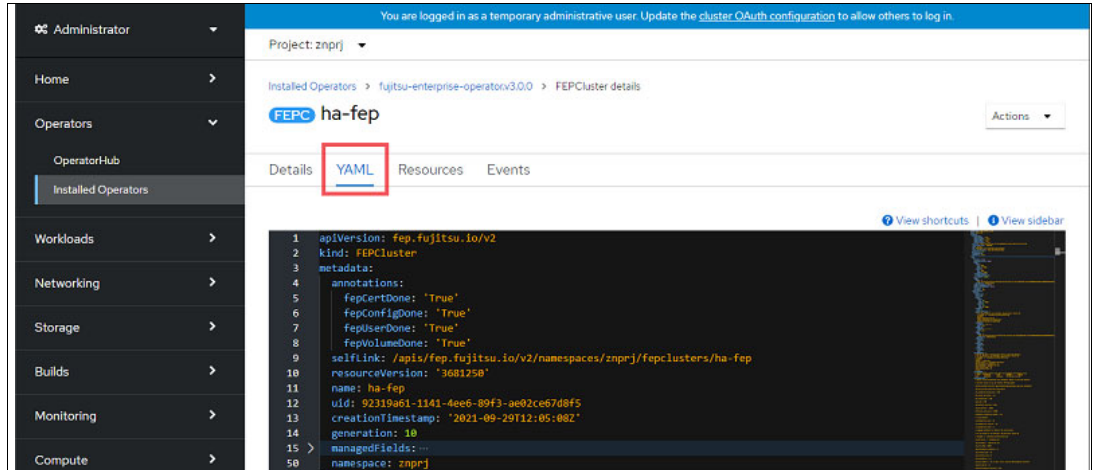


Figure 7-10 Selecting the YAML tab in the FEPCluster details window

Table 7-2 FEPCluster CR file backup settings details

Field	Value	Details
spec: fepChildCrVal: backup:	pgbackrestParams: [global] repo1-retention-full=5 repo1-retention-full- type=count	How long backups are retained. Use “ ” to specify multiple rows. You can set repo1-retention-full-type to count or time. If repo1-retention-full-type is count, then repo1-retention-full is the number of generations, and if repo1-retention-full-type is time, then repo1-retention-full is the number of days. In this example, it is set to keep the full backup of up to five generations.

Field	Value	Details
spec: fepChildCrVal: backup: schedule1: schedule	0 5 * * 0	<p>Set the first backup schedule in Cron format.</p> <p>The cron format accepts values in the fields for minutes (0 - 59), hours (0 - 23), days (1 - 31), months (1 - 12), and days (0 - 7, where 0 and 7 are Sundays) from left to right, which are separated by white spaces. "*" is used to specify the entire range of possible values for the field.</p> <p>The date and time are Coordinated Universal Time time.</p> <p>In this example, we set it to midnight every Sunday in the Coordinated Universal Time -5 time zone. For example, if you want to back up at midnight every day, set the following value:</p> <pre>0 5 * * *</pre>
spec: fepChildCrVal: backup: schedule1: type	full	<p>Set the backup type of the first backup schedule to full backup.</p> <p>full: Perform a full backup (back up the contents of the database cluster).</p> <p>incr: Perform an incremental backup (back up only the database cluster files that were changed since the last backup).</p>
spec: fepChildCrVal: backup: schedule2: schedule	0 6 * * *	<p>Set the second backup schedule in cron format.</p> <p>The date and time are Coordinated Universal Time time.</p> <p>In this example, we set the time to 13:00 every day in the Coordinated Universal Time -5 time zone.</p> <p>For example, if you want to take a backup every day at 13:00, set the following value:</p> <pre>0 6,18 * * *</pre>
spec: fepChildCrVal: backup: schedule2: type	incr	<p>Set the backup type of the second backup schedule to incremental backup.</p> <p>full: Perform a full backup (back up the contents of the database cluster).</p> <p>incr: Perform an incremental backup (back up only the database cluster files that changed since the last backup).</p>

- Click **Save** to apply the changes, as shown in Figure 7-11. A message displays that confirms that the changes were saved successfully, as shown in Figure 7-12.

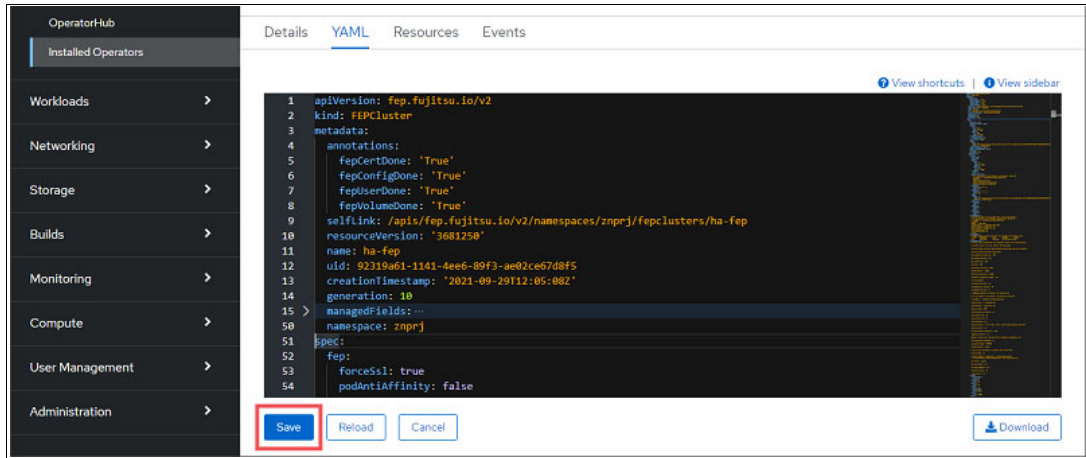


Figure 7-11 Saving configuration changes

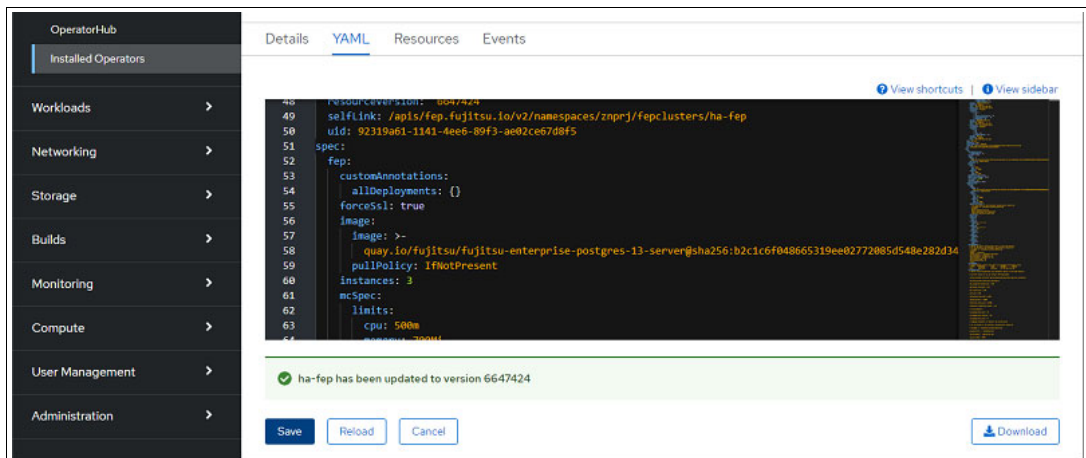


Figure 7-12 Confirmation message

Note: If saving is unsuccessful, click **Reload** to apply the changes again, and then click **Save**.

- Wait for the operator to apply the saved changes.
- Check that the backup schedule was created in the CronJobs window, as shown in Figure 7-13 on page 167.

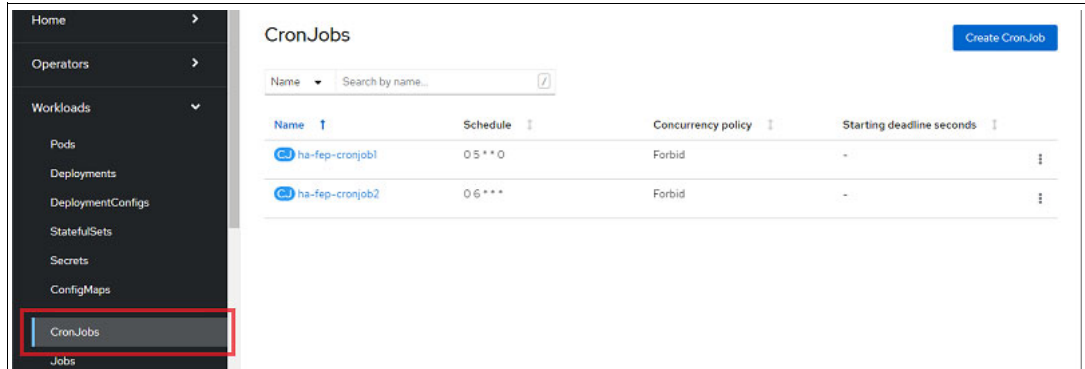


Figure 7-13 Confirming that the backup schedules were created

Verifying the automatic backup

To verify the automatic backup, complete the following steps:

1. Prepare data (Example 7-1).

By using the FUJITSU Enterprise Postgres client, create a database that is named publisher with pgbench.

Example 7-1 Inserting data by using pgbench

```
sh-4.4$ createdb -h ha-fep-primary-svc -p 27500 -U postgres publisher
Password:
sh-4.4$ pgbench -h ha-fep-primary-svc -p 27500 -U postgres -i -s 10 publisher
Password:
dropping old tables...
NOTICE: table "pgbench_accounts" does not exist, skipping
NOTICE: table "pgbench_branches" does not exist, skipping
NOTICE: table "pgbench_history" does not exist, skipping
NOTICE: table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
1000000 of 1000000 tuples (100%) done (elapsed 4.35 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 9.87 s (drop tables 0.01 s, create tables 0.06 s, client-side generate
4.99 s, vacuum 3.26 s, primary keys 1.55 s).
```

2. Check the backup information on Prometheus.

In the Red Hat OpenShift Console, select **Monitoring** → **Metrics**, as shown in Figure 7-14.

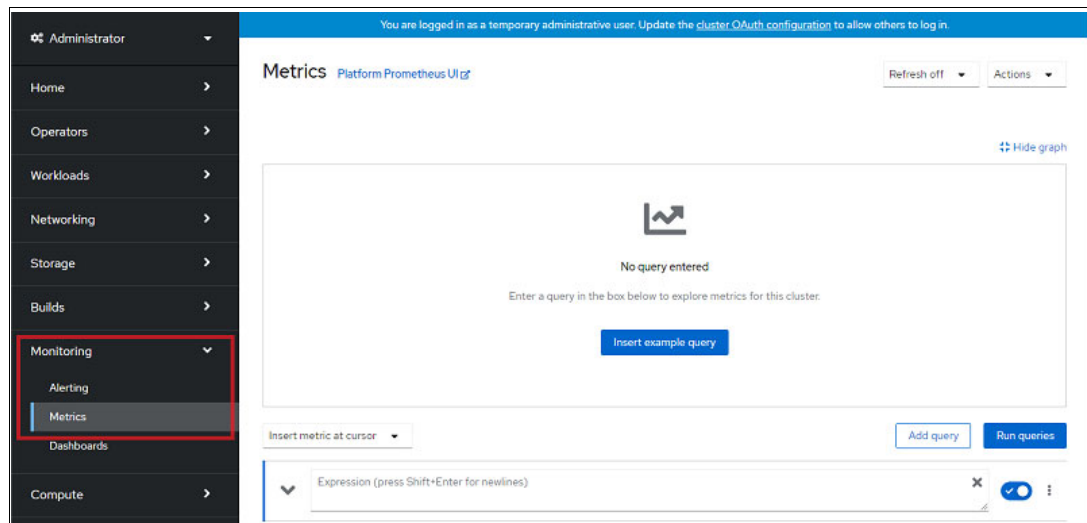


Figure 7-14 Metrics window

Note: This procedure requires the user to have a `cluster-admin` role for their Red Hat OpenShift service account.

3. Select the drop-down list **Insert metric at cursor**. In the search box, type `pg_backup_info_last_full_backup`. Possible candidates appear when you start typing, so select the appropriate metrics, as shown in Figure 7-15.

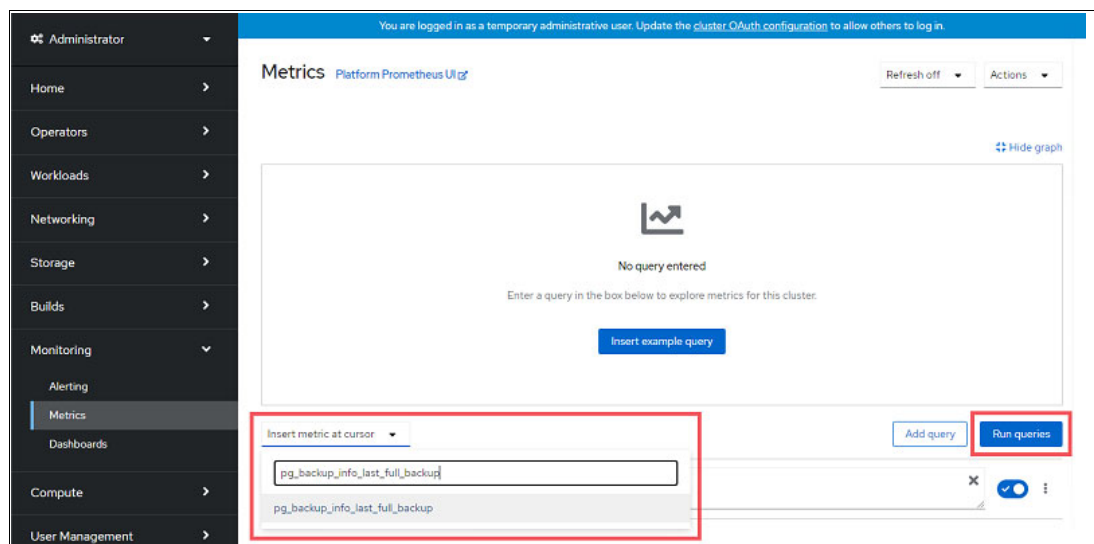


Figure 7-15 Selecting `pg_backup_info_last_full_backup`

- Click **Run queries**. A chronological graph appears at the top, and a table appears at the bottom, as shown in Figure 7-16. Check the timestamp of the latest backup under the **Value** column of the table.

Timestamps are shown in UNIX time. The following steps convert the timestamp into a more readable format:

- Copy the timestamp value.
- Run the following command in a Linux terminal:

```
$ date --date '@<TimestampValue>'
```

In our example, the command is:

```
$ date --date '@1634029228'
```

The following output is from our example command:

```
Sat Oct 2 20:15:18 EDT 2021
```

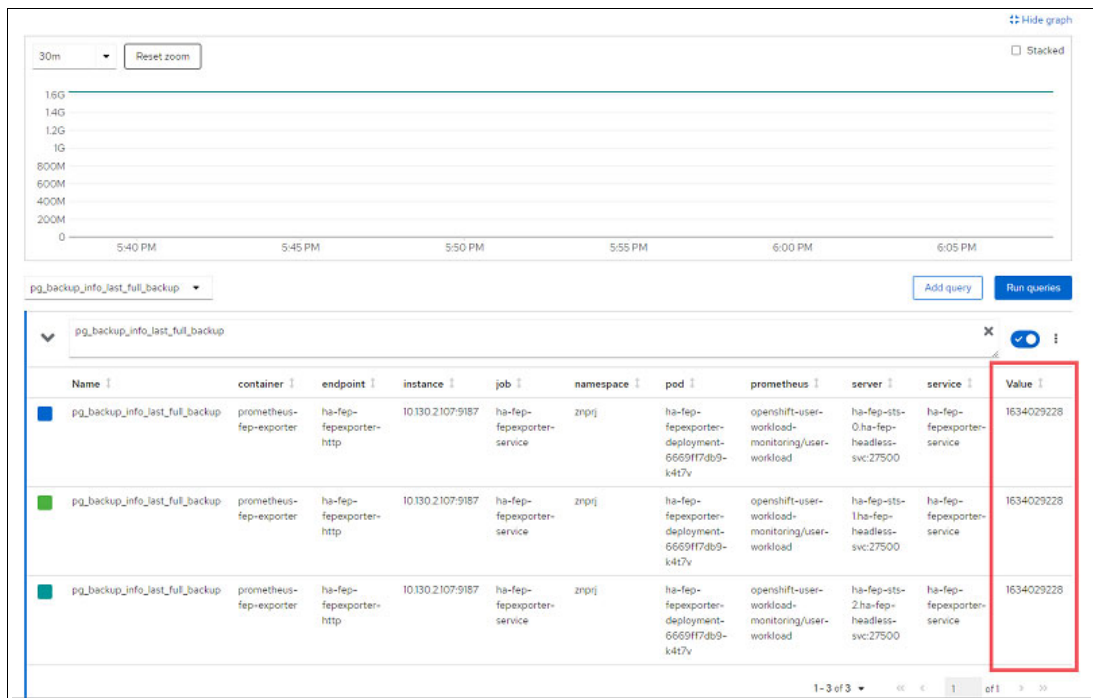


Figure 7-16 Displaying timestamps of the latest backup

- 5. Similarly, you can check the recovery windows. Select the drop-down list **Insert metric at cursor**. In the search box, type `pg_backup_info_recovery_window`. Possible candidates appear when you start typing, so select the appropriate metrics, as shown in Figure 7-17.

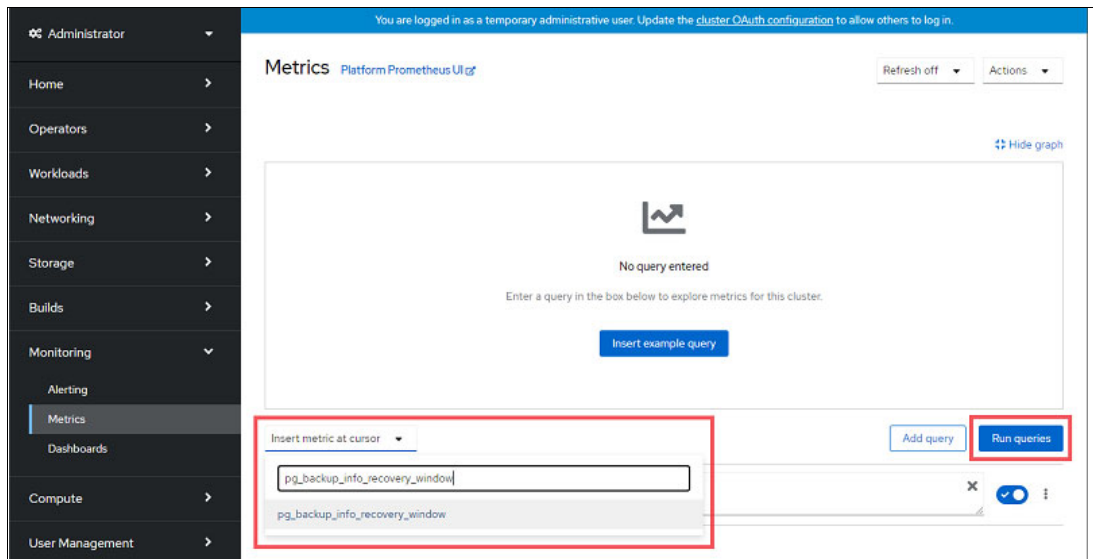


Figure 7-17 Selecting `pg_backup_info_recovery_window`

- 6. Click **Run queries**. The timestamps of recovery windows appear, as shown in Figure 7-18.

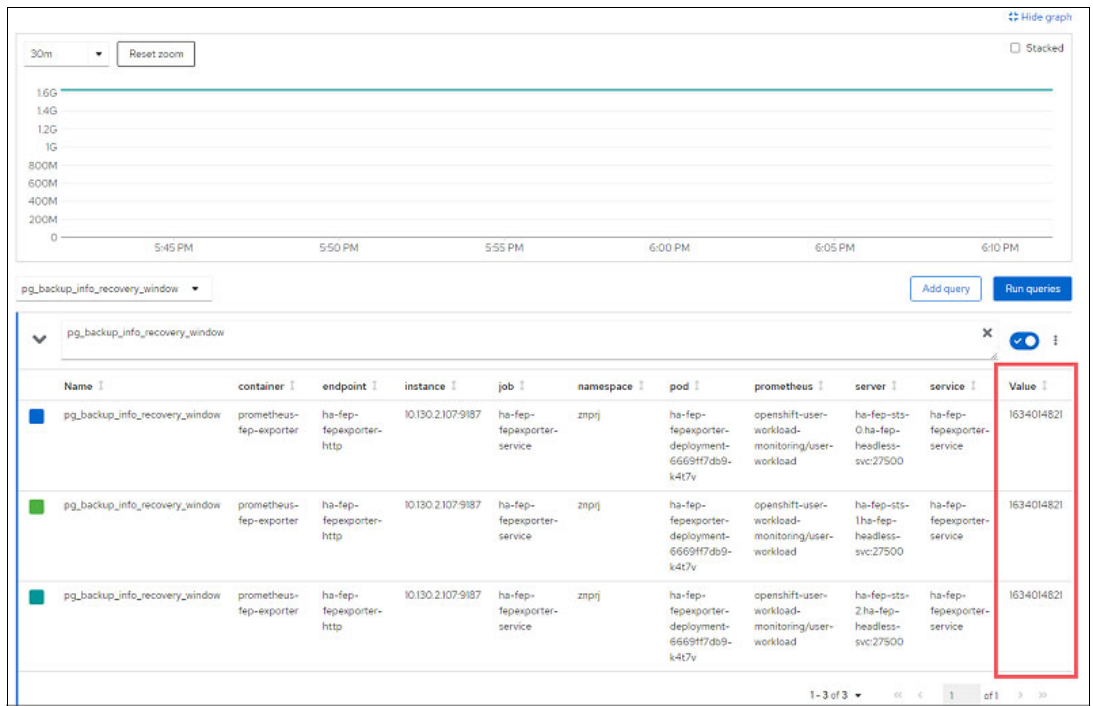


Figure 7-18 Checking recovery windows

Point-in-time recovery by using a backup

This section provides an example of a scenario where data was lost due to misoperation by a user at 10:00 on 5 October. The user will recover data from the backup that is taken at midnight of 5 October, as shown in Figure 7-19.

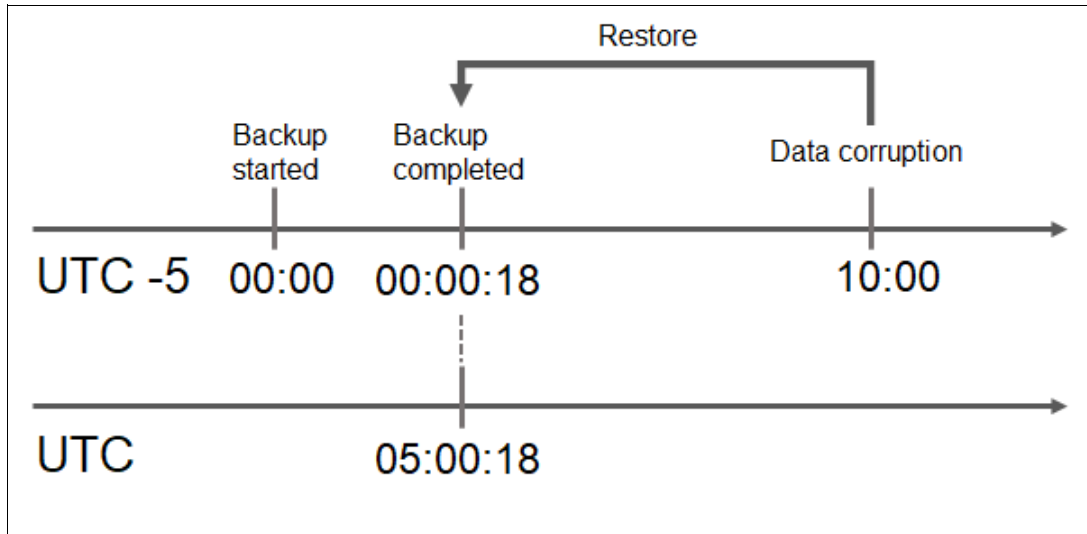


Figure 7-19 Example scenario

Complete the following steps:

1. On Red Hat OpenShift Console, clicks **Installed Operators**, as shown in Figure 7-20.

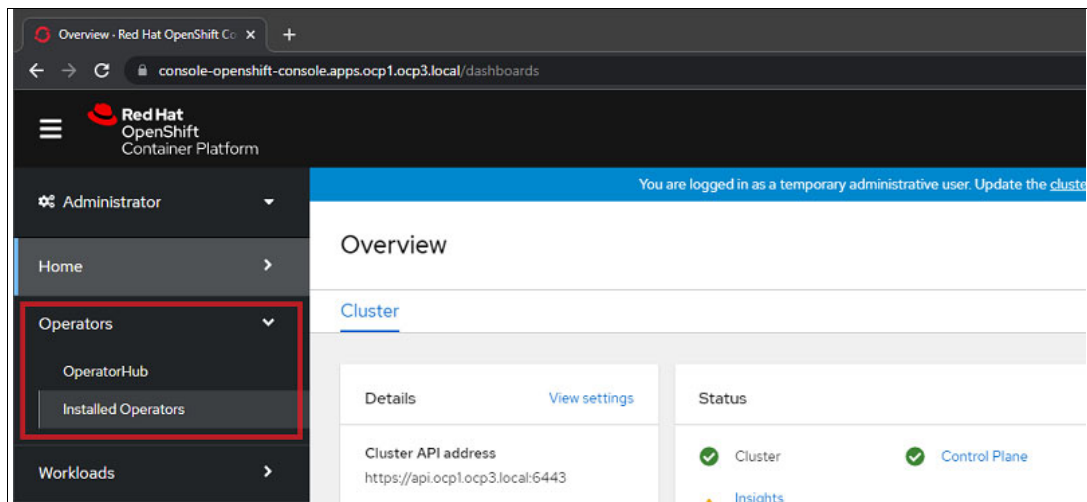


Figure 7-20 Installed Operators window

2. Select **FUJITSU Enterprise Postgres 13 Operator**, as shown in Figure 7-21.

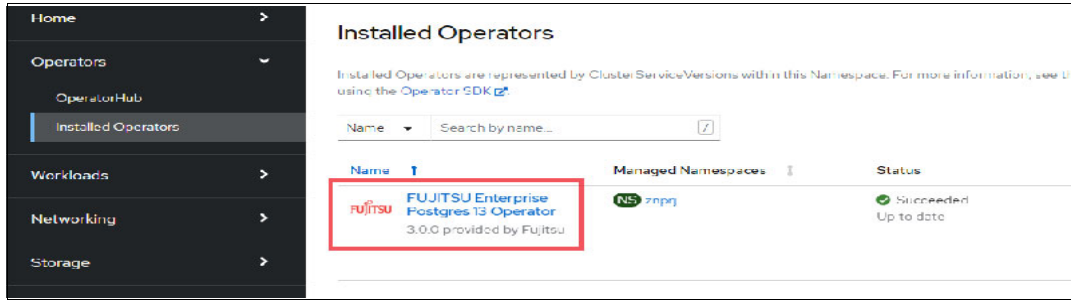


Figure 7-21 Selecting FUJITSU Enterprise Postgres 13 Operator

3. Select **Create Instance** for FEPRestore, as shown in Figure 7-22.

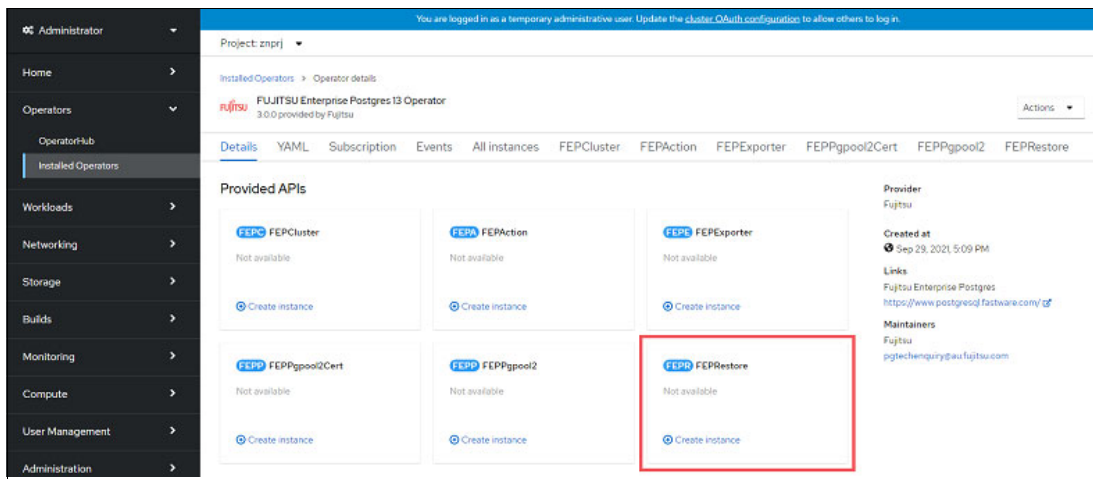


Figure 7-22 Creating FEPRestore instance

4. In the Create FEPRestore window, click the **YAML** tab. Update the values as shown in Table 7-3 and click **Create**, as shown in Figure 7-23 on page 173. Wait for the FEPCluster to be re-created and restored.

Table 7-3 FEPRestore CR configuration file details

Field	Value	Details
metadata: name:	ha-fep-restore	Name of the FEPRestore instance. Must be unique within a namespace.
metadata: namespace:	znprj	The name of the project of the source FUJITSU Enterprise Postgres cluster for restore.
spec: fromFEPcluster:	ha-fep	The name of the source FUJITSU Enterprise Postgres cluster for restore.
spec: toFEPcluster:	(Delete this parameter.)	When you omit this parameter, restore is performed on the existing cluster.
spec: restoretype:	PITR	Specify the restore type: latest: Restore to the latest state. PITR: Restore to a specified date and time.

Field	Value	Details
spec: restoredate:	“2021-10-05”	If spec.restoretype is PITR, specify the PITR target date (Coordinated Universal Time) in the YYYY-MM-DD format.
spec: restoretime:	“05:00:18”	If spec.restoretype is PITR, specify the PITR target time (Coordinated Universal Time) in the HH:MM:SS format. Use the time that you obtained in step 4 on page 169. Make sure to convert the time to Coordinated Universal Time.

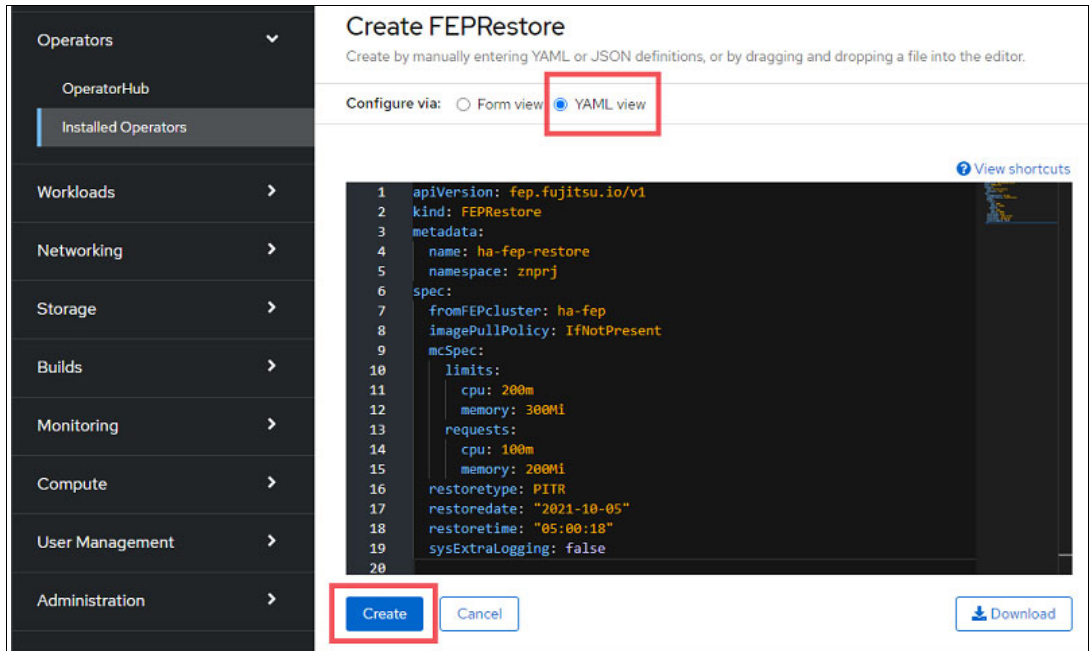


Figure 7-23 Changing the configuration and creating FEPRestore

- The restore is complete when all FUJITSU Enterprise Postgres cluster pods are re-created and the pod status is Running.
- With FUJITSU Enterprise Postgres client, check that the database is restored (Example 7-2).

Example 7-2 Checking the database that was restored

```
sh-4.4$ psql -h ha-fep-primary-svc -p 27500 -U postgres -d publisher
Password for user postgres:
psql (13.3)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression:
off)
Type "help" for help.
publisher=#
publisher=# SELECT COUNT(*) FROM public.pgbench_accounts;
count
-----
1000000
(1 row)
```

Autohealing

This section describes the autohealing features of FUJITSU Enterprise Postgres Operator.

Organizations must be prepared for any kind of failure in database operations. However, creating and rehearsing recovery procedures is costly. In addition, systems with strict availability requirements require HA configurations, which demand extensive workloads for system administrators.

FUJITSU Enterprise Postgres Operator enables automatic failover and automatic recovery to recover systems without human intervention in the event of a problem. Organizations benefit from the stability of the database systems that are provided with less cost.

Automatic failover promotes a replica pod to master when the master pod fails, and the database connection is switched. Automatic recovery re-creates the failed pod and restores the database multiplexing configuration. To use automatic failover, see 7.3, “Oracle containers” on page 156 to learn how to deploy a FUJITSU Enterprise Postgres cluster in an HA configuration.

Simulating automatic failover and automatic recovery

In this section, we simulate the behavior of the system when the master pod goes offline. To perform a failover test before a service release, complete the following steps:

1. In the Red Hat OpenShift Console, select **Workload** → **Pod**.
2. Identify the current master pod. Click **Name** to open the drop-down list and select **Label**. In the search box, type `feprle=master` and `app=ha-fep-sts`. Possible candidates appear when you start typing, so select the appropriate label, as shown in Figure 7-24. The list of master pods displays, and we can confirm that `ha-fep-sts-0` is a master pod.

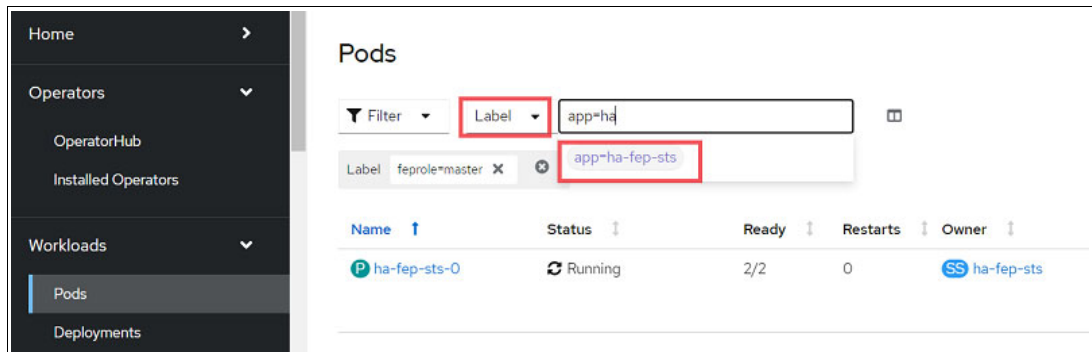


Figure 7-24 Identifying the current master pod

3. To simulate a failure, remove the master pod. Click the menu icon of `ha-fep-sts-0` and select **Delete Pod**, as shown in Figure 7-25 on page 175.

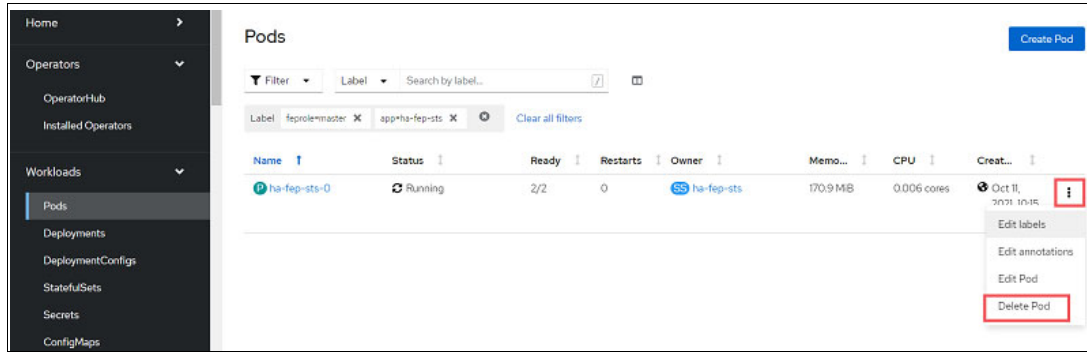


Figure 7-25 Removing the master pod

4. The status of ha-fep-sts-0 pod changes to *Terminating*, as shown in Figure 7-26.

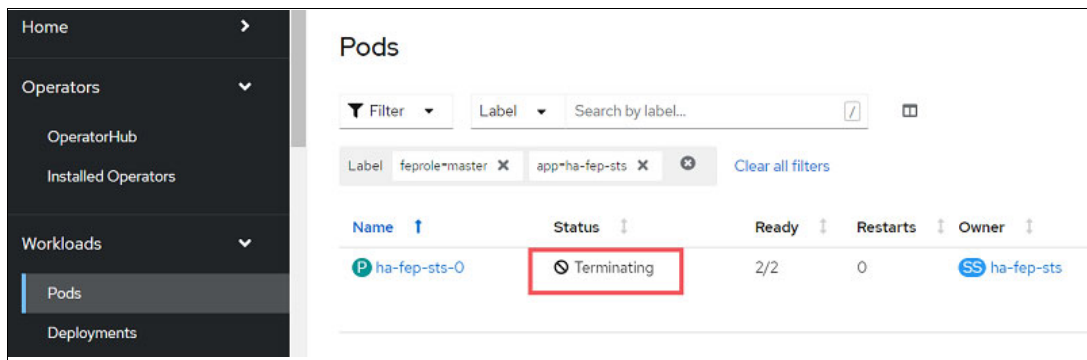


Figure 7-26 Checking the master pod status

5. The pod that was promoted to master appears, as shown in Figure 7-27. In this simulation, we can see that ha-fep-sts-2 pod was promoted to the master.

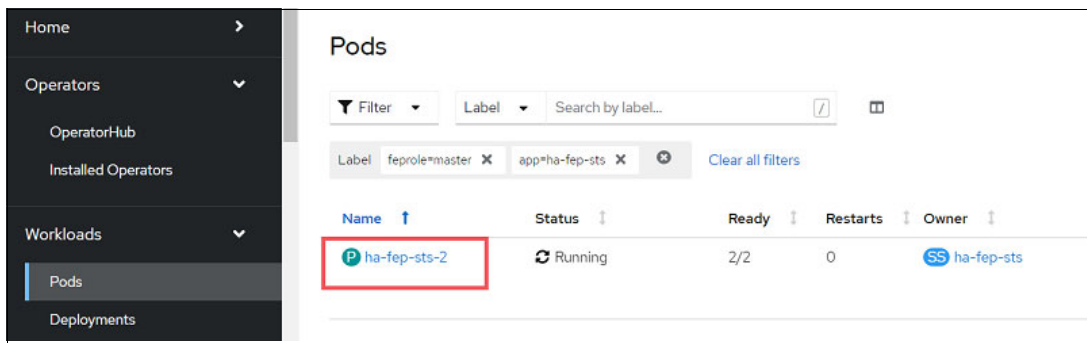


Figure 7-27 Master pod after automatic failover

6. To check the results of automatic recovery, open the replica pods in the list by removing the `feprimary=master` label, as shown in Figure 7-28. The status of `ha-fep-sts-0`, which is the old master pod, changed to *Running*. We can confirm that automatic recovery completed.

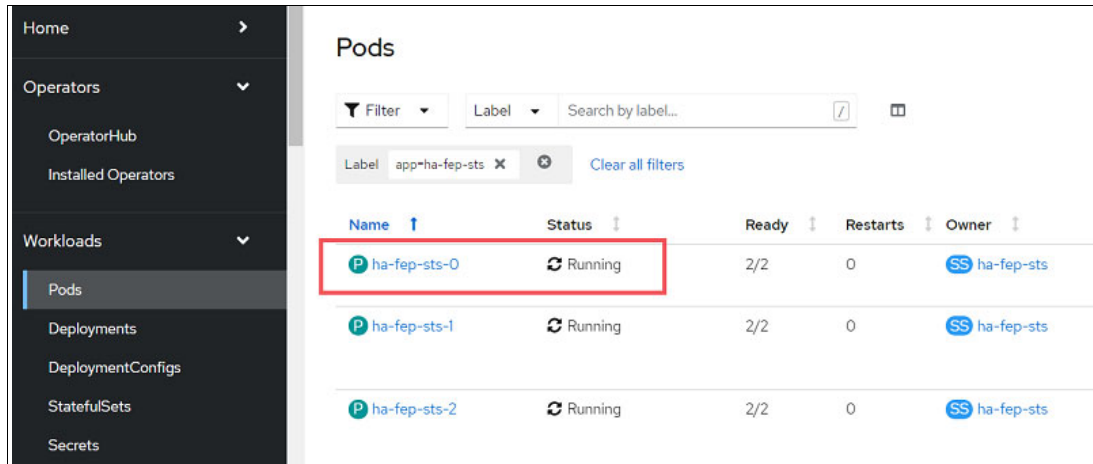


Figure 7-28 Checking automatic recovery

7. Type `feprimary=replica` into the search box. The list of replica pods displays, as shown in Figure 7-29. We can confirm that the old master `ha-fep-sts-0` is now running as a replica pod, and the database multiplexing configuration is restored.

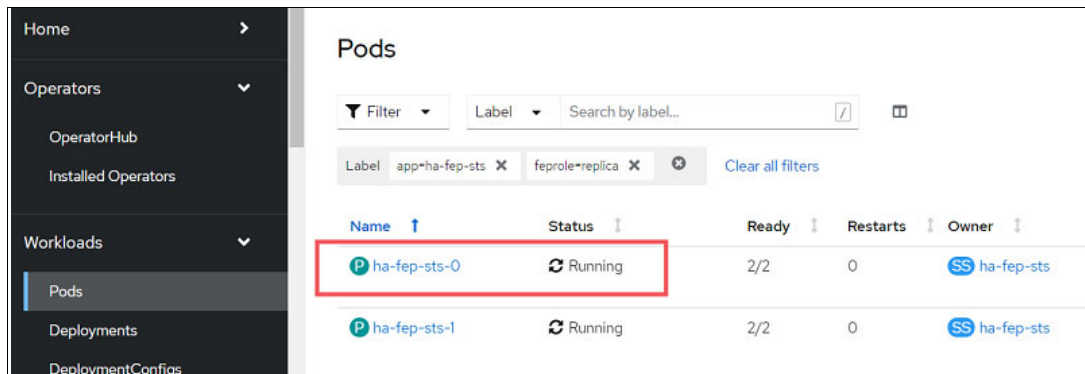


Figure 7-29 Replica pods after automatic failover

Monitoring

This section describes the monitoring capabilities of FUJITSU Enterprise Postgres Operator.

In addition to predictable fluctuations such as seasonal fluctuations and known events, database operations also face unpredictable fluctuations such as changes in trends. These unpredictable changes are becoming more prevalent, and monitoring is essential for stable system operation.

Furthermore, during continuous system reforms, it is important to identify by monitoring early signs of resource shortages such as in disk and memory so that system expansion can be planned and conducted.

The Grafana, Alertmanager, and Prometheus stack

Many software components that run as containers are open-source software, and integrations among the components are actively worked on to provide optimal usability of the system.

Grafana and Prometheus are two of the most common open-source components that are integrated in a container management platform and containerized software for observability. Prometheus is a pull-based metrics collection component. Prometheus comes with a built-in real-time alerting mechanism that is provided by Alertmanager so that users can use existing communication applications to receive notifications. Grafana is a visualization layer that is closely tied to Prometheus that offers a template function for dynamic dashboards that can be customized. Grafana, Alertmanager, and Prometheus together are referred to as the “GAP stack”, and they provide a convenient way to monitor the health of the cluster and the pods that run inside the cluster.

FUJITSU Enterprise Postgres Operator provides a standard Grafana user interface that organizations can use to start monitoring basic database information. In addition, it is possible for system administrators to respond to sudden fluctuations by using the alert function.

The following tasks are demonstrated in this section:

- ▶ Settings for monitoring
- ▶ Checking monitoring metrics
- ▶ Alert settings
- ▶ Confirming alert settings
- ▶ Using custom Grafana dashboards

Settings for monitoring

This section explains how to set up monitoring.

Note: Before completing the following steps, you must enable monitoring in your project. For more information, see [Enabling monitoring for user-defined projects](#).

1. Enable the monitoring settings for the FUJITSU Enterprise Postgres cluster. For a new deployment, add the parameter that is shown in Table 7-4 by performing step 4 on page 159. For a deployed cluster, add this parameter into the existing FEPCluster CR. To save the changes, click **Save**.

Table 7-4 FEPCluster CR file configuration details

Field	Value	Details
spec: fep: monitoring: enable:	true	When using the monitoring feature, set it to true. FEPEXporter is created when this parameter is set to true.

2. Exporter is deployed. Select **Workloads** → **Pods** and check that the pod status for Exporter (ha-fep-fepexporter-deployment-XXX) is *Running*, as shown in Figure 7-30.

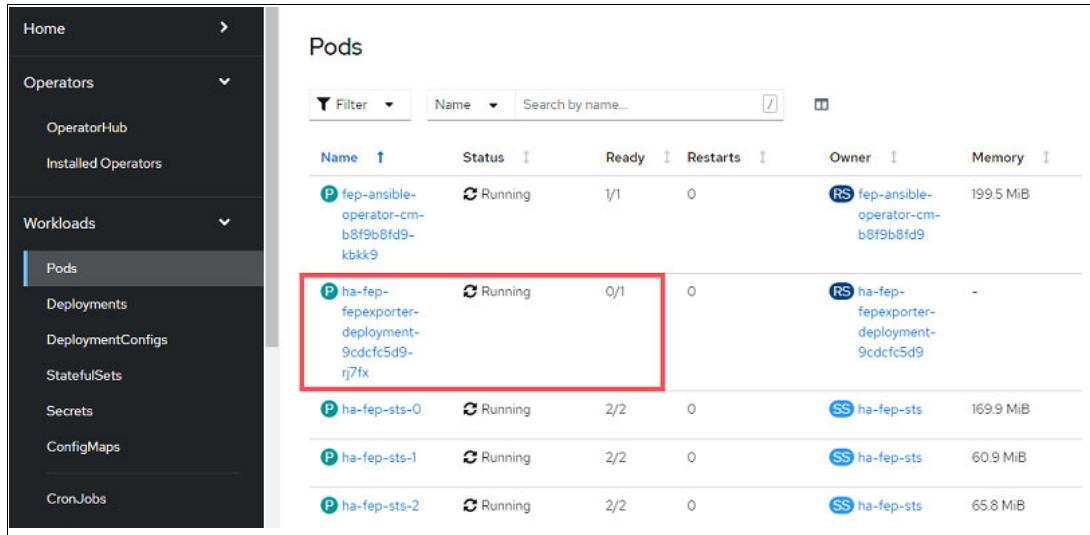


Figure 7-30 Checking the Exporter status

Checking monitoring metrics

Complete the following steps:

1. Check the monitoring metrics on Prometheus by selecting **Monitoring** → **Dashboards**, as shown in Figure 7-31.

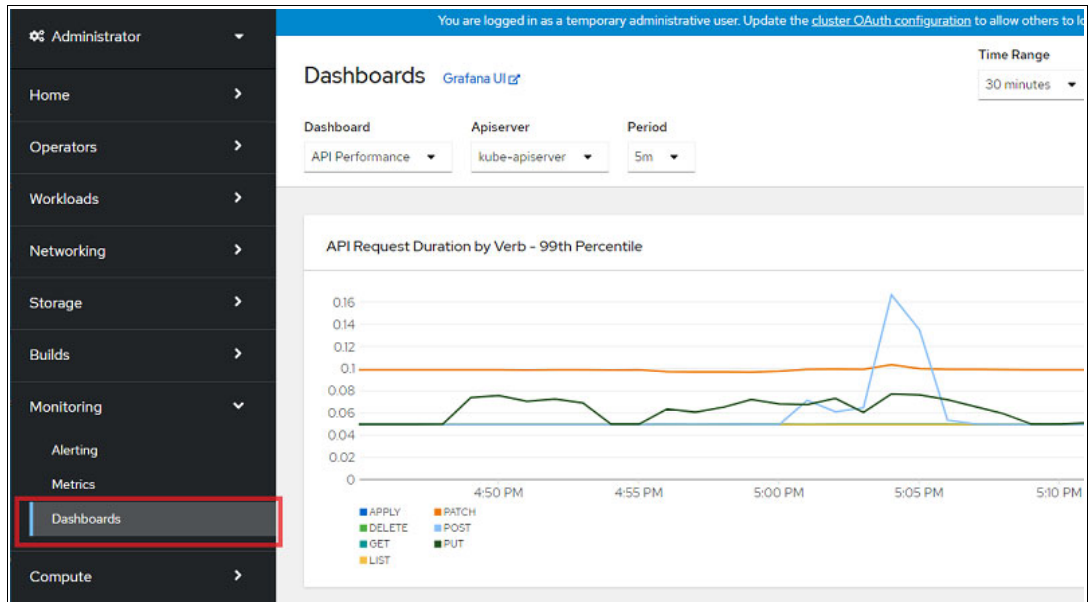


Figure 7-31 Navigating to the monitoring dashboard window

2. In the drop-down list for **Dashboard**, select **Kubernetes / Compute Resources / Pod**, as shown in Figure 7-32 on page 179.

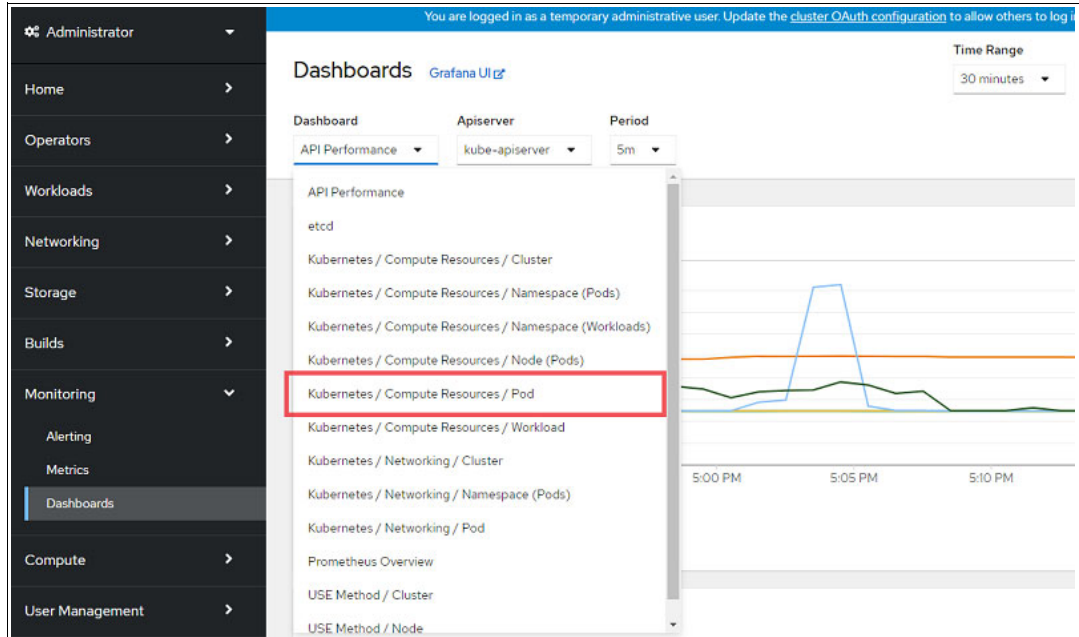


Figure 7-32 Selecting monitoring metrics

3. In the drop-down list for **Namespace**, select the project that was created in 7.5.1, “Automatic instance creation” on page 158. In the drop-down list for **Pod**, select **ha-fep-sts-0**. CPU utilization can be checked, as shown in Figure 7-33. Based on the information that is displayed in this window, database administrators can decide whether a resource should be added.

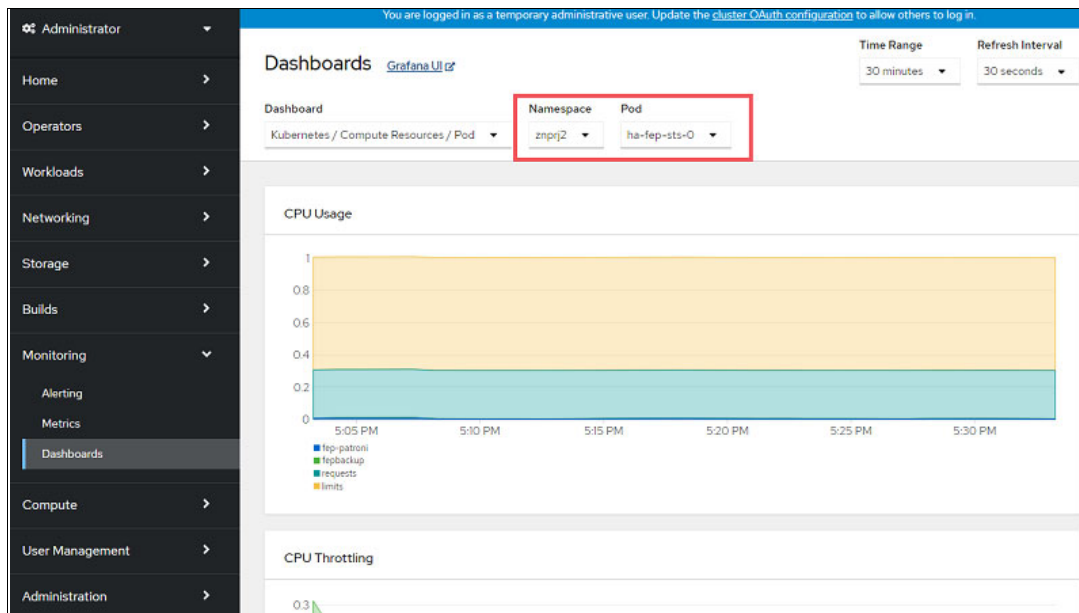


Figure 7-33 Checking the CPU utilization on Prometheus

- In addition, graphs can be viewed by using the Grafana sample template. Select **Grafana UI** at the top of the window, as shown in Figure 7-34.

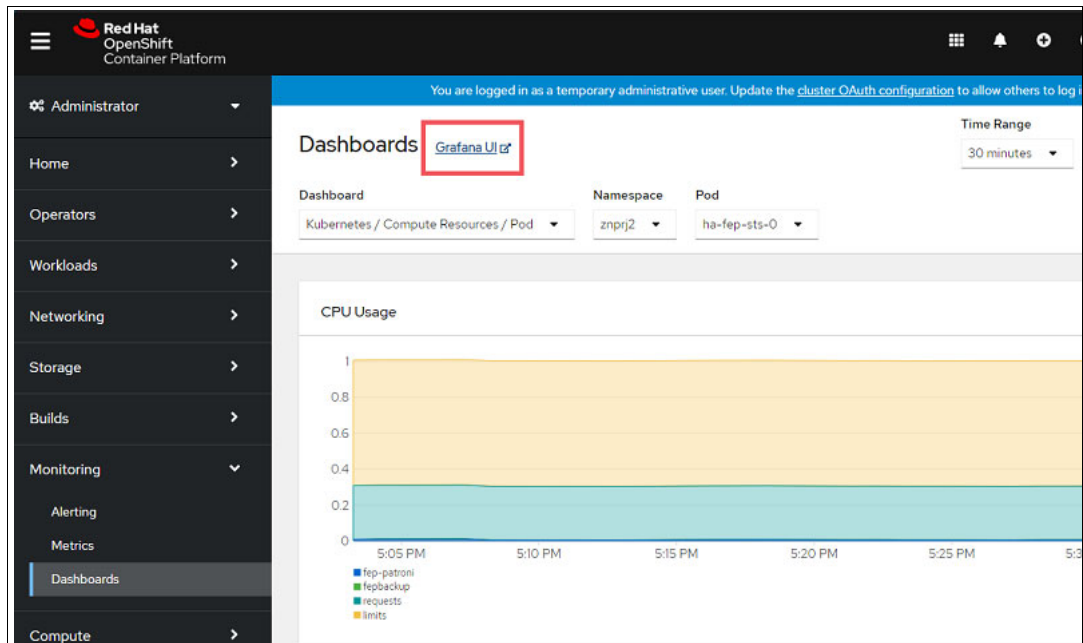


Figure 7-34 Selecting Grafana UI

- Grafana opens in a separate window. When prompted to log in, enter your credentials. If permission is requested, grant permission.
- On the Grafana home window, select the search icon, as shown in Figure 7-35.

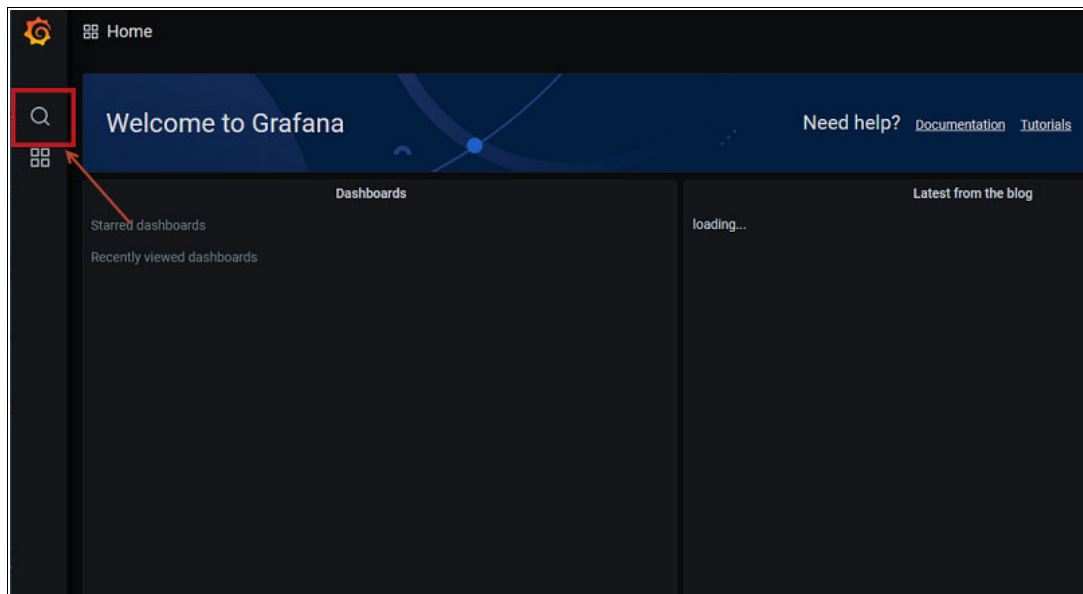


Figure 7-35 Selecting the search icon on the Grafana UI home page

- Select **Kubernetes / Compute Resources / Pod** under the **Default** folder, as shown in Figure 7-36 on page 181.

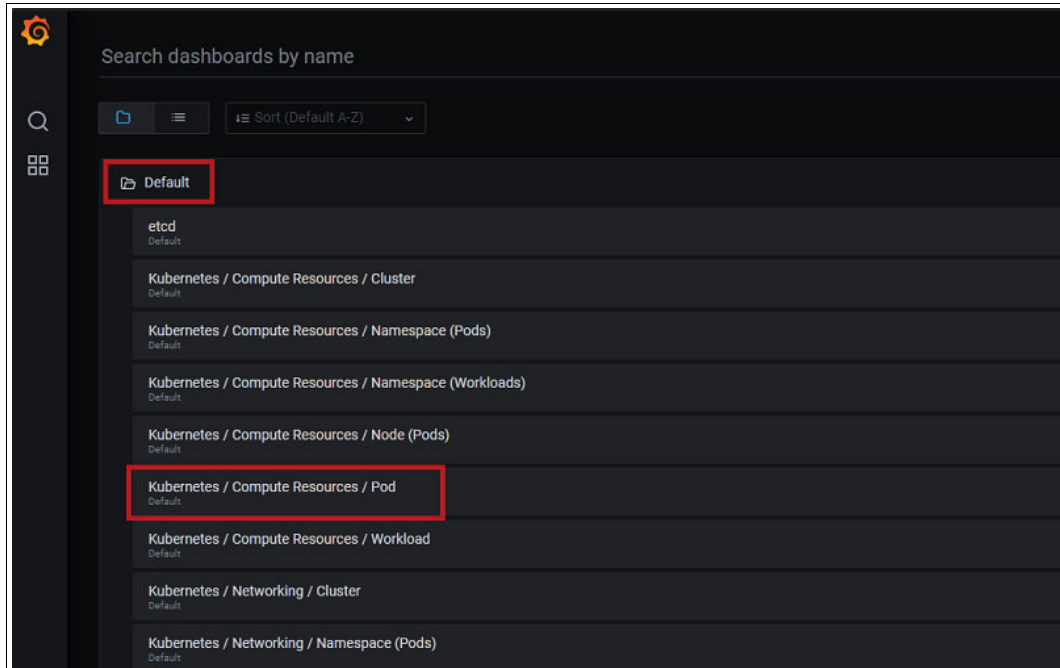


Figure 7-36 Selecting CPU utilization on the Grafana UI

8. In each drop-down list, select the target namespace and pod. The metrics for the selected pod appear, and resource information can be viewed in rich GUI, as shown in Figure 7-37. In this example, users see that a CPU resource configuration is appropriate because the pod is running within the CPU resource allocation.

Note: Grafana, which is used in this example, is integrated with the RHOCP cluster. Dashboards cannot be customized. To customize the dashboard, see “Using custom Grafana dashboards” on page 187.

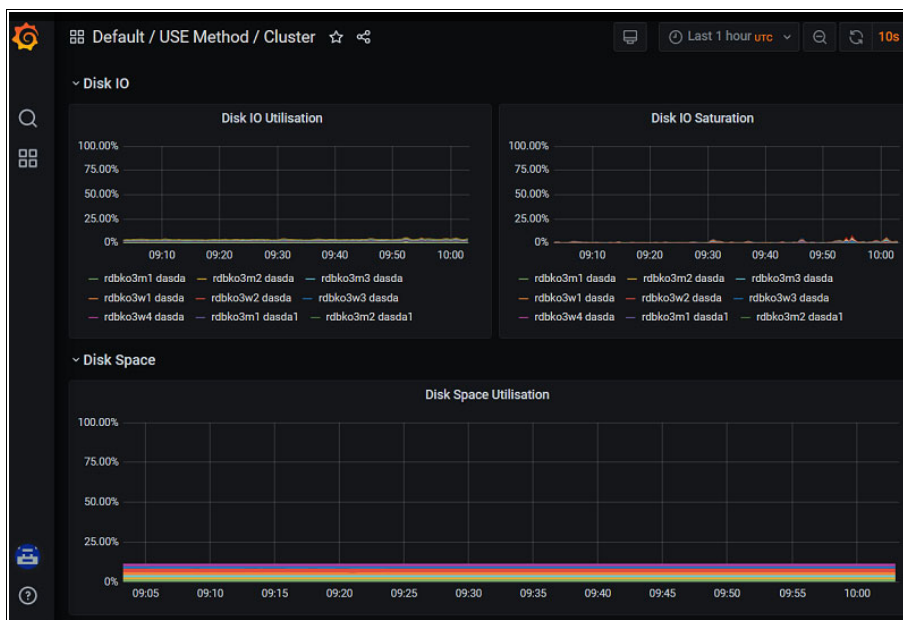


Figure 7-37 Checking the CPU utilization on the Grafana UI

9. You can also check the disk usage. Select the search icon, and then select **USE Method /Cluster** under the **Default** folder, as shown in Figure 7-38.

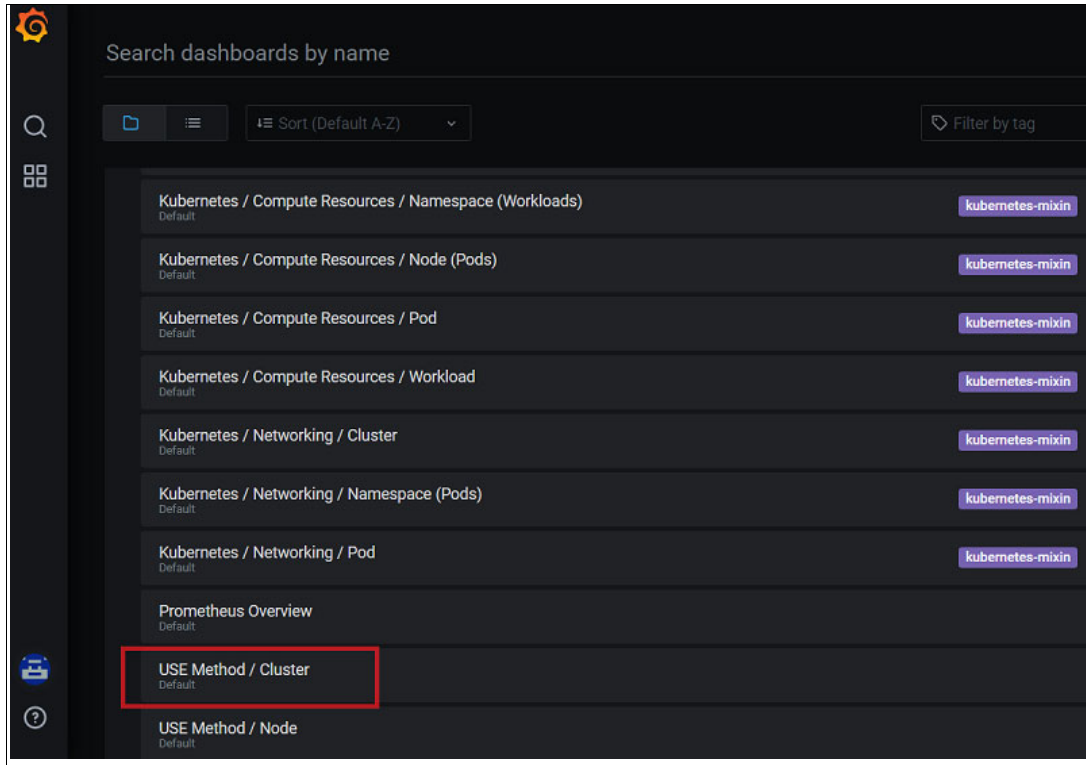


Figure 7-38 Selecting disk information on the Grafana UI

10. Scroll down to view a graph of the disk usage, as shown in Figure 7-39.

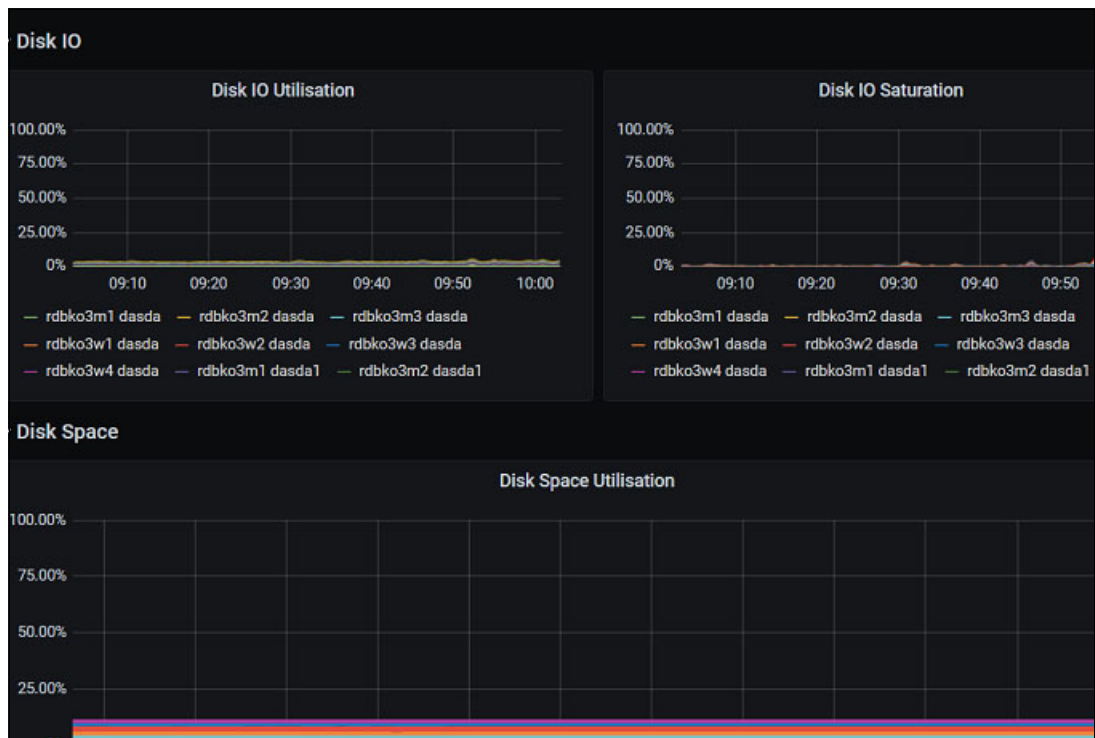


Figure 7-39 Checking the disk usage on the Grafana UI

Alert settings

By default, FUJITSU Enterprise Postgres Operator comes with an alert rule that sends out notifications if the number of connections exceeds 90% of the maximum number of connections that is possible. In Alertmanager, set an email receiver and set routing so that the alert is routed to the email receiver.

Note: For the list of default alerts, see [FUJITSU Enterprise Postgres 13 for Kubernetes User Guide](#).

Alert rules are configurable. Alert levels, intervals, and thresholds can be set for any monitoring metrics. For more information, see [FUJITSU Enterprise Postgres 13 for Kubernetes User Guide](#).

1. On the Red Hat OpenShift Console, select **Administration** → **Cluster Settings**, as shown in Figure 7-40.

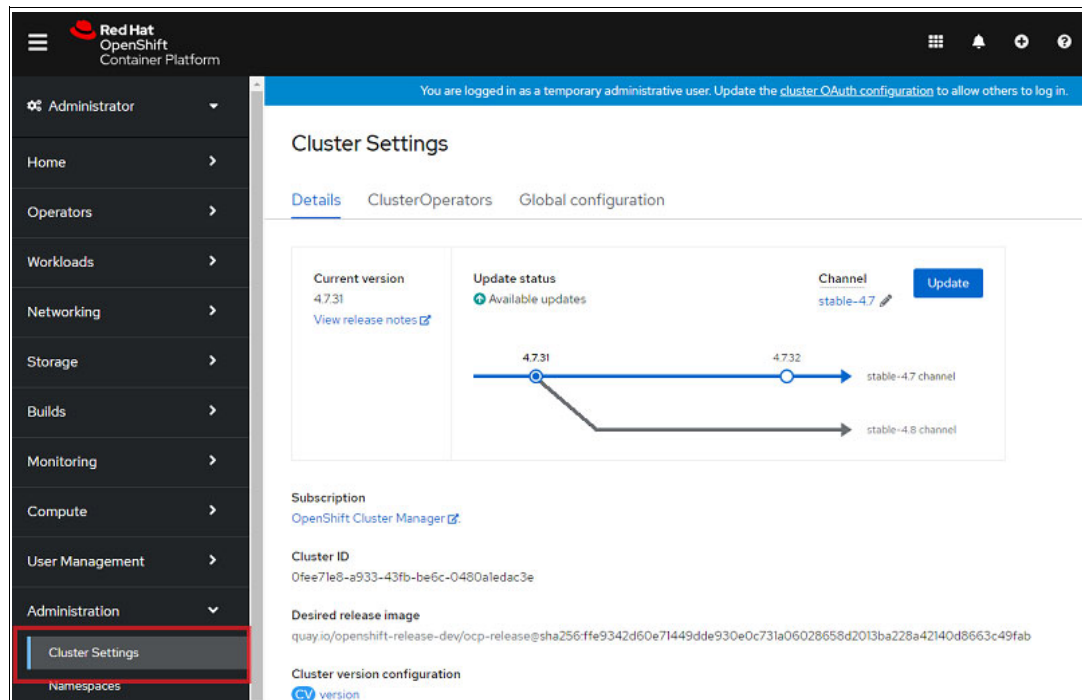


Figure 7-40 Navigating to the Cluster Settings window

2. Select the **Global configuration** tab and select **Alertmanager**, as shown in Figure 7-41.

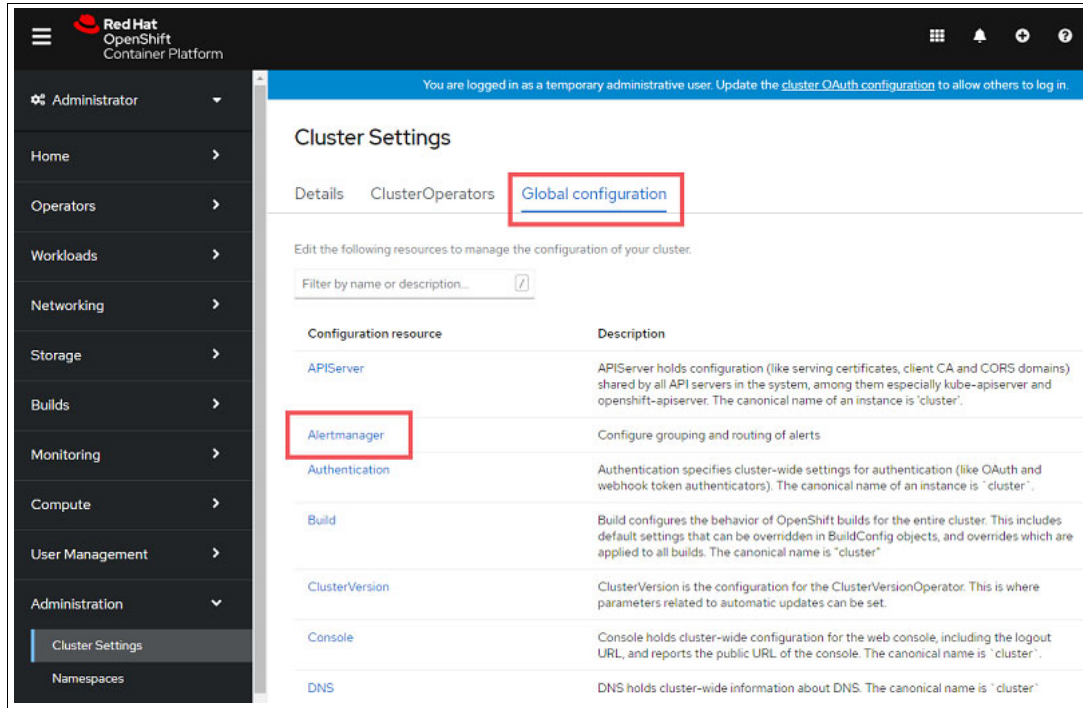


Figure 7-41 Selecting Alertmanager

3. Select **Create Receiver**, as shown in Figure 7-42.

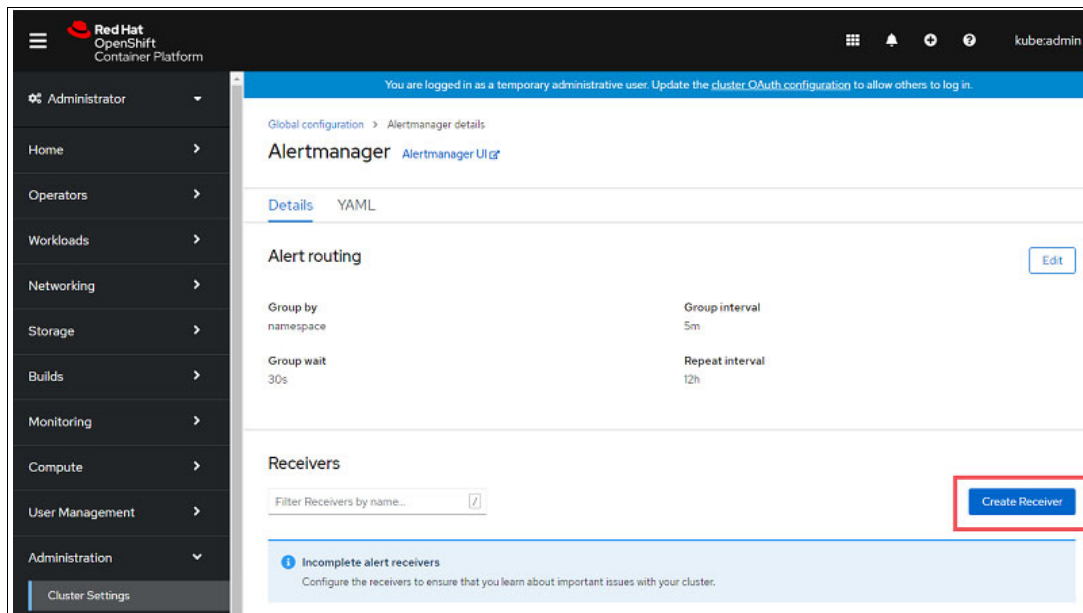
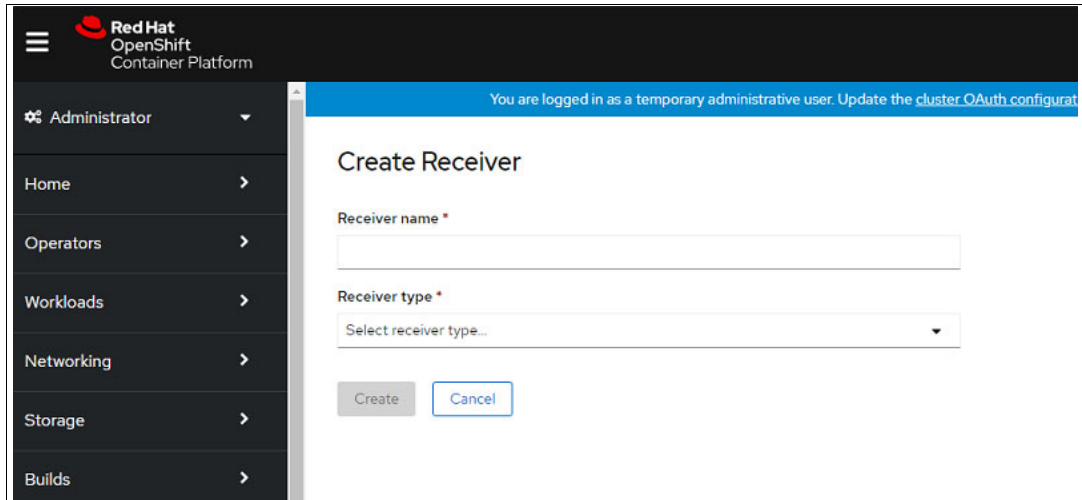


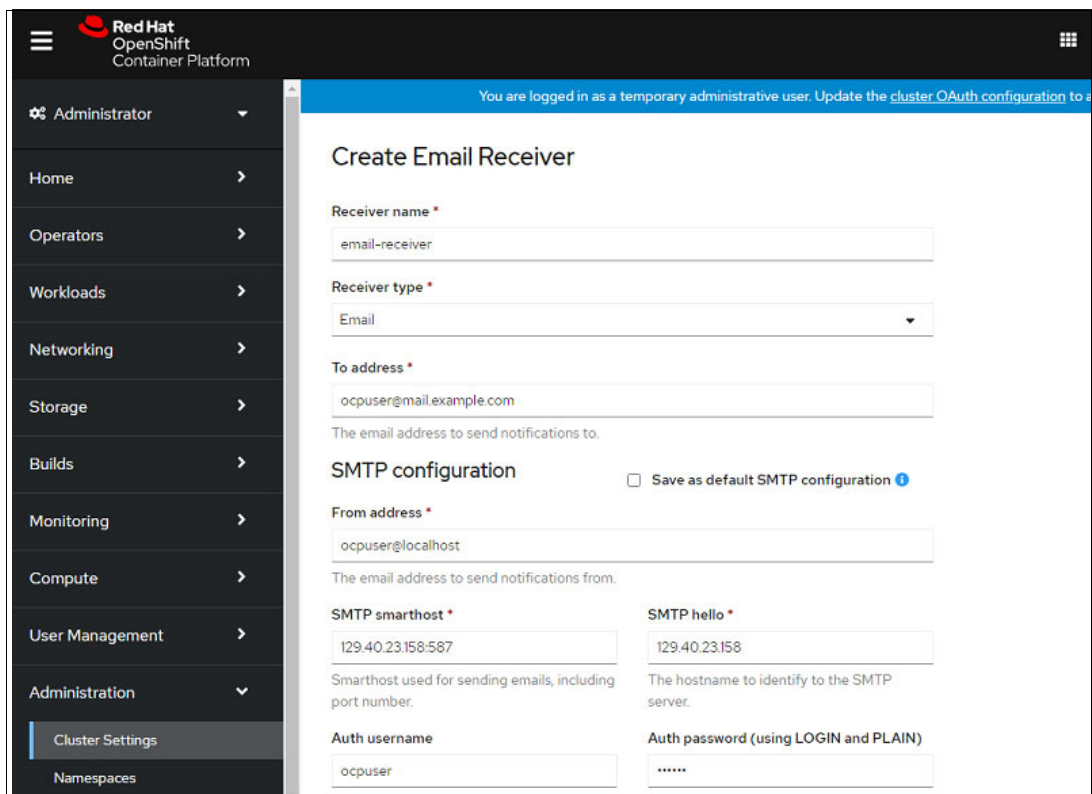
Figure 7-42 Selecting Create Receiver

4. Enter the **Receiver name** and select **Email** for the **Receiver type**, as shown in Figure 7-43. The detail fields appear, as shown in Figure 7-44.



The screenshot shows the 'Create Receiver' form in the Red Hat OpenShift Container Platform. The form is titled 'Create Receiver' and has a blue header bar with the text 'You are logged in as a temporary administrative user. Update the cluster OAuth configuration to...'. The form contains two main input fields: 'Receiver name' (a text input field) and 'Receiver type' (a dropdown menu with the option 'Email' selected). Below these fields are two buttons: 'Create' and 'Cancel'. The left sidebar shows the navigation menu with 'Administrator' selected.

Figure 7-43 Creating an email receiver



The screenshot shows the 'Create Email Receiver' form in the Red Hat OpenShift Container Platform. The form is titled 'Create Email Receiver' and has a blue header bar with the text 'You are logged in as a temporary administrative user. Update the cluster OAuth configuration to...'. The form contains several input fields and a checkbox: 'Receiver name' (text input, value: 'email-receiver'), 'Receiver type' (dropdown menu, value: 'Email'), 'To address' (text input, value: 'ocpuser@mail.example.com'), 'SMTP configuration' (checkbox, value: 'Save as default SMTP configuration'), 'From address' (text input, value: 'ocpuser@localhost'), 'SMTP smarthost' (text input, value: '129.40.23.158:587'), 'SMTP hello' (text input, value: '129.40.23.158'), 'Auth username' (text input, value: 'ocpuser'), and 'Auth password (using LOGIN and PLAIN)' (password input, value: '*****'). The left sidebar shows the navigation menu with 'Administrator' selected.

Figure 7-44 Detailed fields for an email receiver

5. As needed by your environment, enter details such as the email address and SMTP server.

- 6. In the **Routing labels** section, map this receiver to PostgreSQLTooManyConnections, as shown in Figure 7-45.

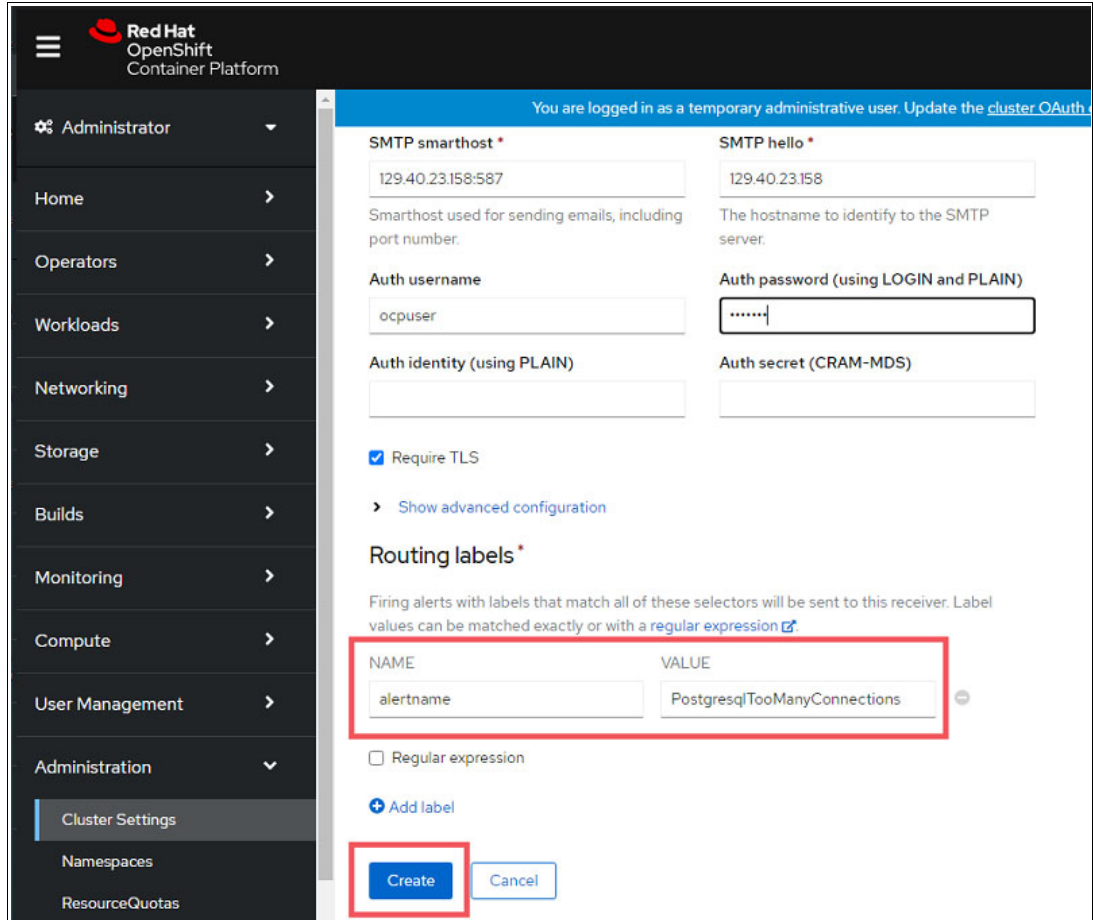


Figure 7-45 Routing alerts to the receiver

Confirming alert settings

Receivers are notified that the number of connections reached 90% of the maximum number of connections, as shown in Figure 7-46 on page 187.

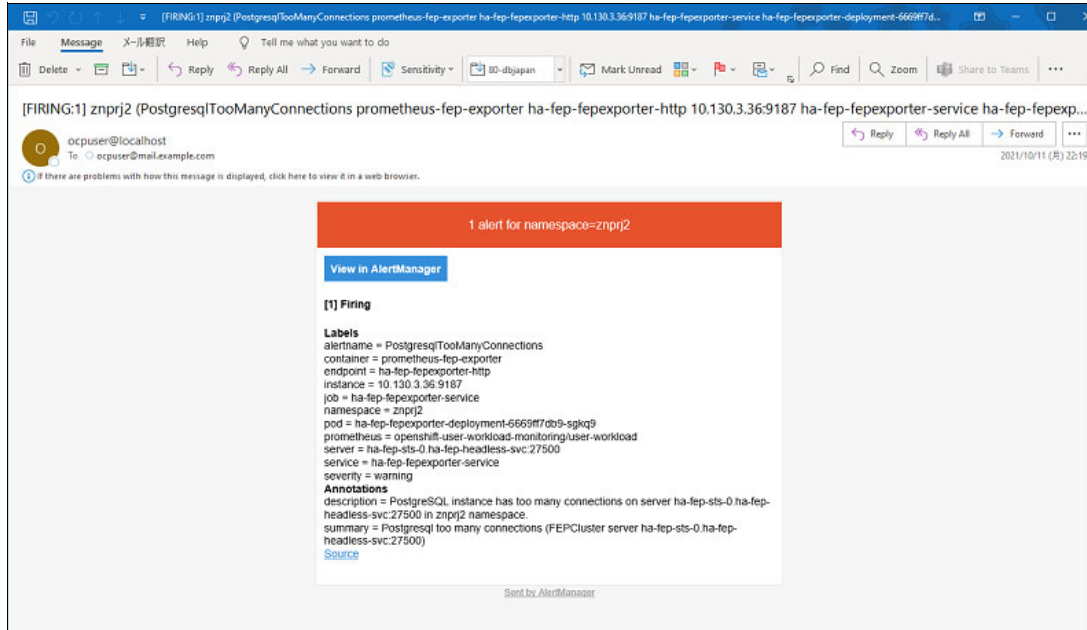


Figure 7-46 Example email notification

Using custom Grafana dashboards

Custom Grafana dashboards that are provided by the open-source community are available. To use them, complete the following steps:

1. Install and set up the community version of Grafana Operator, as described at [Setting up Grafana on IBM LinuxONE](#) that is
2. Click the dashboard icon on the Grafana UI, as shown in Figure 7-47.

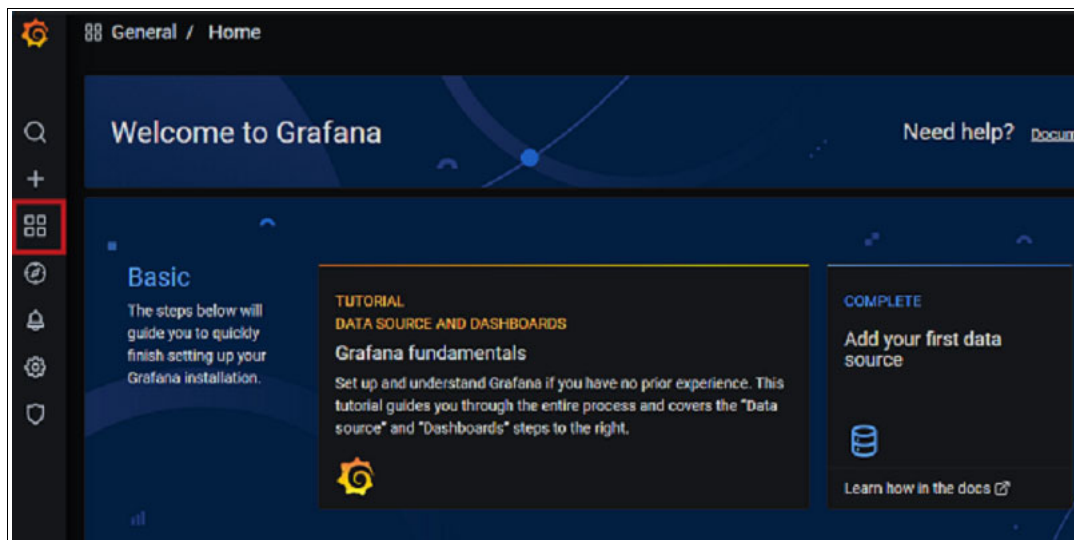


Figure 7-47 Selecting the dashboard icon on the Grafana UI home page

3. Select **Manage**, as shown in Figure 7-48.

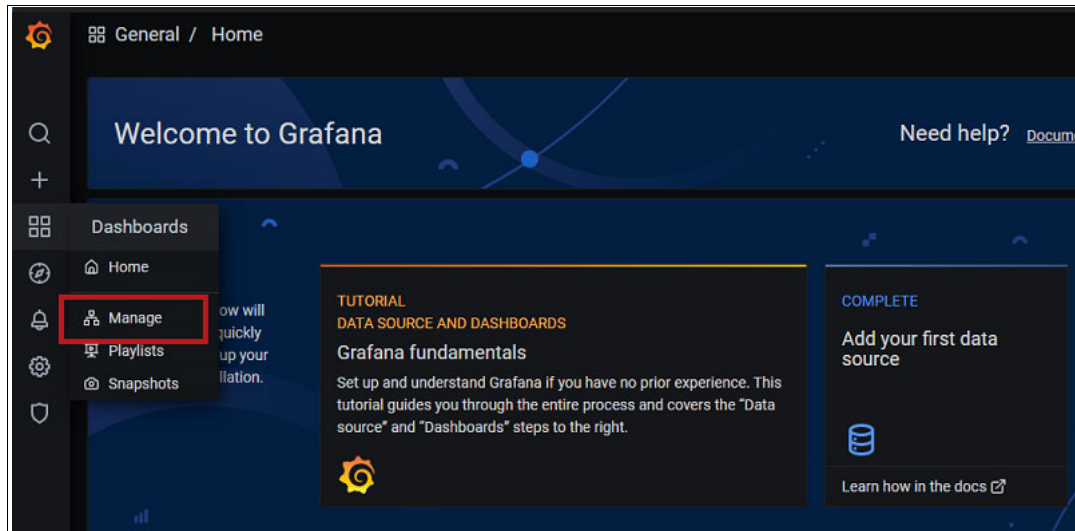


Figure 7-48 Selecting Manage

4. Click **Import**, as shown in Figure 7-49.

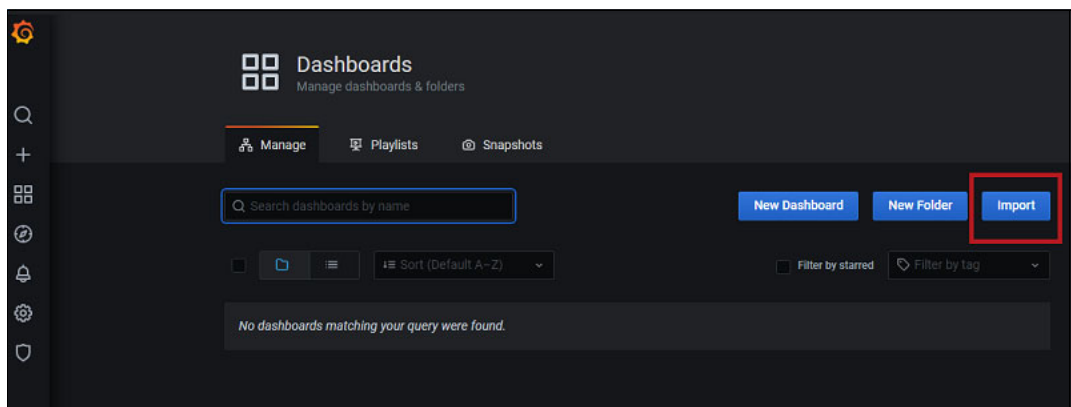


Figure 7-49 Selecting Import in the Grafana dashboard Manage window

5. Click **Upload JSON file**, and upload the JSON file that you downloaded in Step 2 on page 187, as shown in Figure 7-50 on page 189.

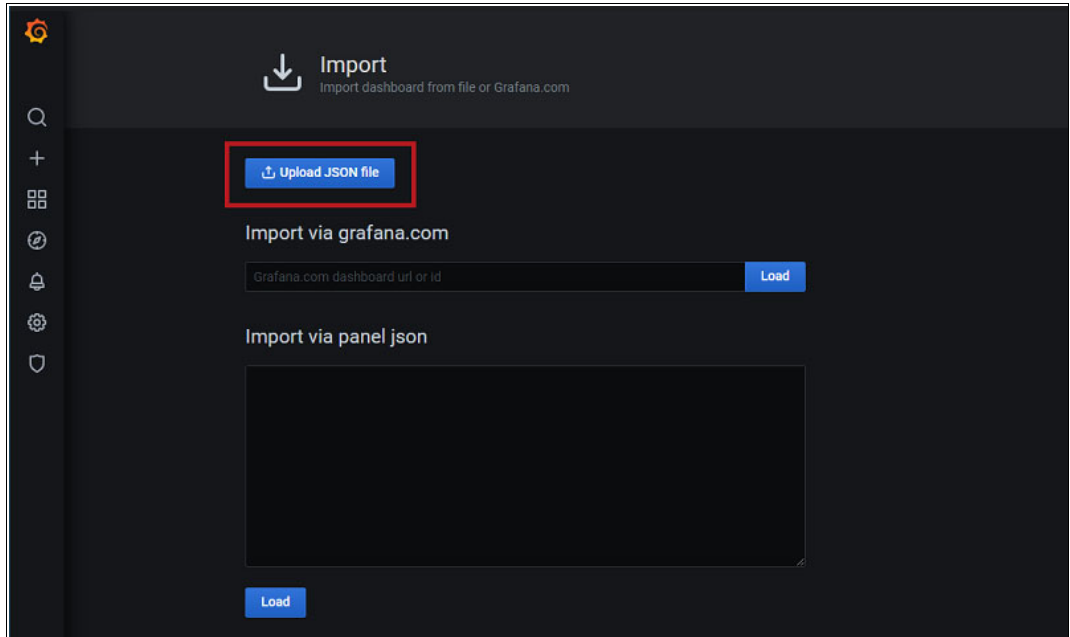


Figure 7-50 Uploading a custom dashboard

6. Click **Import**, as shown in Figure 7-51.

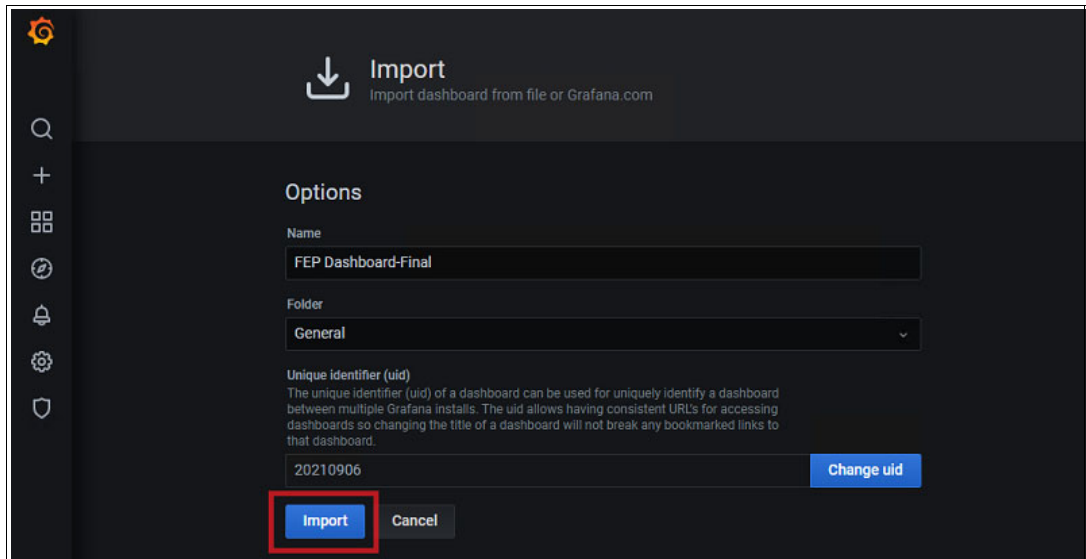


Figure 7-51 Importing a custom dashboard

7. A customized dashboard for FUJITSU Enterprise Postgres appears, as shown in Figure 7-52.

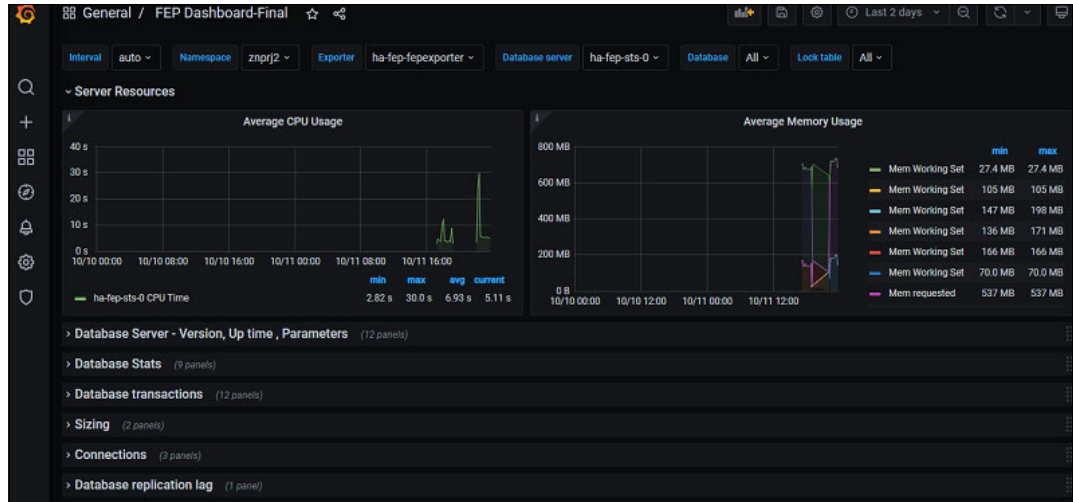


Figure 7-52 FUJITSU Enterprise Postgres custom dashboard

7.5.3 Fluctuation

As businesses grow, more data processing power is vital to support the continuous and stable growth of these businesses. FUJITSU Enterprise Postgres Operator monitors resources and scales data processing power flexibly to ensure that your system maintains optimal performance.

Autoscaling

This section describes the autoscaling feature of FUJITSU Enterprise Postgres Operator.

Database administrators determine the best configurations that meet database performance requirements in the design phase. During operations, expansion by scale-up and scale-out is planned and conducted according to fluctuations in system growth and changes in the business environment.

However, in recent years, in addition to predictable fluctuations, unpredictable fluctuations such as sudden trend changes have also increased. It is important to be ready for unforeseeable changes in referencing business transactions. To this end, it is necessary to flexibly scale data processing power without constantly monitoring fluctuations manually.

FUJITSU Enterprise Postgres Operator constantly monitors changes so that it can flexibly scale data processing power against unexpected fluctuations. Auto scale-out can be set up to scale out replica pods automatically according to the workload to expand system capacity. This feature leverages the high scalability of the IBM LinuxONE platform so that the system obtains performance stability that is resistant to load fluctuations in referencing business transactions.

The following tasks are demonstrated in this section:

- ▶ Settings for automatic scale-out
- ▶ Confirming automatic scale-out

Settings for automatic scale-out

Complete the following steps:

1. Specify the replica service as the connection destination for referencing the business transactions application by using the command that is shown in Example 7-3.

Example 7-3 Connecting to the replica service

```
sh-4.4$ psql -h ha-fep-replica-svc -p 27500 -U postgres publisher
```

2. In the Red Hat OpenShift Console, select **Installed Operators**, and then click **FUJITSU Enterprise Postgres 13 Operator**, as shown in Figure 7-53.

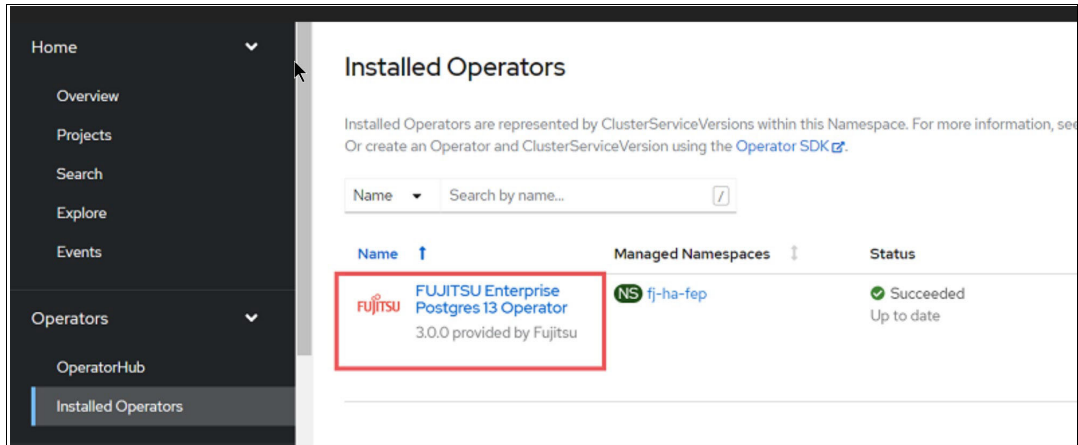


Figure 7-53 Selecting FUJITSU Enterprise Postgres 13 Operator

3. Go to the **FEPCluster** tab and select **ha-fep**, as shown in Figure 7-54.

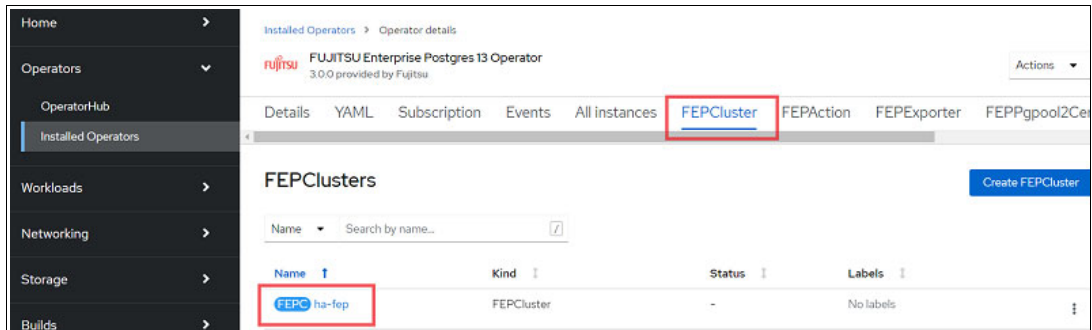


Figure 7-54 Selecting the name of FEPCluster

4. In the FEPCluster Details window, select the **YAML** tab and set the parameters that are shown in Table 7-5, as shown in Figure 7-55. In this example, we set up a policy so that an instance is created whenever the average CPU utilization of the master pod and replica pods in the FEPCluster exceeds 70%.

Table 7-5 Automatic scale-out setting for the FEPCluster CR file

Field	Value	Details
spec: fepChildCrVal: autoscale: scaleout: policy:	cpu_utilization	Scale-out policy. Set to scale-out based on CPU usage.
spec: fepChildCrVal: autoscale: scaleout: threshold:	70	Threshold for the policy. Set 70% CPU utilization as the scale-out threshold.
spec: fepChildCrVal: autoscale: limits: maxReplicas:	4	Maximum number of replicas 0 - 15. In this example, we set it to four replicas.

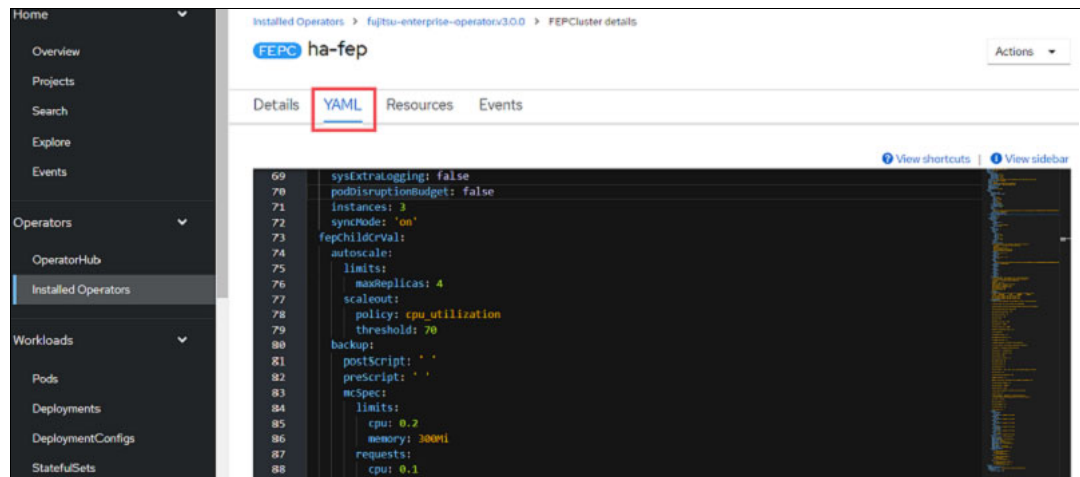


Figure 7-55 Updating the parameters

5. Click **Save** to apply the changes, as shown in Figure 7-56 on page 193.

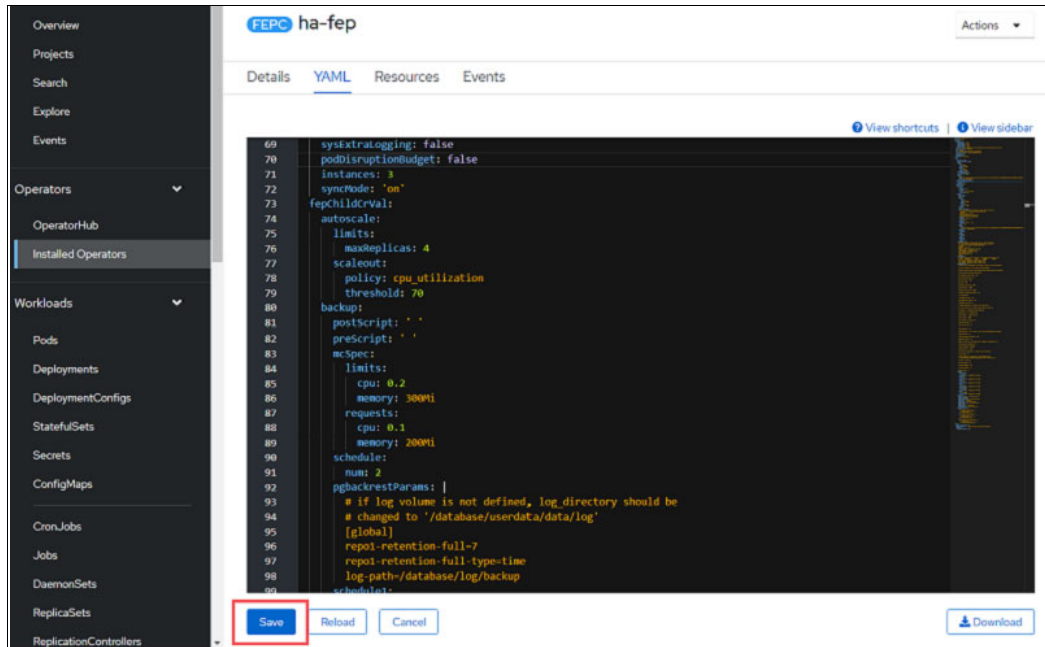


Figure 7-56 Saving the changes

Confirming automatic scale-out

Scale-out is automatically conducted when CPU utilization increases and the scale-out policy conditions are satisfied. To view the list of pods to confirm that the number of replicas increased, select **Workloads** → **Pods**, as shown in Figure 7-57. Performance stability is ensured by expanded capacity even in the case of further increase of the workload.

Name	Status	Ready	Restarts	Owner	Memory	CPU
fep-ansible-operator-cm-85c86d9647-kj94b	Running	1/1	0	fep-ansible-operator-cm-85c86d9647	233.6 MiB	0.286 cores
ha-fep-sts-0	Running	2/2	0	ha-fep-sts	172.6 MiB	0.297 cores
ha-fep-sts-1	Running	2/2	0	ha-fep-sts	146.3 MiB	0.314 cores
ha-fep-sts-2	Running	2/2	0	ha-fep-sts	224.3 MiB	0.005 cores
ha-fep-sts-3	Running	2/2	0	ha-fep-sts	99.8 MiB	0.007 cores
ha-fep-sts-4	Running	2/2	0	ha-fep-sts	46.9 MiB	-

Figure 7-57 Checking the added pods

Note: When using the auto scale-out feature, consider synchronous mode. The default for synchronous mode is on. When the number of replicas increases after scale-out, SQL performance might degrade. Use the auto scale-out feature after validating that the performance remains within the requirements of the system.

If the performance degradation risks the violation of the system requirements, set synchronous mode to off. By turning synchronous mode to off, remember the impacts to the database behavior:

- ▶ When data is updated and the same data is read by another session immediately, the old data might be fetched.
- ▶ When the master database instance fails and a failover to another database instance is performed, updates that were committed on the old master database instance might not be reflected in the new master. After a failover occurs due to a master database failure, investigate records such as the application log to identify the updates that were in progress at the time of failure. Verify that the results of those updates are correctly reflected to all the database instances in the database cluster.

Note: When the workload on the system decreases, users should consider scaling-in to reduce redundant resources. This task is performed manually by editing the FEPCluster CR. For more information, see [FUJITSU Enterprise Postgres 13 for Kubernetes User's Guide](#).

7.5.4 Next steps

Continued operation and fluctuation during the reform of your systems leads to successful completion. The success of one reform is the beginning of the next steps in a customer journey. Based on the foundation that is achieved in a reform, customers can continue to reform with different scales and locations of systems toward further modernization and creation of businesses.

Service expansion leveraging IBM LinuxONE capabilities

This section describes a use case where a successful database in one tenant is expanded to multiple tenants on IBM LinuxONE by leveraging the high consolidation capabilities of IBM LinuxONE.

When expanding tenants on IBM LinuxONE, FUJITSU Enterprise Postgres Operator makes it easy to deploy new tenants.

Even if the database structure is common, the processing capacity that is required for each tenant might be different. With FUJITSU Enterprise Postgres Operator, users can adjust the scale factors (CPU, memory, and disk allocation) of the template that is used in the successful tenant database to quickly deploy a new database with optimal capacity.

To deploy a database in system expansion on IBM LinuxONE, complete the following steps.

Note: The example that is provided in this section assumes that the storage that is used in the new database that will be deployed was pre-provisioned.

1. In the Red Hat OpenShift Console of the existing system, select **Installed Operators** → **FUJITSU Enterprise Postgres 13 Operator** → **FEPCluster** → **ha-fep** and download the CR configuration of FEPCluster on the **YAML** tab, as shown in Figure 7-58.

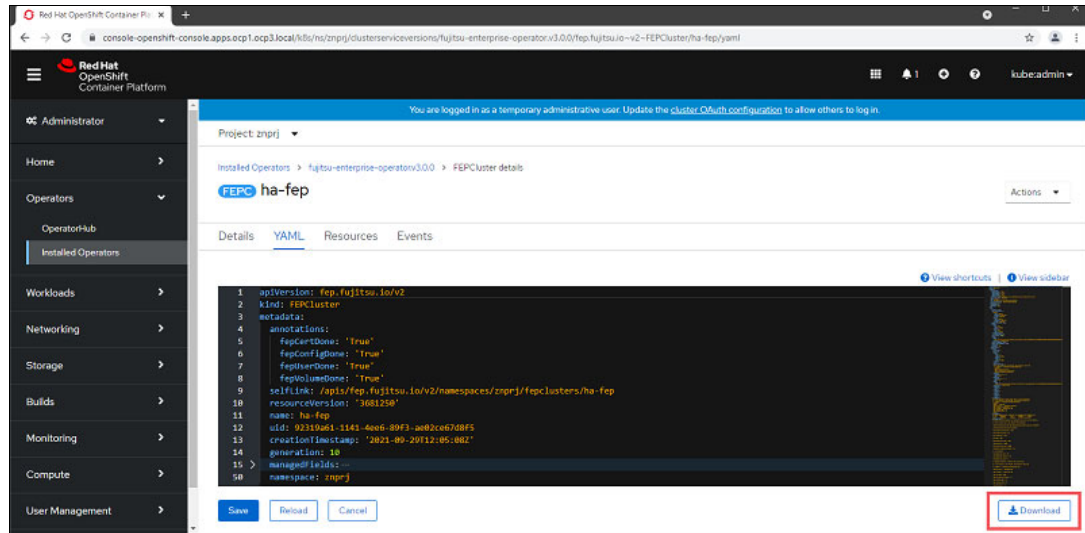


Figure 7-58 Downloading the CR configuration

2. On the Red Hat OpenShift Console of the new system, select **Installed Operators**.
3. Select **FUJITSU Enterprise Postgres 13 Operator**, as shown in Figure 7-59.

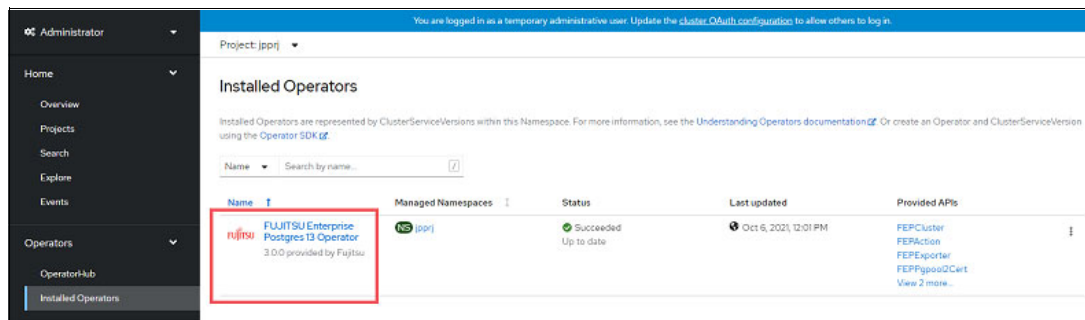


Figure 7-59 Selecting FUJITSU Enterprise Postgres 13 Operator

4. Click **Create Instance**, as shown in Figure 7-60.

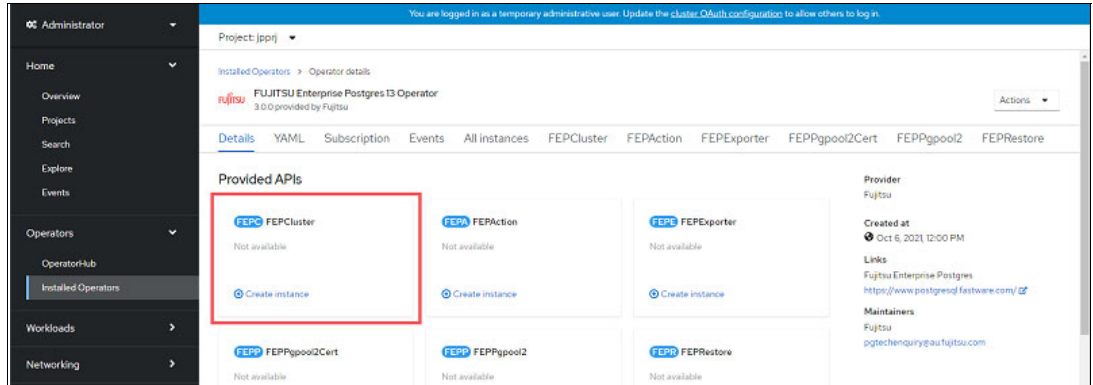


Figure 7-60 Creating a cluster

5. Copy the file that was downloaded on the existing system in step 1 on page 195 to the target location of the system expansion. Open the downloaded file in a text editor, and copy the content of the CR configuration.
6. In the Create FEPCluster window, click the **YAML** tab, and paste the copied contents. Update the value of the CR configuration parameters, as described in Table 7-6. Click **Create** to create a cluster, as shown in Figure 7-61 on page 198.

Table 7-6 FEPCluster CR configuration parameter changes

Field	Value	Details
metadata: name:	ha-fep1	Name of the FUJITSU Enterprise Postgres Cluster. Must be unique within a namespace. Specify any value.
spec: fep: mcSpec:	limits: cpu: 250m memory: 350Mi requests: cpu: 100m memory: 256Mi	Resource allocation to this container. The capacity in this example is assumed to be about 0.5 times that of existing systems.
spec: fepChildCrVal: customPgParams:	shared_buffers = 75 MB	Postgres configuration in postgresql.conf. The capacity in this example is assumed to be about 0.5 times that of existing systems.
spec: fepChildCrVal: customPgHba:	host postgres postgres 10.131.0.213/32 trust	Entries to be inserted into pg_hba.conf. Set the IP address of the trusted client.

Field	Value	Details
spec: fepChildCrVal: sysUsers:	pgAdminPassword: admin-password	Password for the postgres superuser.
	pgdb: mydb	Name of the user database to be created.
	pguser: mydbuser	Name of the user for the user database to be created.
	pgpassword: mydbpassword	Password for pguser.
	pgrepluser: repluser	Name of the replication user. It is used for setting up replication between the primary and replica in FUJITSU Enterprise Postgres Cluster.
	pgreplpassword: repluserpwd	Password for the user to be created for replication.
	tdepassphrase: tde-passphrase	Passphrase for TDE.
spec: fepChildCrVal: storage:	dataVol: size: 2Gi storageClass: gold walVol: size: 1200Mi storageClass: gold tablespaceVol: size: 512Mi storageClass: gold archivewalVol: size: 1Gi storageClass: gold logVol: size: 1Gi storageClass: gold backupVol: size: 2Gi storageClass: gold	Storage allocation to this container. For each volume, set the disk size that you want to allocate and the storageClass name that corresponds to the pre-provisioned storage.

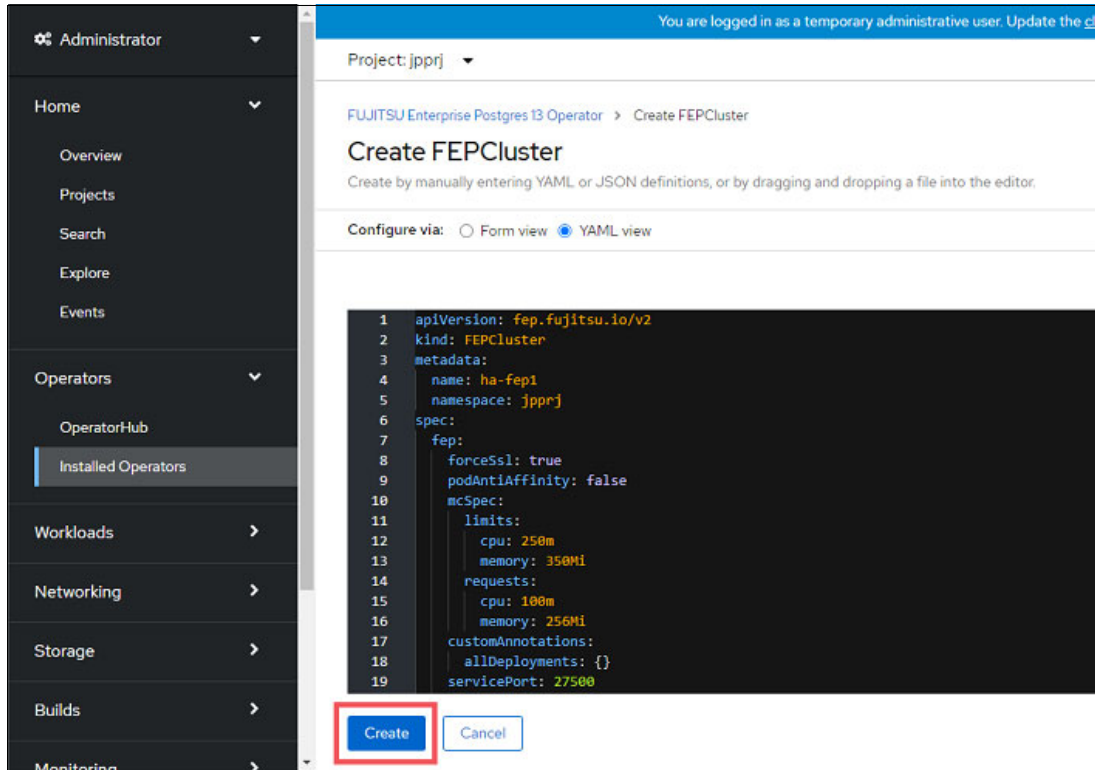


Figure 7-61 Deploying HA cluster

- The HA cluster is deployed, and the deployment status can be checked by selecting **Workloads** → **Pods**. When the cluster is ready, the status is displayed as *Running*, as shown in Figure 7-62.

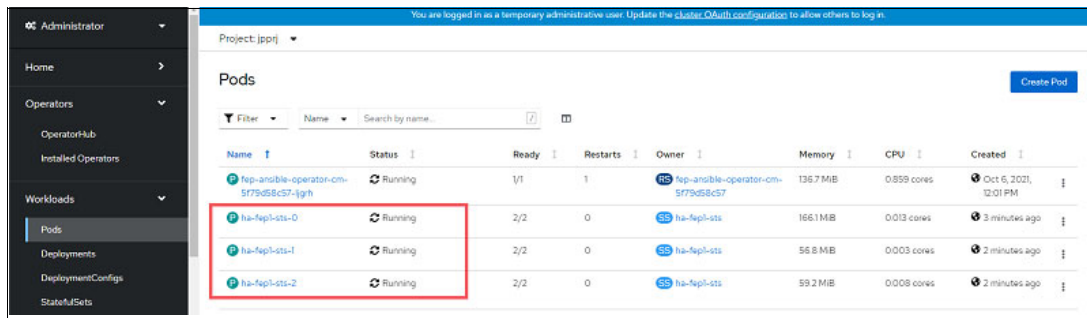


Figure 7-62 Checking the HA cluster deployment

Quick deployment of new databases for business expansion

In this section, we describe a use case where a service that is provided domestically is expanded overseas to launch regional service sites. To launch a new service site, organizations start by creating systems that are based on the database structure and data of the domestic system. To achieve this goal, new databases that are based on the domestic system must be deployed, and parts of the core data must be shared to each region. When expanding to overseas regions, FUJITSU Enterprise Postgres Operator makes it easy to successfully deploy databases and replicate data.

- ▶ Easy deployment of systems

Databases with the same configuration can be easily deployed by using the same template as the domestic database. However, it is likely the case that the new service site at the time of launch does not require as much processing capacity as the domestic system. By adjusting the scale factors (CPU, memory, and disk allocation) of the template, organizations can quickly deploy new systems with optimal capacity.

In this example, the assumed capacity of the new system is approximately 0.5 times the capacity of the central system.

- ▶ Easy and reliable data replication

By using logical replication, selected parts of the core data in the domestic system can be deployed to each region easily and reliably.

A sample deployment scenario that is described in “Deploying databases for regional expansion” on page 199 is shown in Figure 7-63.

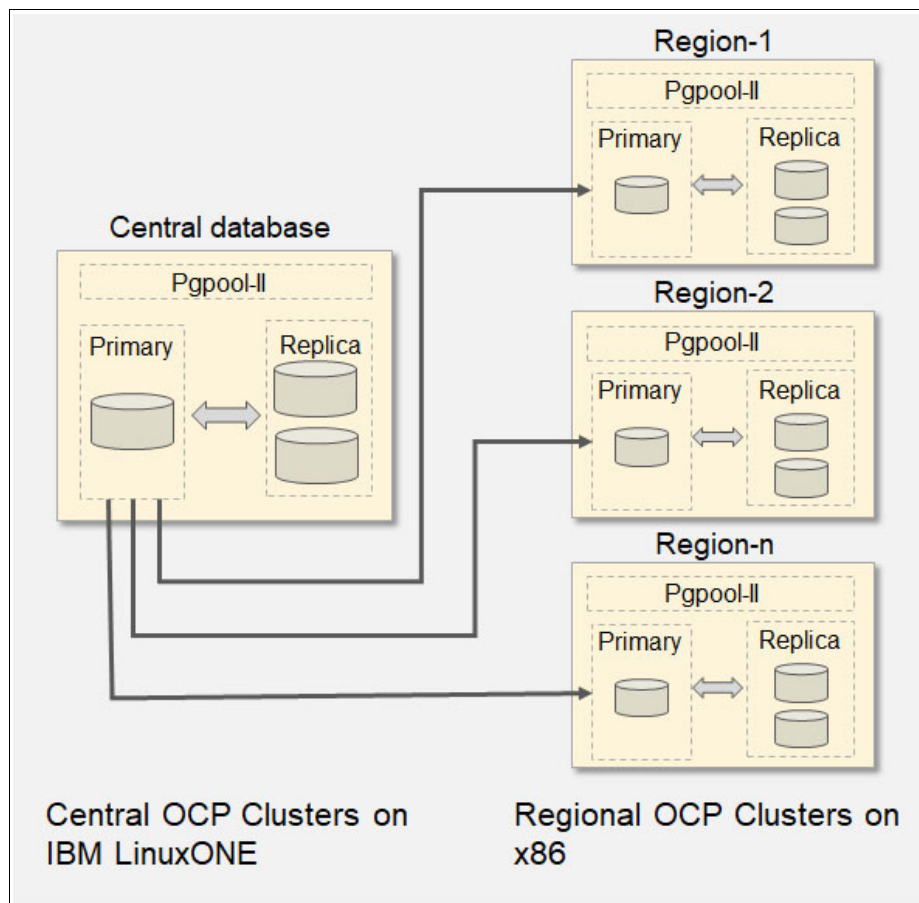


Figure 7-63 Sample expansion in the next step

Deploying databases for regional expansion

By adjusting the scale factors of an existing template that is used in the domestic system, new system deployment is done quickly with optimal capacity, as shown in Figure 7-64.

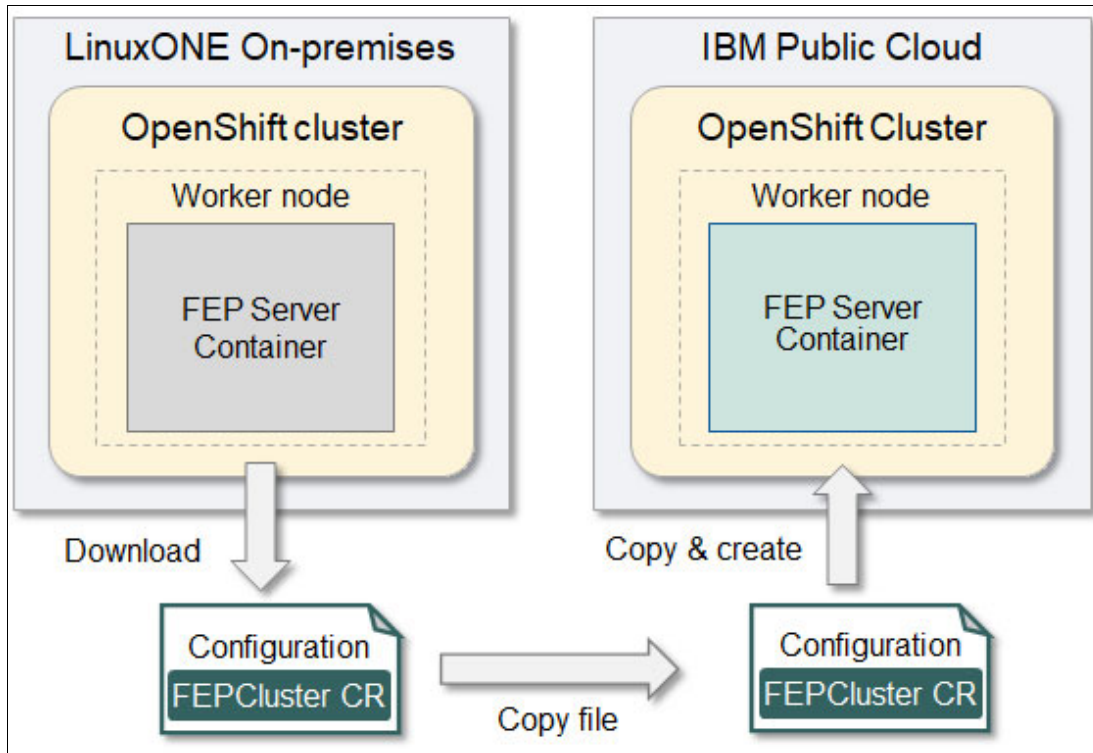


Figure 7-64 New system deployment overview

Note: To understand how to quickly deploy a new database by using an existing template, see , “Quick deployment of new databases for business expansion” on page 198.

Sharing data for regional expansion

It is essential to share data with speed when expanding businesses. After a database deployment, data in new regions must be refreshed by data replication. In doing so, data security is key, so secure communication with mutual authentication such as MTLS is required. Examples of the data that must be refreshed include personal information, such as customer data that is managed in branch offices and employee information for branch offices.

FUJITSU Enterprise Postgres makes it easy and reliable to copy existing data and replicate it in real time with logical replication.

The concept of logical replication is shown in Figure 7-65 on page 201.

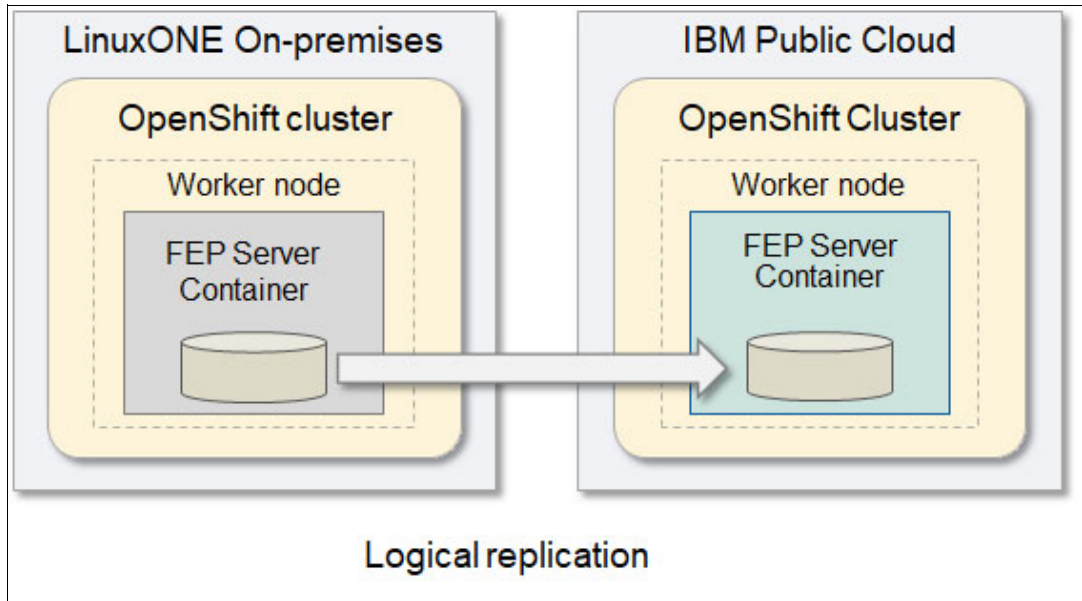


Figure 7-65 Overview of data sharing with logical replication

This use case explains how the database is expanded as a regional service site within one RHOCP Cluster on IBM LinuxONE, as shown in Figure 7-66.

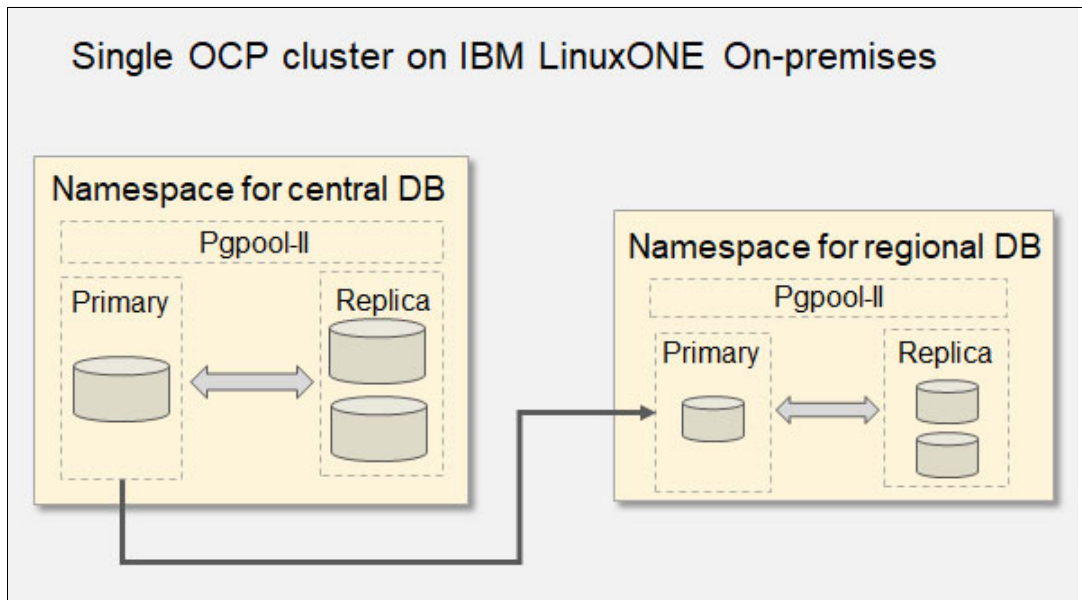


Figure 7-66 Use case in the next step

The following tasks are demonstrated in this section:

- ▶ Publisher settings for logical replication
- ▶ Subscriber settings for logical replication
- ▶ Checking logical replication

Publisher settings for logical replication

In this section, we provide three examples of the steps that are needed for publisher settings for logical replication:

- ▶ Step 1 provides guidance about changing the CR configuration of the FEPCluster.
 - ▶ In logical replication, data is replicated only without table structures, so in step 2 on page 205 we provide guidance about using the `pg_dump` command to dump the schema definition of the publisher cluster’s publisher into a file.
 - ▶ Step 3 on page 206 provides guidance about setting up logical replication on the publisher side.
1. In the Red Hat OpenShift Console on the system that you use as the publisher, select **Installed Operators** → **FUJITSU Enterprise Postgres 13 Operator** → **FEPCluster** → **ha-fep**. Click the **YAML** tab and change the CR configuration of FEPCluster to the following, as shown in Figure 7-67.

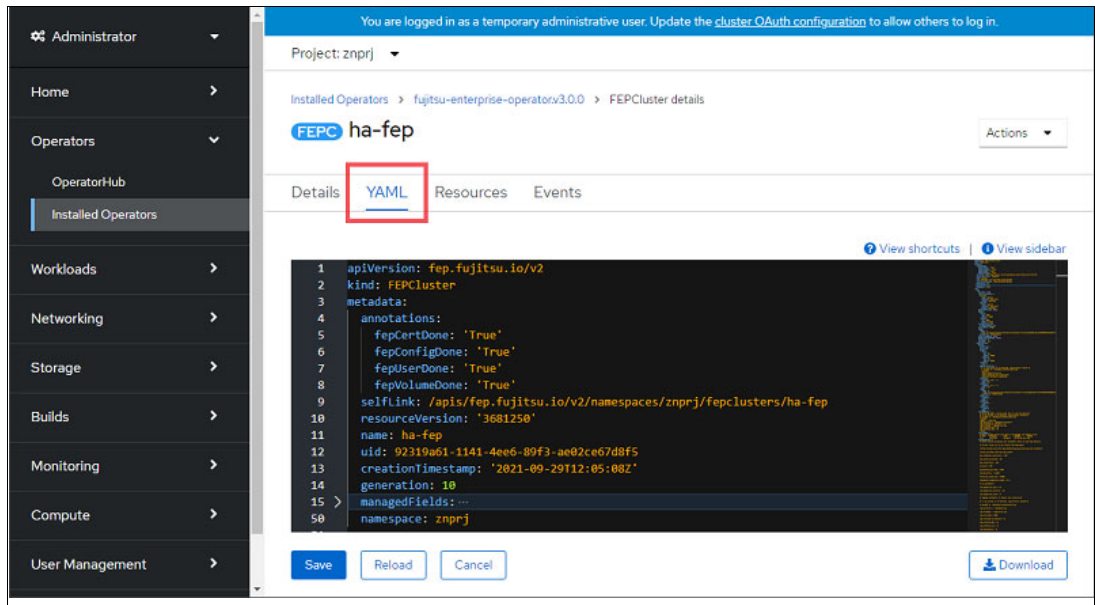


Figure 7-67 Changing the FEPCluster CR configuration of the publisher

- a. Add a replicationSlots section under spec.fep to create replication slots by updating the value of the CR configuration parameter, as shown in Table 7-7.

Table 7-7 FEPCluster CR configuration parameter changes

Field	Value	Details
spec: fep: replicationSlots:	myslot1: type: logical database: publisher plugin: pgoutput myslot2: type: logical database: publisher plugin: pgoutput	List of replication slots that are used for logical replication. database is the name of the database for which you want to set up logical replication. For this example, set publisher to use the database that was created in , “Automatic backup” on page 162. The type and plugin values are fixed, as shown in the Value column on the left.

Note: The slot name that is specified for spec.fep.replicationSlots must be different from the names of the pods in the cluster (in the example in this section, they are ha-fep-sts-0, ha-fep-sts-1, and ha-fep-sts-2). If the name of a pod is also used for the slot name, the replication slot will not be created.

- b. Add a postgres section under spec.fep, as shown in Table 7-8.

Table 7-8 FEPCluster CR configuration parameter changes

Field	Value	Details
spec: fep: postgres:	tls: caName: cacert certificateName: my-fep-cert	caName is the name of the ConfigMap created for the certificate authority (CA). certificateName is the secret that is created by the user that contains the server certificate.

- c. Change the value of wal_level under spec.fepChildCrVal.customPgParams from replica to logical, as shown in Table 7-9.

Table 7-9 FEPCluster CR configuration parameter changes

Field	Value	Details
spec: fepChildCrVal: customPgParams:	wal_level = logical	Postgres configuration in postgresql.conf.

- d. Add the settings to allow replication under spec.fepChildCrVal.customPgHba, as shown in Table 7-10.

Table 7-10 FEPCluster CR configuration parameter changes

Field	Value
spec: fepChildCrVal customPgHba	hostssl all all <SubClusterName>-primary-svc.<SubNamespace>.svc.cluster.local cert

The client must present a certificate, and only certificate authentication is allowed. Replace <SubClusterName> and <SubNamespace> with the appropriate values according to the subscriber FEPCluster.

Table 7-11 FEPAAction CR configuration parameter changes

Field	Value	Details
metadata:	name: ha-fep-action namespace: znprj	name is the object name of FEPAAction CR. Specify a unique value among the same resource type (kind: FEPAAction) within a namespace. namespace is the name of the namespace where the target FEPCluster for the restart is.
spec: fepAction:	args: ha-fep-sts-0 ha-fep-sts-1 ha-fep-sts-2 type: restart	For a restart, the target FUJITSU Enterprise Postgres pod names for the restart must be specified under args. For type, specify restart for a restart.
spec: targetClusterName:	ha-fep	Must specify target a FUJITSU Enterprise Postgres Cluster name within the namespace that is mentioned in metadata.

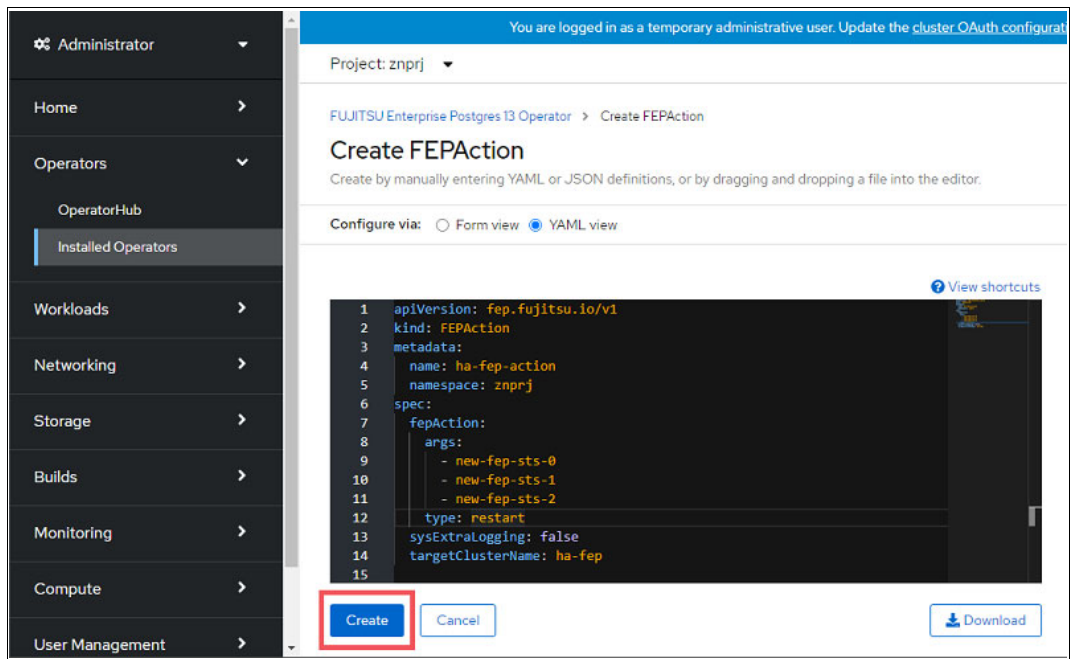


Figure 7-70 Restarting FEPCluster by creating a FEPAAction instance

- Now, a database that is named publisher, which was created in , “Automatic backup” on page 162, with pgbench is used. From the FUJITSU Enterprise Postgres client, use the pg_dump command to dump a schema definition of the publisher cluster’s publisher into a file (Example 7-4).

Example 7-4 The pg_dump command

```
$ pg_dump -h ha-fep-primary-svc -p 27500 -U postgres -d publisher -s -Fp -f /tmp/publisher.schema.dump
```

3. The following steps set up logical replication on the publisher side:
- From the FUJITSU Enterprise Postgres client, connect to publisher, as shown in Example 7-5.

Example 7-5 Connecting to publisher

```
$ psql -h ha-fep-primary-svc -p 27500 -U postgres -d publisher
```

- Create a role that is named `logicalrepluser` and grant the required privileges to this role. The privileges that you grant depend on your requirements, as shown in Example 7-6.

Example 7-6 Creating a role and granting privileges

```
publisher=# CREATE ROLE logicalrepluser WITH REPLICATION LOGIN PASSWORD
'my_password';
publisher=# GRANT ALL PRIVILEGES ON DATABASE publisher TO logicalrepluser;
publisher=# GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO
logicalrepluser;
```

- Create a publication that is named `mypub`. Define the publication for the database and tables that will be replicated, as shown in Example 7-7.

Example 7-7 Creating a publication

```
publisher=# CREATE PUBLICATION mypub FOR TABLE pgbench_branches,
pgbench_accounts, pgbench_tellers;
```

Taking a bank teller system as an example for the regional system, the following three tables are specified in Example 7-7:

- The branch table (`pgbench_branches`)
- Account table (`pgbench_accounts`)
- Bank teller table (`pgbench_tellers`)

All database operations that include **INSERT**, **UPDATE**, and **DELETE** are replicated by default.

- Verify the publication that you created with the query that is shown in Example 7-8. The output of this command is shown in Example 7-9. To verify the publication, use the query that is shown in Example 7-10 on page 207.

Example 7-8 Verifying the publication

```
publisher=# SELECT * FROM pg_publication_tables;
```

Example 7-9 Sample output of the pg_publication tables

pubname	schemaname	tablename
mypub	public	pgbench_tellers
mypub	public	pgbench_accounts
mypub	public	pgbench_branches

(3 rows)

Example 7-10 Verifying the publication with output

```
publisher=# SELECT * FROM pg_publication;
```

oid	pubname	pubowner	puballtables	pubinsert	pubupdate	pubdelete	pubtruncate	pubviaroot
16471	mypub	10	f	t	t	t	t	f

(1 row)

Subscriber settings for logical replication

Complete the following steps:

1. In the cluster that was created in , “Quick deployment of new databases for business expansion” on page 198, in the Red Hat OpenShift Console of the subscriber system, select **Installed Operators** → **FUJITSU Enterprise Postgres 13 Operator** → **FEPCluster** → **ha-fep1**. In the **YAML** tab, as shown in Figure 7-71, add **customCertificates** under **spec.fepChildCrVal**, as shown in Table 7-12 on page 207. Click **Save** to save this change.

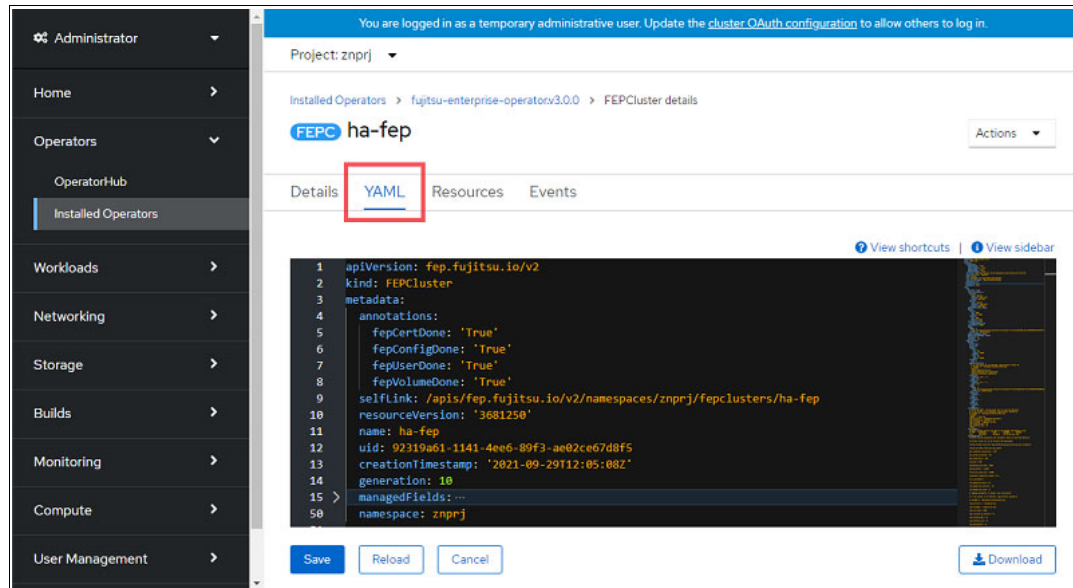


Figure 7-71 Changing the FEPCluster CR configuration

Table 7-12 FEPCluster CR configuration parameter changes

Field	Value	Details
spec: fepChildCrVal:	customCertificates: caName: cacert certificateName: my-logicalrepl-cert username: logicalrepluser	caName is the name of the ConfigMap that was created for the CA. certificateName is the secret that was created by the user that contains the client certificate. username is the name of the role that was created on the publisher cluster for logical replication.

2. From the FUJITSU Enterprise Postgres client on the subscriber side, create a database that is named subscriber on the subscriber side by using the command that is shown in Example 7-11.

Example 7-11 Creating a database

```
$ createdb -h ha-fep1-primary-svc -p 27500 -U postgres subscriber
```

- Transfer the file that was dumped by the FUJITSU Enterprise Postgres client that was created on the publisher side to the FUJITSU Enterprise Postgres client that was created on the subscriber side. Use the `psql` command to point to the transferred file on the subscriber side to create the tables, as shown in Example 7-12.

Example 7-12 Creating tables

```
$ psql -h ha-fep1-primary-svc -p 27500 -U postgres -d subscriber -f
/tmp/publisher.schema.dump
```

- From the FUJITSU Enterprise Postgres client on the subscriber side, connect to the database that you created by using the command that is shown in Example 7-13.

Example 7-13 Connecting to the database

```
$ psql -h ha-fep1-primary-svc -p 27500 -U postgres -d subscriber
```

- Define a subscription by using the command that is shown in Example 7-14. Logical replication starts when this command completes. The existing data in the publication that is targeted in a subscription is copied when replication starts.

Example 7-14 Defining a subscription

```
subscriber=# CREATE SUBSCRIPTION mysub CONNECTION
'host=ha-fep-primary-svc.jprrj.svc.cluster.local port=27500
sslcert=/tmp/custom_certs/logicalrepluser/tls.crt
sslkey=/tmp/custom_certs/logicalrepluser/tls.key
sslrootcert=/tmp/custom_certs/logicalrepluser/ca.crt sslmode=verify-full
password=my_password user=logicalrepluser dbname=publisher' PUBLICATION mypub
WITH (slot_name=myslot1, create_slot=false);
```

- Check the created subscription with the command that is shown in Example 7-15. A sample output is provided in Example 7-16 on page 208.

Example 7-15 Checking the subscription

```
subscriber=# SELECT * FROM pg_subscription;
```

Example 7-16 Sample output of pg_subscription

```
oid | subdbid | subname | subowner | subenabled | subconninfo | subslotname | subsynccommit |
subpublications
-----+-----+-----+-----+-----+-----+-----+-----+-----
 16466 | 16446 | mysub | 10 | t | host= ha-fep-primary-svc.jprrj.svc.cluster.local
port=27500 sslcert=/tmp/custom_certs/logicalrepluser/tls.crt
sslkey=/tmp/custom_certs/logicalrepluser/tls.key sslrootcert=/tmp/custom_certs/logicalrepluser
/ca.crt sslmode=verify-full password=my_password us-er=logicalrepluser dbname=publisher | myslot1 | off
| {mypub}
(1 row)
```

Checking logical replication

The following examples provide the queries and their subsequent output to confirm that data was successfully replicated. The objective of these examples is to **COUNT** the entries on the subscriber side database and verify whether the results match the ones on the publisher side database.

Example 7-17 Checking the logical replication of pgbench_accounts

```
subscriber=# SELECT COUNT(*) FROM public.pgbench_accounts;
```

```
count
-----
 1000000
(1 row)
```

Example 7-18 Checking the logical replication of pgbench_branches

```
subscriber=# SELECT COUNT(*) FROM public.pgbench_branches;
```

```
count
-----
     10
(1 row)
```

Example 7-19 Checking the logical replication of pgbench_tellers

```
subscriber=# SELECT COUNT(*) FROM public.pgbench_tellers;
```

```
count
-----
     100
(1 row)
```

This use case explained how to expand the database within a single RHOCP cluster. It is also possible to expand to other environments, such as:

- ▶ Different RHOCP clusters
- ▶ x86 based IBM Cloud clusters, which have a different CPU architecture than IBM LinuxONE

Note: To set up logical replication, consider the following points:

- ▶ A connection from the subscriber to the publisher must be available.
- ▶ The subscriber must connect by using the hostname that is mentioned in the FUJITSU Enterprise Postgres server certificate of the publisher.

7.6 MongoDB and containers

7.7 Open source databases and containers



8

Database migration

Digital transformation (DX) is a strategy for enabling business innovation by leveraging new technologies. Successful DX requires data modernization that leverages the data that is used in legacy systems while also responding to new technologies. Also, modernizing the current system might increase the cost and effort that is required to apply new technologies, which might block the road for advancing data modernization. Therefore, it is a best practice to consider adopting a database with an open interface to combine new data processing technologies with legacy data and to optimize system management costs and advance DX.

When migrating databases, feasibility must be considered from various perspectives, including availability, security, performance, and cost. With a wealth of knowledge and use cases, we successfully can migrate between heterogeneous databases.

In this chapter, we provide an example of database migration that uses Fujitsu Enterprise Postgres on IBM LinuxONE.

8.1 Increasing demand for database migration

Since the advent of Linux, there are two main trends in system development that leverage open source software:

- ▶ Enhancing mission-critical qualities so that open source software can be used in existing enterprise systems.
- ▶ Diversifying processable data so that open source software can be used for various purposes.

Processable data has become more diverse because it was developed as open source. As a result, many features for business use were developed, and open source became increasingly important, which encouraged enhanced mission-critical quality that is a non-functional requirement and mandatory for business use. This synergy made the acceptance of open source technology in mainstream businesses and open source to become an essential part of enterprise systems.

In the database field, there are two major trends:

- ▶ A database management system (DBMS), which plays a vital role in enterprise systems, has enhanced features such as high availability (HA), security, and performance, such that the usage of a DBMS in mission-critical systems has increased.
- ▶ The rise of various types of NoSQL databases, which enable users to easily manage unstructured data.

In recent years, many organizations started the process of data modernization for DX. As a result, they are increasingly handling data that uses various open source software. In addition, software with sufficient functions for practical use is appearing. Table 8-1 shows examples of open source software that is used for data handling in enterprise systems.

Table 8-1 Examples of open source software for data handling

Classification	Examples of open source software
Data collector	Fluentd
Messaging	Apache Kafka and Apache ZooKeeper
Parallel distributed processing	Apache Hadoop, Apache Spark, and Apache Hive
Data governance	Apache Atlas and Apache Ranger

For this reason, linking a mission-critical DBMS with peripheral data sources and data processing tools to create values is necessary to drive data modernization. To achieve this goal, the DBMS requires an open interface that can work with peripheral data sources and data processing tools.

Oracle databases are established in enterprise systems. Therefore, migration to an open interface database is considered as an option to smoothly promote data modernization.

8.2 Key considerations for database migration

As mentioned in 8.1, “Increasing demand for database migration” on page 212, it is increasingly important to build ecosystems in the DBMS area. Each ecosystem consists of

various data sources that are linked through open interfaces. For this reason, many database engineers who are considering database migration from Oracle Database have considered adopting open source PostgreSQL as their target database because of its open interface and the benefits of reduced licensing fees.

However, database engineers might hesitate to choose open source PostgreSQL because of their concerns about reliability and operations, especially when migrating from enterprise systems with HA and reliability requirements. Additionally, if database specialists in the organization have used only Oracle Database, the skills development for migration is a major concern. The costs of investigating features and training engineers might be significant if there is not sufficient knowledge about a migration to PostgreSQL.

This section describes how to solve these challenges with knowledge that is based on numerous migrations that Fujitsu has carried out, with two viewpoints to be considered in database migration:

► Product

The combination of IBM LinuxONE and Fujitsu Enterprise Postgres enables database engineers to build highly reliable data processing systems that meet the essential requirements of enterprise systems, which include HA architectures with FIPS 140-2 Level 4 security.

The following two sections highlight key considerations for migration:

- Section 8.2.1, “Business continuity” on page 215

This section introduces the features that Fujitsu Enterprise Postgres provides for business continuity and the HA features that are further strengthened by IBM LinuxONE.

- Section 8.2.2, “Mitigating security threats” on page 217

This section introduces the enhanced security features of Fujitsu Enterprise Postgres and the data encryption features that are available in combination with IBM LinuxONE.

Figure 8-1 shows one of the HA, highly secure architecture implementations of Fujitsu Enterprise Postgres on LinuxONE.

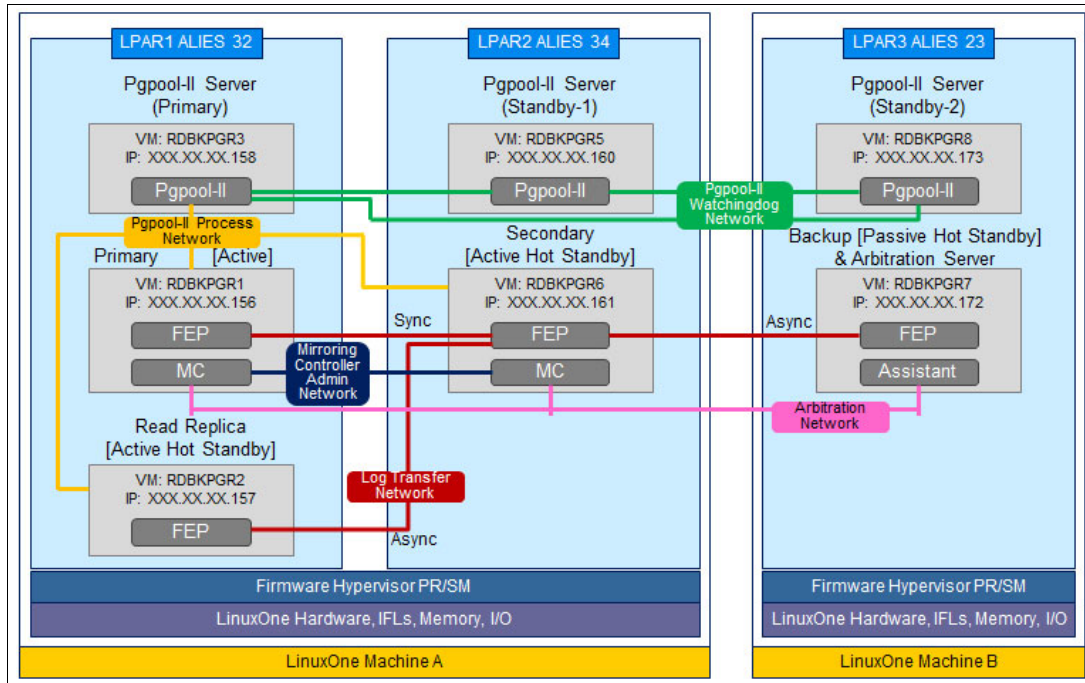


Figure 8-1 Database configuration for an enterprise system with IBM LinuxONE and FUJITSU Enterprise Postgres

Note: For more information about implementing the architecture that is shown in Figure 8-1, see Chapter 5 “High availability and high reliability architectures” and Chapter 6 “Connection pooling and load balancing with Pgpool-II” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

► Knowledge based on migration experience

A Fujitsu highly experienced team can assist organizations with decreasing the complexities and increasing the success of migration projects from Oracle Database to FUJITSU Enterprise Postgres. Fujitsu Professional Services enables database engineers to greatly reduce the costs of investigation and training for database migration.

Leveraging Fujitsu Professional Services helps to achieve two key areas in migration: performance tuning and cost optimization, which are described in the following sections.

– Section 8.2.3, “SQL performance tuning” on page 221

This section provides an introduction to SQL tuning to resolve PostgreSQL performance challenges that organizations often face during enterprise system migration.

– Section 8.2.4, “Optimizing database migration costs” on page 229

This section provides an introduction to a feature of Fujitsu Enterprise Postgres to reduce the memory usage for database systems. Fujitsu services that are available for database migration are also introduced in this section.

Note: To learn more about SQL performance tuning or other migration works, contact Fujitsu Professional Services, found at:

<https://www.postgresql.fastware.com/contact>

Fujitsu services are available for proof-of-concept (POC) assistance and for migration in production environments for smoother delivery of migration projects.

8.2.1 Business continuity

Business continuity is key for enterprise systems. Many enterprise systems use two-node HA architectures to achieve requirements for business continuity. In fact, existing enterprise systems often use Oracle Real Application Cluster (Oracle RAC) as their HA architecture. Fujitsu Enterprise Postgres allows two-node HA architectures, which meet the same requirements with different mechanisms and technical elements. Therefore, many systems that are configured with Oracle RAC can be migrated to FUJITSU Enterprise Postgres.

Comparing business continuity between Oracle RAC and Fujitsu Enterprise Postgres

Oracle RAC and Fujitsu Enterprise Postgres use different ways to configure HA architectures, for example, they use different technical elements such as storage allocation or data synchronizing methods.

Oracle RAC is a clustered architecture where multiple nodes make up a single database on shared storage such as SAN or NAS. Oracle RAC supports an active/active configuration and load balancing across nodes. Fujitsu Enterprise Postgres supports active/standby cluster as a HA architecture. One of the nodes is used for read/write workloads, and the other node is used for read-only workloads. Load balancing can be implemented by using a connection-pooling software that is known as *Pgpool-II*.

Figure 8-2 shows the HA architectures for Oracle RAC and FUJITSU Enterprise Postgres.

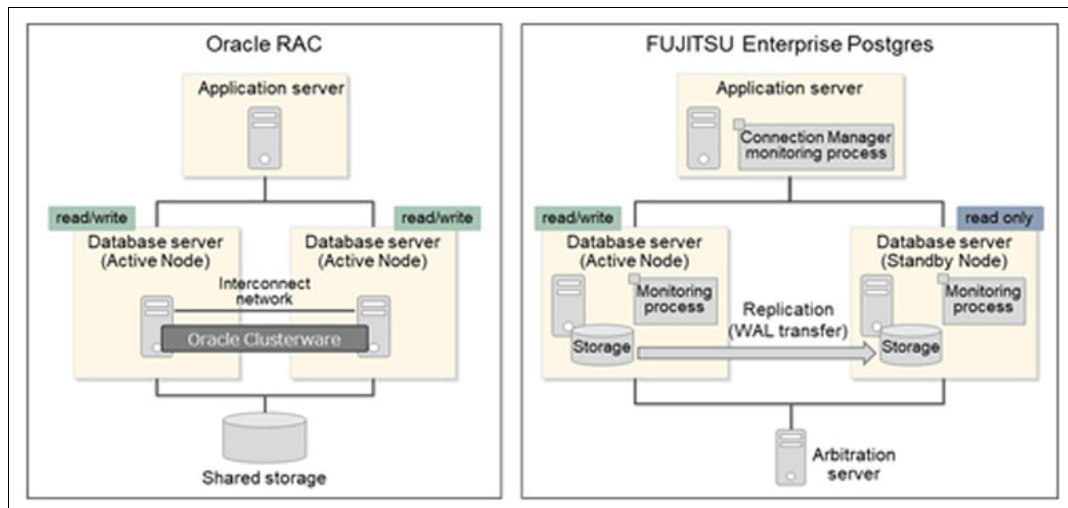


Figure 8-2 HA architectures for Oracle RAC and FUJITSU Enterprise Postgres

Despite the differences in storage and synchronization, similarities can be seen in Figure 8-2 on page 215. Both databases require appropriate computer resources for each node to configure HA architectures. If a node fails, the HA mode of operation is degraded to single-node operations in both database products, which means that the system should be designed for single-node operations in the case of node failures. For example, in a node failure to maintain the same performance as before the failure, each node requires twice the CPU resources at peak hours.

Fujitsu Enterprise Postgres follows a similar approach to Oracle RAC to achieve the business continuity that is required for enterprise systems.

Enhanced features for enterprise-level business continuity

Fujitsu Enterprise Postgres provides two enterprise features as standard for ensuring more than five nines availability: *Database Multiplexing* and *Connection Manager*. These features allow organizations to build HA systems with ease.

► Database Multiplexing

When Database Multiplexing is used, each node of the HA system operates synchronously and autonomously, and data is always kept consistent between the two nodes. In addition, Fujitsu Enterprise Postgres continually monitors the system looking for any issues. Even when monitoring does not work because of network errors, issues are detected by monitoring through the arbitration server so that a database server can be switched seamlessly and quickly to an alternative database server if an abnormality is detected.

This feature does not require shared storage or dedicated clustering software. Therefore, database systems can be deployed on any platform, including cloud and virtualized environments. Figure 8-3 shows how Database Multiplexing works.

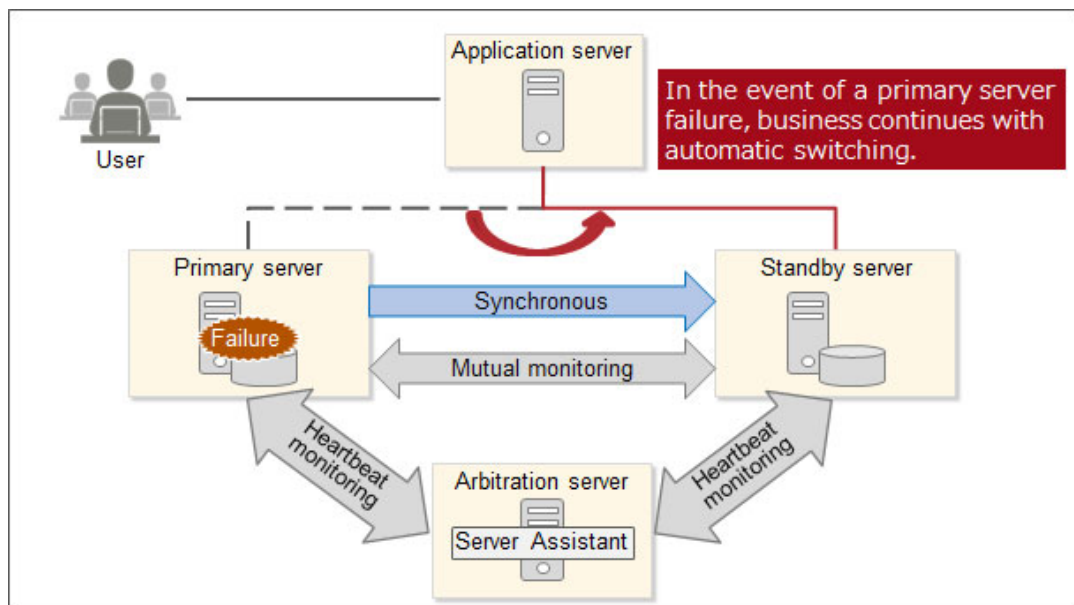


Figure 8-3 Fujitsu Enterprise Postgres Database Multiplexing

Note: For more information about Database Multiplexing, see 2.1 “Availability and reliability features” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

► Connection Manager

Connection Manager allows quick detection of network errors and server outages with no response for extended periods. This task is done by using mutual heartbeat monitoring between the client and the database servers. When an abnormality is detected, the database server is notified in the form of a forced collection of SQL connections with the client, and the client is notified by an error event through SQL connection. Because Connection Manager determines which database server to connect to, applications need to retry only the SQL that returned an error, which ensures business restarts with minimal downtime. Figure 8-4 shows the Connection Manager processes.

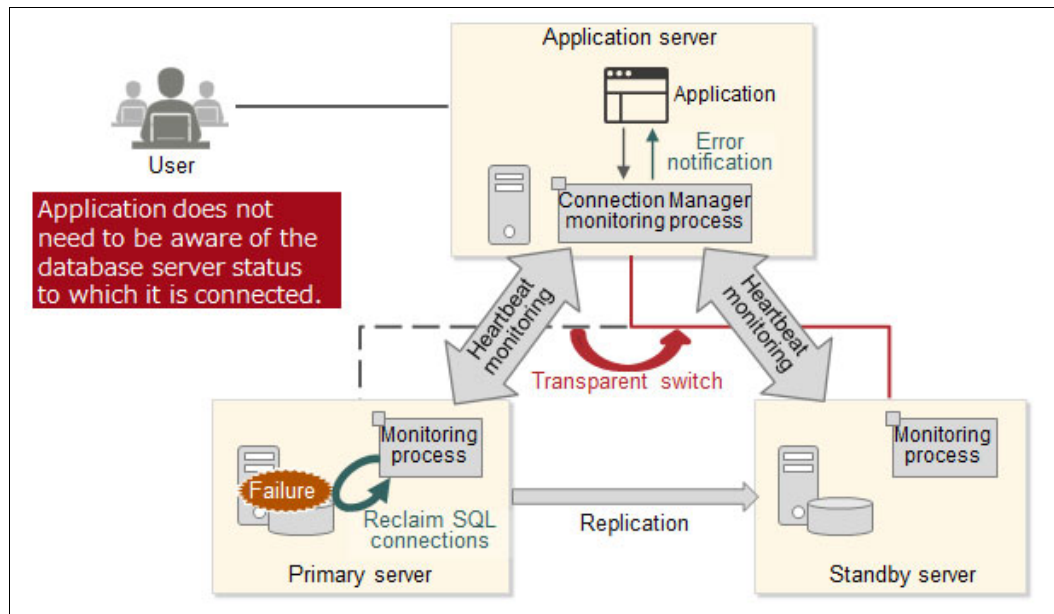


Figure 8-4 Connection Manager

Examples of the HA architecture deployment scenarios

To understand how to build and configure HA systems on Fujitsu Enterprise Postgres on IBM LinuxONE, see the following chapters or sections in [Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE, SG24-8499](#):

- Chapter 5, “High availability and high reliability architectures”
- Chapter 6, “Connection pooling and load balancing with Pgpool-II”
- Section 7.2, “Connection Manager”

8.2.2 Mitigating security threats

Recently, cyberattacks and unauthorized access into systems have become more aggressive, sophisticated, and complicated. Security risks such as information leakage have increased. Preventive measures against these security threats should be considered at each layer: human, physical, application, and database.

To keep data secure, database security requires comprehensive management of the following three aspects of information assets:

- ▶ **Confidentiality:** Access to information is managed to prevent leakage of information outside of the company. Access control or prevention of information leakage must be considered.
- ▶ **Integrity:** Information integrity is ensured, and information cannot be changed or misused by an unauthorized party. Prevention or detection of data falsification is required.
- ▶ **Availability:** Information is accessible to authorized users anytime. Power supplies must be ensured and a redundant system configuration should be set.

In addition, databases that are used in enterprise systems often require the following two security features:

- ▶ *Advanced encryption* to prevent critical data exposure
- ▶ *An audit feature* for early detection of unauthorized access

Fujitsu Enterprise Postgres extends PostgreSQL for use in an enterprise system and provides these two security features so that organizations can migrate from commercial enterprise systems such as Oracle Database without compromising their security.

Advanced encryption to prevent critical data exposure

To protect critical database data, organizations must prevent theft of stored and backup data from the database by any means. If it is stolen, the data must be indecipherable.

It is also necessary to prevent critical data from being exposed when SQL fetches the data. Critical data should be obfuscated so that unauthorized users who do not have the correct permissions cannot see that sensitive information.

Fujitsu Enterprise Postgres fulfills these security requirements through two enterprise features: *Transparent Data Encryption (TDE)* and *Data Masking*.

▶ TDE

The key to data encryption is how seamless it is to encrypt data and how secure the encryption key management is:

– Seamless data encryption

Storage data and backup data can be transparently encrypted without application modification:

- The encryption algorithm does not change the size of the object that is encrypted, so there is no storage overhead.
- The encryption level fulfills the requirements for the Payment Card Industry Data Security Standard (PCI-DSS) and allows confidential information such as credit card numbers to be made unrecognizable on disk.
- CP Assist for Cryptographic Functions (CPACF) in the IBM Z processor is used to minimize the encryption and decryption overhead.

Figure 8-5 on page 219 shows the Fujitsu Enterprise Postgres Transparent Data Encryption processes.

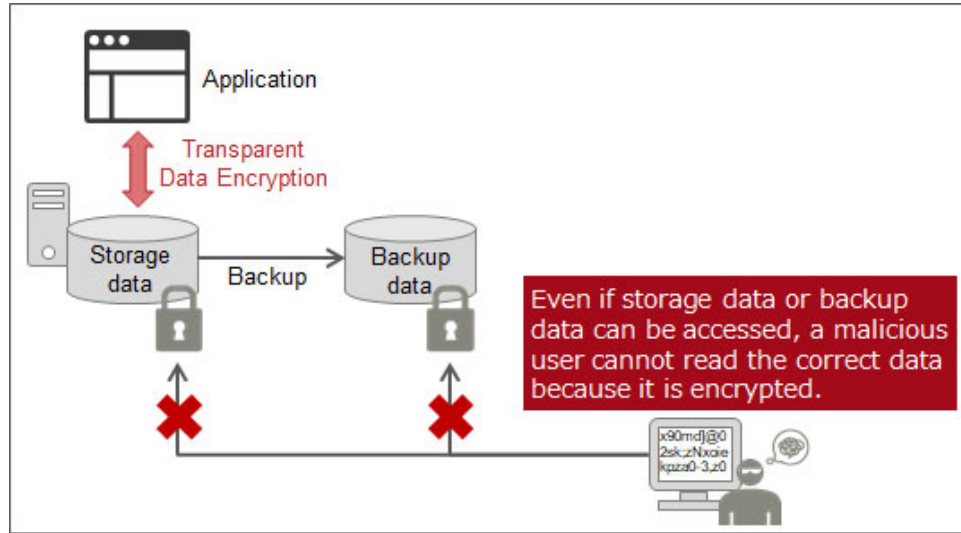


Figure 8-5 Fujitsu Enterprise Postgres Transparent Data Encryption

- Secured encryption key management

Keystore management is essential for data encryption. Fujitsu Enterprise Postgres provides security-enhanced management of keystores with IBM LinuxONE CryptoCard Hardware Security Module (HSM). Fujitsu Enterprise Postgres is integrated to use CryptoCard based encryption and keystore management, as shown in Figure 8-6.

Fujitsu Enterprise Postgres supports both file-based keystore management and hardware-based keystore management. In file-based management, keystore files are managed in folders. In hardware-based management, the keystore is managed by an IBM LinuxONE CryptoCard HSM. The CryptoCard is an HSM that protects digital keys by storing them in separate hardware that is FIPS 140-2 Level 4 certified. Using this security enhanced, hardware-based keystore management, organizations can safely migrate from Oracle Database servers that use a hardware-based keystore.

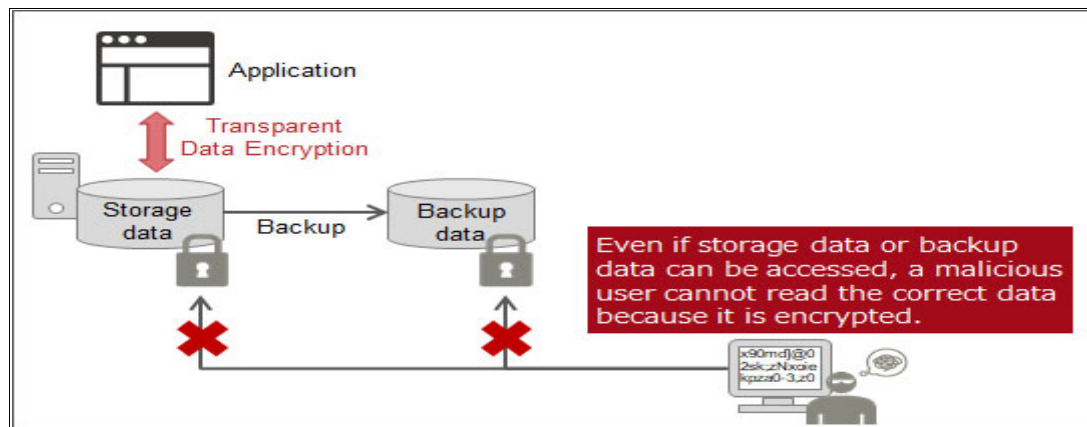


Figure 8-6 File-based and hardware-based keystore management options in Fujitsu TDE

► Data Masking

With Fujitsu Data Masking, you can obfuscate specific columns or part of the columns of tables that store sensitive data while still maintaining the usability of the data. The data that is returned for queries to the application is changed so that users can reference the data without exposing the data. For example, for a query of a credit card number, all the digits except the last 4 digits of the credit card number can be changed to "*" so that the credit card number can be referenced.

The benefit of using Fujitsu Data Masking is that SQL modification in existing application is not required to obfuscate sensitive data. Query results are masked according to the configured data masking policy. Database administrators can specify the masking target, masking type, masking condition, and masking format in a masking policy.

Figure 8-7 shows a Data Masking use case for test data management.

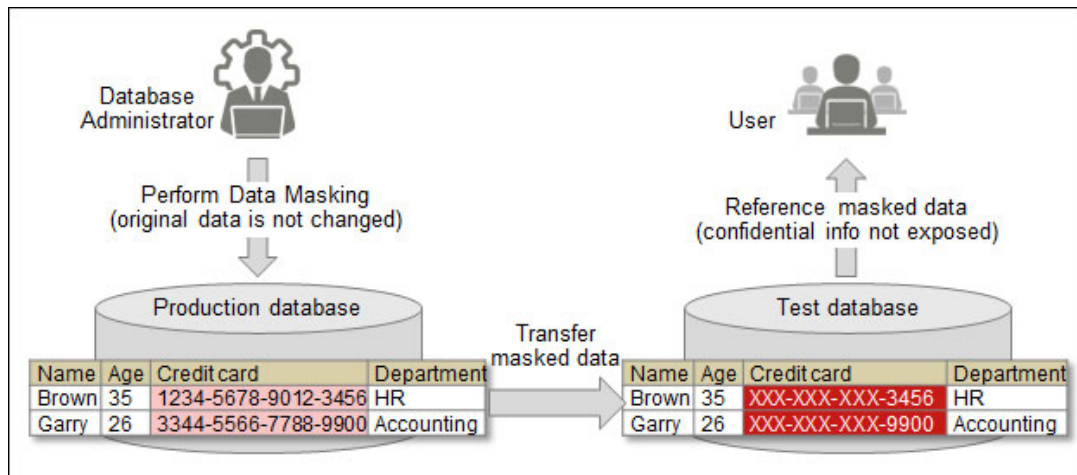


Figure 8-7 Test data management use case for data masking

Audit Logging feature for early detection of unauthenticated access

The Audit Logging feature enables organizations to log details of database access. This feature can be used to prevent security threats such as unauthenticated access or system authority abuse of databases. For example, in unusual database access, easy detection is possible so that organizations can investigate and act as soon as possible. Fujitsu Enterprise Postgres provides the Audit Logging feature to satisfy this security requirement.

By using Fujitsu Audit Logging, the database access related logs can be retrieved in audit logs. Actions by the administrators and users that are related to the databases are output to the audit log.

The benefit of using Fujitsu Enterprise Postgres Audit Logging is that audit logs can be output to a dedicated log file that is separate from the server log, which enables efficient and accurate log monitoring. Also, the audit log is written asynchronously, so there is no performance impact for logging.

Figure 8-8 on page 221 shows an example of the Audit Logging process.

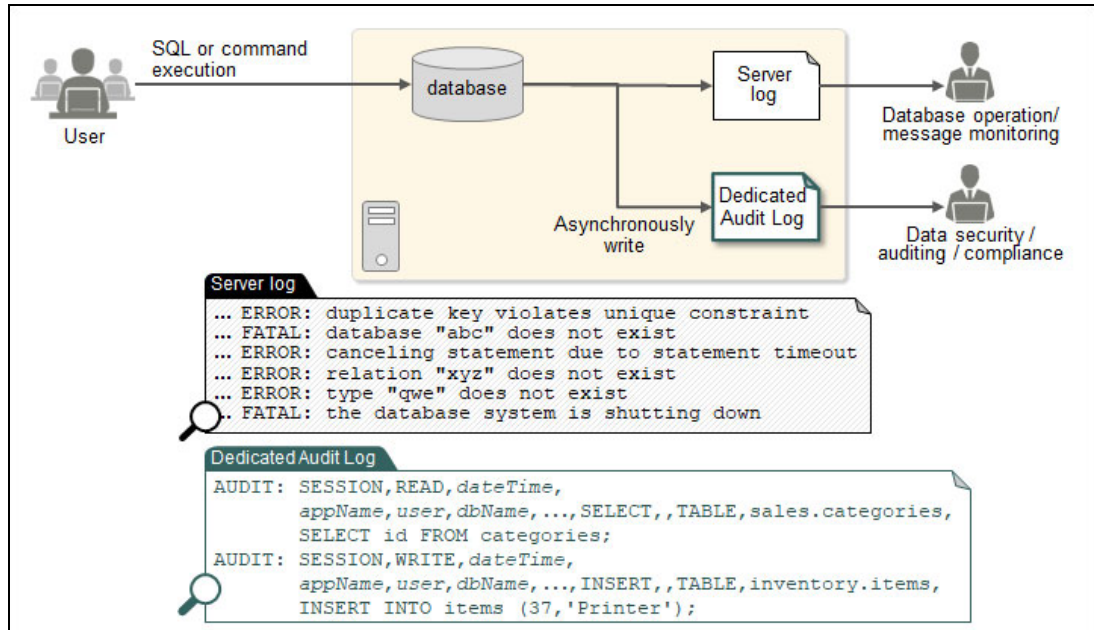


Figure 8-8 Fujitsu Enterprise Postgres Audit Logging

Examples of using security features

For more information about to configure security features on Fujitsu Enterprise Postgres on IBM LinuxONE, see Chapter 4, “Data security with TDE, Data Masking, and Audit Logs”, in [Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE, SG24-8499](#).

8.2.3 SQL performance tuning

When migrating from Oracle Database to FUJITSU Enterprise Postgres, organizations might prefer to use the same SQL that was used in Oracle Database. Although the SQL runs successfully on PostgreSQL, it often fails to meet performance requirements because each DBMS has different performance characteristics. Therefore, it is important to verify performance when migrating databases. If there are some issues with performance, SQL performance tuning is the key to solving these issues and achieving your tuning goal. As shown in Table 8-25 on page 260, using partition pruning is one of the effective ways of performance tuning. Partition pruning improves query performance by localizing I/O processing.

This section describes the following topics:

- ▶ Table partitioning and the effects of partition pruning
- ▶ SQL tuning cases to enable partition pruning

Table partitioning and the effects of partition pruning

Partitioning is a feature that is used to split a single large table into multiple partitions by using partition keys. Example 8-1 shows table *t1*, which is created with 10 partitions, *t1_0* - *t1_9*. This example uses the **PARTITION BY LIST** option by using CHAR data type column as the partitioning criteria.

Example 8-1 Creating partitions

```
[fsepuser@rdbkpg1 ~]$ psql -p 27500 -d postgres
psql (13.1)
Type "help" for help.
postgres=# CREATE TABLE t1 (id char(3), value int) PARTITION BY LIST (id);
CREATE TABLE
postgres=# CREATE TABLE t1_0 PARTITION OF t1 FOR VALUES IN ('ID0');
CREATE TABLE
postgres=# CREATE TABLE t1_1 PARTITION OF t1 FOR VALUES IN ('ID1');
CREATE TABLE
postgres=# CREATE TABLE t1_2 PARTITION OF t1 FOR VALUES IN ('ID2');
CREATE TABLE
postgres=# CREATE TABLE t1_3 PARTITION OF t1 FOR VALUES IN ('ID3');
CREATE TABLE
postgres=# CREATE TABLE t1_4 PARTITION OF t1 FOR VALUES IN ('ID4');
CREATE TABLE
postgres=# CREATE TABLE t1_5 PARTITION OF t1 FOR VALUES IN ('ID5');
CREATE TABLE
postgres=# CREATE TABLE t1_6 PARTITION OF t1 FOR VALUES IN ('ID6');
CREATE TABLE
postgres=# CREATE TABLE t1_7 PARTITION OF t1 FOR VALUES IN ('ID7');
CREATE TABLE
postgres=# CREATE TABLE t1_8 PARTITION OF t1 FOR VALUES IN ('ID8');
CREATE TABLE
postgres=# CREATE TABLE t1_9 PARTITION OF t1 FOR VALUES IN ('ID9');
CREATE TABLE
postgres=# INSERT INTO t1 SELECT 'ID' || (i / 1000000), i from generate_series(0,
9999999) as i;
INSERT 0 10000000
```

One of the benefits of partitioning is enabling the use of partition pruning. If partition pruning is enabled, only the partitions that match SQL search conditions are accessed to read data, which improves SQL query performance compared to accessing all partitions.

Example 8-2 shows how partition pruning affects the query plan and improves performance. It is an SQL query that retrieves all rows where the ID column is ID1.

Example 8-2 SQL query retrieving all rows with a specific column value

```
SELECT * FROM t1 WHERE id = 'ID1';
```

The query plan for the SQL query that is shown in Example 8-2 is shown in Example 8-3 on page 223.

Example 8-3 Query plan with partition pruning

```
[fsepuser@rdbkpgr1 ~]$ psql -p 27500 -d postgres -c "EXPLAIN ANALYZE SELECT * FROM t1 WHERE id = 'ID1';"
```

QUERY PLAN

```
Seq Scan on t1_1 t1 (cost=0.00..16925.00 rows=1000000 width=8) (actual
time=0.054..502.058 rows=1000000 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Planning Time: 0.329 ms
  Execution Time: 932.492 ms
(4 rows)
```

In this query plan, only `t1_1` is used, and all other partitions such as `t1_2` are not used because the PostgreSQL query engine creates an access plan to fetch data only from partition `t1_1`. This process is known as partition pruning, where data is extracted only from those partitions that match the partitioning key criteria.

Next, we compare the query plans that are created for tables with partition pruning and without partition pruning.

For comparison, partition pruning can be disabled. To verify the effects of partition pruning, the query plan for running the same SQL query without partition pruning is shown in Example 8-4.

Example 8-4 Query plan without partition pruning

```
[fsepuser@rdbkpgr1 ~]$ psql -p 27500 -d postgres
psql (13.1)
Type "help" for help.
postgres=# SET enable_partition_pruning = off;
SET
postgres=# EXPLAIN ANALYZE SELECT * FROM t1 WHERE id = 'ID1';
```

QUERY PLAN

```
Append (cost=0.00..174250.05 rows=1000009 width=8) (actual time=81.068..1852.613
rows=1000000 loops=1)
-> Seq Scan on t1_0 t1_1 (cost=0.00..16925.00 rows=1 width=8) (actual
time=80.999..81.000 rows=0 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Rows Removed by Filter: 1000000
-> Seq Scan on t1_1 t1_2 (cost=0.00..16925.00 rows=1000000 width=8) (actual
time=0.061..469.408 rows=1000000 loops=1)
  Filter: (id = 'ID1'::bpchar)
-> Seq Scan on t1_2 t1_3 (cost=0.00..16925.00 rows=1 width=8) (actual
time=61.166..61.167 rows=0 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Rows Removed by Filter: 1000000
-> Seq Scan on t1_3 t1_4 (cost=0.00..16925.00 rows=1 width=8) (actual
time=60.914..60.915 rows=0 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Rows Removed by Filter: 1000000
-> Seq Scan on t1_4 t1_5 (cost=0.00..16925.00 rows=1 width=8) (actual
time=61.962..61.963 rows=0 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Rows Removed by Filter: 1000000
```

```

-> Seq Scan on t1_5 t1_6 (cost=0.00..16925.00 rows=1 width=8) (actual
time=58.657..58.658 rows=0 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Rows Removed by Filter: 1000000
-> Seq Scan on t1_6 t1_7 (cost=0.00..16925.00 rows=1 width=8) (actual
time=58.656..58.657 rows=0 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Rows Removed by Filter: 1000000
-> Seq Scan on t1_7 t1_8 (cost=0.00..16925.00 rows=1 width=8) (actual
time=59.788..59.788 rows=0 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Rows Removed by Filter: 1000000
-> Seq Scan on t1_8 t1_9 (cost=0.00..16925.00 rows=1 width=8) (actual
time=60.740..60.740 rows=0 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Rows Removed by Filter: 1000000
-> Seq Scan on t1_9 t1_10 (cost=0.00..16925.00 rows=1 width=8) (actual
time=55.382..55.382 rows=0 loops=1)
  Filter: (id = 'ID1'::bpchar)
  Rows Removed by Filter: 1000000
Planning Time: 0.821 ms
JIT:
  Functions: 20
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 2.086 ms, Inlining 0.000 ms, Optimization 0.562 ms, Emission
19.462 ms, Total 22.111 ms
Execution Time: 2298.119 ms
(36 rows)

```

As shown in Example 8-4 on page 223, this query plan accesses and uses all the partitions t1_1 - t1_9. Notice that the execution time that is shown in the last line is 2298 ms without partition pruning. The execution time was 932 ms with partition pruning, as shown in Example 8-3 on page 223, which means that the partition pruning feature produced improved query performance in our test.

SQL tuning cases to enable partition pruning

Partitioning is a relatively new feature that was first supported in PostgreSQL 10, which was released in 2017. Since then, this feature has been enhanced and improved continuously, and now includes partition pruning. Partition pruning is a feature that improves query performance, but it does not always work effectively in all cases.

Even if an SQL query does not support partition pruning, partition pruning can be enabled with SQL tuning to improve SQL query performance.

In this section, we present two use cases of SQL tuning. These examples use table t1 in Example 8-1 on page 222 and table t2 in Example 8-5.

Example 8-5 Definition of table t2

```

[fsepuser@rdbkpg1 ~]$ psql -p 27500 -d postgres
psql (13.1)
Type "help" for help.
postgres=# CREATE TABLE t2 (id char(3), value2 int);
CREATE TABLE

```



```
postgres=# INSERT INTO t2 SELECT 'ID' || (i / 10), i from generate_series(0, 99)
as i;
INSERT 0 100
postgres=# INSERT INTO t2 SELECT 'ID' || (i / 10), 99 - i from generate_series(0,
99) as i;
INSERT 0 100
```

Use case 1

Use case 1 uses the SQL query that is shown in Example 8-6. This query specifies `id = 'ID1'` as a condition for the sub-query. Because the main query specifies condition is `t1.id = t3.id`, this query retrieves only rows where `t1.id` is `ID1`. Running this query requires searching only the `t1_1` partition.

Example 8-6 Use case 1: SQL before tuning

```
SELECT * FROM t1, (SELECT id, max(value2) FROM t2 where id = 'ID1' GROUP BY id) t3
WHERE t1.id = t3.id;
```

However, in some cases, PostgreSQL may not allow partition pruning based on the conditions that are specified in the sub-query. Therefore, as shown in Example 8-7, all partitions are accessed and searched, and the execution time is 17,907 ms.

Example 8-7 Use case 1: Query plan before tuning

```
[fsepuser@rdbkpgr1 ~]$ psql -p 27500 -d postgres -c "EXPLAIN ANALYZE SELECT * FROM
t1, (SELECT id, max(value2) FROM t2 where id = 'ID1' GROUP BY id) t3 WHERE t1.id =
t3.id;"
```

QUERY PLAN

```
-----
Hash Join (cost=3.89..231628.89 rows=9000000 width=16) (actual
time=1755.674..17476.359 rows=1000000 loops=1)
  Hash Cond: (t1.id = t2.id)
    -> Append (cost=0.00..194250.00 rows=10000000 width=8) (actual
time=0.100..12519.146 rows=10000000 loops=1)
      -> Seq Scan on t1_0 t1_1 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.098..450.030 rows=1000000 loops=1)
      -> Seq Scan on t1_1 t1_2 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.032..436.196 rows=1000000 loops=1)
      -> Seq Scan on t1_2 t1_3 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.038..441.154 rows=1000000 loops=1)
      -> Seq Scan on t1_3 t1_4 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.027..434.305 rows=1000000 loops=1)
      -> Seq Scan on t1_4 t1_5 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.036..426.024 rows=1000000 loops=1)
      -> Seq Scan on t1_5 t1_6 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.034..427.988 rows=1000000 loops=1)
      -> Seq Scan on t1_6 t1_7 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.029..424.753 rows=1000000 loops=1)
      -> Seq Scan on t1_7 t1_8 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.020..428.120 rows=1000000 loops=1)
      -> Seq Scan on t1_8 t1_9 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.016..445.612 rows=1000000 loops=1)
      -> Seq Scan on t1_9 t1_10 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.017..431.368 rows=1000000 loops=1)
    -> Hash (cost=3.78..3.78 rows=9 width=8) (actual time=11.901..11.905 rows=1
loops=1)
```

```

        Buckets: 1024 Batches: 1 Memory Usage: 9kB
        -> GroupAggregate (cost=0.00..3.69 rows=9 width=8) (actual
time=11.896..11.898 rows=1 loops=1)
            Group Key: t2.id
            -> Seq Scan on t2 (cost=0.00..3.50 rows=20 width=8) (actual
time=11.858..11.879 rows=20 loops=1)
                Filter: (id = 'ID1'::bpchar)
                Rows Removed by Filter: 180
    Planning Time: 0.529 ms
    JIT:
        Functions: 13
        Options: Inlining false, Optimization false, Expressions true, Deforming true
        Timing: Generation 1.334 ms, Inlining 0.000 ms, Optimization 0.390 ms, Emission
11.333 ms, Total 13.057 ms
    Execution Time: 17907.322 ms
(26 rows)

```

Even if partition pruning does not work as shown in Example 8-6 on page 225, it is possible to enable partition pruning by adding the search condition `id = 'ID1'` in the main query, as shown in Example 8-8.

Example 8-8 Use case 1: SQL after tuning

```

SELECT * FROM t1, (SELECT id, max(value2) FROM t2 where id = 'ID1' GROUP BY id) t3
WHERE t1.id = t3.id and t1.id = 'ID1';

```

The query plan for the SQL that is shown in Example 8-8 is shown in Example 8-9. In this query plan, only partition `t1_1` is used. As a result, the execution time is reduced to 3593 ms, which improves performance.

Example 8-9 Use case 1: Query plan after tuning with partition pruning enabled

```

[fseuser@rdbkpr1 ~]$ psql -p 27500 -d postgres -c "EXPLAIN ANALYZE SELECT * FROM
t1, (SELECT id, max(value2) FROM t2 where id = 'ID1' GROUP BY id) t3 WHERE t1.id =
t3.id and t1.id = 'ID1';"

```

QUERY PLAN

```

-----
Nested Loop (cost=0.00..129428.80 rows=9000000 width=16) (actual
time=14.037..3140.969 rows=1000000 loops=1)
    -> Seq Scan on t1_1 t1 (cost=0.00..16925.00 rows=1000000 width=8) (actual
time=13.970..512.296 rows=1000000 loops=1)
        Filter: (id = 'ID1'::bpchar)
    -> Materialize (cost=0.00..3.82 rows=9 width=8) (actual time=0.000..0.001
rows=1 loops=1000000)
        -> GroupAggregate (cost=0.00..3.69 rows=9 width=8) (actual
time=0.055..0.057 rows=1 loops=1)
            Group Key: t2.id
            -> Seq Scan on t2 (cost=0.00..3.50 rows=20 width=8) (actual
time=0.012..0.035 rows=20 loops=1)
                Filter: (id = 'ID1'::bpchar)
                Rows Removed by Filter: 180
    Planning Time: 0.461 ms
    JIT:
        Functions: 12
        Options: Inlining false, Optimization false, Expressions true, Deforming true

```

Timing: Generation 1.331 ms, Inlining 0.000 ms, Optimization 0.404 ms, Emission 12.987 ms, Total 14.722 ms
Execution Time: 3593.280 ms
 (15 rows)

Use case 2

Use case 2 uses the SQL query that is shown in Example 8-10. This query specifies `t2.value2 = 11` as a search condition, which means that the `t1.id` column, which is the partition key, is not specified in the search condition.

Example 8-10 Use case 2: SQL before tuning

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id AND t2.value2 = 11;
```

Therefore, as shown in Example 8-11, all partitions are accessed, and the execution time is 19,545 ms.

Example 8-11 Use case 2: Query plan before tuning

```
[fsepuser@rdbkpggr1 ~]$ psql -p 27500 -d postgres -c "EXPLAIN ANALYZE SELECT * FROM t1, t2 WHERE t1.id = t2.id AND t2.value2 = 11;"
```

QUERY PLAN

```
-----
Hash Join (cost=3.52..264253.53 rows=2000000 width=16) (actual
time=1702.343..18657.177 rows=2000000 loops=1)
  Hash Cond: (t1.id = t2.id)
    -> Append (cost=0.00..194250.00 rows=10000000 width=8) (actual
time=0.030..12953.621 rows=10000000 loops=1)
      -> Seq Scan on t1_0 t1_1 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.028..436.972 rows=1000000 loops=1)
      -> Seq Scan on t1_1 t1_2 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.102..447.430 rows=1000000 loops=1)
      -> Seq Scan on t1_2 t1_3 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.078..454.245 rows=1000000 loops=1)
      -> Seq Scan on t1_3 t1_4 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.047..484.550 rows=1000000 loops=1)
      -> Seq Scan on t1_4 t1_5 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.052..446.387 rows=1000000 loops=1)
      -> Seq Scan on t1_5 t1_6 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.066..433.649 rows=1000000 loops=1)
      -> Seq Scan on t1_6 t1_7 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.050..447.711 rows=1000000 loops=1)
      -> Seq Scan on t1_7 t1_8 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.044..454.333 rows=1000000 loops=1)
      -> Seq Scan on t1_8 t1_9 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.019..477.213 rows=1000000 loops=1)
      -> Seq Scan on t1_9 t1_10 (cost=0.00..14425.00 rows=1000000 width=8)
(actual time=0.013..432.855 rows=1000000 loops=1)
    -> Hash (cost=3.50..3.50 rows=2 width=8) (actual time=8.330..8.333 rows=2
loops=1)
      Buckets: 1024 Batches: 1 Memory Usage: 9kB
      -> Seq Scan on t2 (cost=0.00..3.50 rows=2 width=8) (actual
time=8.314..8.323 rows=2 loops=1)
      Filter: (value2 = 11)
      Rows Removed by Filter: 198
Planning Time: 0.577 ms
```

JIT:

Functions: 9

Options: Inlining false, Optimization false, Expressions true, Deforming true

Timing: Generation 0.880 ms, Inlining 0.000 ms, Optimization 0.347 ms, Emission 7.850 ms, Total 9.077 ms

Execution Time: 19545.829 ms

(24 rows)

Now, think about the **INSERT** statement that was shown in Example 8-5 on page 224. Table `t2` has only two corresponding ID columns when `value2` is determined. Suppose it is known that even if table `t2` is updated in the future, only a few ID columns corresponding to the value of `value2` will be updated. In this case, it is a best practice that you change the SQL query to retrieve the `id` value corresponding to `value2` first, and then specify the search condition by using the `id` value, as shown in Example 8-12.

Example 8-12 Use case 2: SQL after tuning

```
SELECT DISTINCT t2.id FROM t2 WHERE t2.value2 = 11;
// This SQL returns 'ID1' and 'ID9'
SELECT * FROM t1, t2 WHERE t1.id = t2.id AND t2.value2 = 11 AND t1.id IN ('ID1',
'ID9');
```

In the second **SELECT** statement, `t1.id` is specified in the search condition. Therefore, as shown in Example 8-13, partition pruning is used and only two partitions, `t1_1` and `t1_9`, are accessed, which results in a total execution time of 4,670 ms for the two **SELECT** statements, which improves performance.

Example 8-13 Use case 2: Query plan after tuning with partition pruning enabled

```
[fsepuser@rdbkpg1 ~]$ psql -p 27500 -d postgres -c "EXPLAIN ANALYZE SELECT
DISTINCT t2.id FROM t2 WHERE t2.value2 = 11;"
```

QUERY PLAN

```
-----
Unique (cost=3.51..3.52 rows=2 width=4) (actual time=0.049..0.056 rows=2
loops=1)
-> Sort (cost=3.51..3.51 rows=2 width=4) (actual time=0.048..0.050 rows=2
loops=1)
```

Sort Key: id

Sort Method: quicksort Memory: 25kB

```
-> Seq Scan on t2 (cost=0.00..3.50 rows=2 width=4) (actual
time=0.010..0.019 rows=2 loops=1)
```

Filter: (value2 = 11)

Rows Removed by Filter: 198

Planning Time: 0.183 ms

Execution Time: 0.123 ms

(9 rows)

```
[fsepuser@rdbkpg1 ~]$ psql -p 27500 -d postgres -c "EXPLAIN ANALYZE SELECT * FROM
t1, t2 WHERE t1.id = t2.id AND t2.value2 = 11 AND t1.id IN ('ID1', 'ID9');"
QUERY PLAN
```

```
-----
Hash Join (cost=3.52..57853.53 rows=400000 width=16) (actual
time=0.073..4240.195 rows=1000000 loops=1)
```

Hash Cond: (t1.id = t2.id)

```
-> Append (cost=0.00..43850.00 rows=2000000 width=8) (actual
time=0.033..2775.092 rows=2000000 loops=1)
```

```

-> Seq Scan on t1_1 (cost=0.00..16925.00 rows=1000000 width=8) (actual
time=0.031..519.849 rows=1000000 loops=1)
  Filter: (id = ANY ('{ID1,ID9}'::bpchar[]))
-> Seq Scan on t1_9 t1_2 (cost=0.00..16925.00 rows=1000000 width=8)
(actual time=0.019..533.337 rows=1000000 loops=1)
  Filter: (id = ANY ('{ID1,ID9}'::bpchar[]))
-> Hash (cost=3.50..3.50 rows=2 width=8) (actual time=0.020..0.024 rows=2
loops=1)
  Buckets: 1024 Batches: 1 Memory Usage: 9kB
-> Seq Scan on t2 (cost=0.00..3.50 rows=2 width=8) (actual
time=0.005..0.016 rows=2 loops=1)
  Filter: (value2 = 11)
  Rows Removed by Filter: 198
Planning Time: 0.547 ms
Execution Time: 4670.653 ms
(14 rows)

```

In this section, we described specific examples of SQL tuning to improve query performance. SQL performance tuning requires field experience because it involves deep understanding of the working of PostgreSQL query engine processing. In addition, often the necessity of SQL tuning is brought to attention only after performance verification is performed at the end of the migration process, which means that SQL tuning knowledge is required to keep the schedule as planned and ensure a successful migration.

Note: For more information about Fujitsu performance tuning, see 8.3.2, “Performance tuning tips” on page 258.

8.2.4 Optimizing database migration costs

One of the advantages of adopting open source PostgreSQL is reducing licensing fees. This section introduces two other aspects of optimizing migration costs:

- ▶ Optimization of hardware resources for database systems, which is important to further reduce cost. Fujitsu Enterprise Postgres provides the cache feature, which reduces memory usage. This feature enables systems to reduce the required memory resources for databases and reduce cost.
- ▶ Acquiring knowledge of database migration. When using open source PostgreSQL in migration projects, organizations might take time to investigate and understand the expertise and knowledge that is required for migration because the information about migration to open source PostgreSQL is not consolidated and systematized for easy consumption. Fujitsu offers homogeneous and heterogeneous database migration services to solve this problem.

Reducing memory usage with the Global Meta Cache feature

Database caching is a feature that stores frequently queried data in the memory to minimize I/O. For this mechanism to work effectively, a considerable portion of the memory of database servers should be used for data caching.

Applications that are used in enterprise systems often require concurrent connections to databases to improve throughput. In multiprocessing, memory is allocated for each process, even for common information, which might lead to a lack of memory.

Fujitsu Enterprise Postgres provides the Global Meta Cache feature to reduce memory usage by deploying a *meta cache*, which is the common information between connections, on shared memory and deploying only the process-specific information to each process memory. In enterprise systems with several thousand connections and more than 100,000 tables, Global Meta Cache reduces memory usage from a dozen terabytes to several dozen gigabytes.

Note: For more information about Global Meta Cache on FUJITSU Enterprise Postgres, see 2.3.3 “Global Meta Cache” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

Benefits of Fujitsu Professional Services

Migration projects involve complexity. Knowledge about the differences in source and target databases is essential for a successful migration project. This expertise is used to identify what changes are required and how to migrate to the target databases.

Note: For more information about migration technical knowledge, see 8.3.1, “Experience-based migration technical knowledge” on page 231.

In addition, understanding the features of the target product of migration is critical to bringing out the performance of the product and ensuring stability. Therefore, in enterprise systems where stable operation is a key requirement, skills development of the members performing the migration work is essential.

The content of the required training depends on the level of proficiency of the members performing the migration work and the roles that they are responsible for. Fujitsu offers Professional Services to flexibly assist organizations.

8.3 Success-focused migration methodology

As described in 8.2, “Key considerations for database migration” on page 212, there are several requirements that must be considered, beginning with a pre-migration planning phase. A migration project uses product features and knowledge that is based on migration experience to achieve these requirements and ensure that the project completes as planned.

In a migration project from Oracle Database to FUJITSU Enterprise Postgres, database engineers must consider the feasibility of migration from several perspectives. Projects can involve migrating platforms, business applications, and database servers to achieve the equivalent or better refined operations on FUJITSU Enterprise Postgres. Organizations must plan carefully to meet their requirements when moving the workloads to the new platform.

In this chapter, the knowledge and approach for successful migration from Oracle Database to Fujitsu Enterprise Postgres are introduced:

- ▶ Experience-based migration technical knowledge: Introduction to the Fujitsu approach for pre-migration planning and migration.
- ▶ Performance tuning tips: Outline of the tasks that are required for performance tuning.

Note: The migration expertise that is introduced in this chapter is essential to the success of migration projects. For more information about migration works, contact Fujitsu Professional Services at:

<https://www.postgresql.fastware.com/contact>

8.3.1 Experience-based migration technical knowledge

The goal of system migration is to provide equivalent functions in the target system and leverage target system features so that organizations benefit from the advantages from having conducted the migration. To achieve this objective, you must understand the differences in database architecture and functions and follow the migration process to migrate to the target system.

This section includes the following topics:

- ▶ General differences in database architecture and functions
- ▶ The migration process

General differences in database architecture and functions

Database migration requires an understanding of the architecture of both source and target databases. This knowledge is important to design the database configuration and operations.

The following sections cover the differences between Oracle and Fujitsu Enterprise Postgres in the following areas:

- ▶ File structure of tables
- ▶ Concurrency control
- ▶ Transactions
- ▶ Locking
- ▶ Expansion of data storage capacity
- ▶ History of data changes
- ▶ Encoding
- ▶ Database configuration files
- ▶ Schemas
- ▶ Other differences and their complexity level of migration

File structure of tables

Like Oracle, Fujitsu Enterprise Postgres is an object-relational database management system (ORDBMS). Both represent data that is grouped into relations (or tables) and store data belonging to relations in separate physical files. However, the structure of the physical files is different between Oracle and Fujitsu Enterprise Postgres, as shown in Figure 8-9.

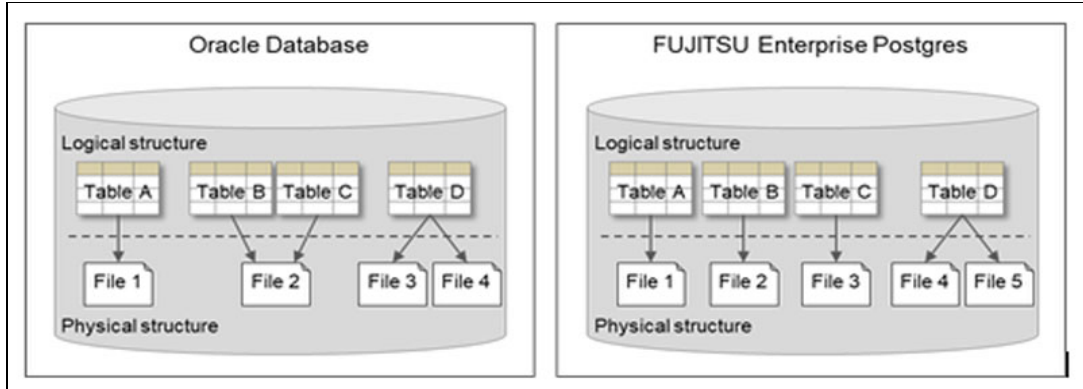


Figure 8-9 File structure of tables

Table 8-2 shows a comparison between Oracle Database and Fujitsu Enterprise Postgres. A key difference to be aware of during migration is that Fujitsu Enterprise Postgres cannot store the data of multiple tables in one data file.

Table 8-2 Oracle and Fujitsu Enterprise Postgres file structure comparison

Oracle Database	Fujitsu Enterprise Postgres
Oracle Database stores objects such as tables and indexes in data files, which are physical files. Data files can be allocated to table spaces.	Fujitsu Enterprise Postgres creates one or more physical data files and stores data of one table. Data files are stored in a fixed directory. However, by using a table space, you can store data files in any directory.
Oracle Database stores table data as follows: <ul style="list-style-type: none"> ▶ The data of one table can be divided into multiple data files and stored. ▶ The data of multiple tables can be stored in one data file. 	Fujitsu Enterprise Postgres Fujitsu Enterprise Postgres stores table data as follows. <ul style="list-style-type: none"> ▶ The data of one table can be divided into multiple data files and stored. ▶ The data of multiple tables cannot be stored in one data file.

Concurrency control

The management of concurrent access to data is essential for good performance. It also prevents excessive locking that might potentially restrict access to data while still allowing the flexibility that is provided by different isolation levels.

Fujitsu Enterprise Postgres achieves concurrency with read consistency by using a similar method to Oracle Database. Both apps make multiple copies of a row to present the appropriate information to a client based on when the transaction started. However, Fujitsu Enterprise Postgres does not remove the old copies of row data when it is updated or deleted, which results in physical files that can increase considerably in size, especially where a high frequency of update occurs. Therefore, the key consideration is the strategy to mitigate excessive growth of physical files, which can lead to performance degradation over time.

Table 8-3 compares concurrency control between Oracle Database and FUJITSU Enterprise Postgres.

Table 8-3 Oracle and Fujitsu Enterprise Postgres concurrency control comparison

Oracle Database	FUJITSU Enterprise Postgres
When Data Manipulation Language (DML) statements change data, Oracle Database stores the old value of data in an UNDO table space. This space can be reused when it is no longer needed for other data updates.	When DML statements change data, Fujitsu Enterprise Postgres marks the old value of row data and adds new values of data to the end. The old value of data is not deleted and left for the purposes of rollback.
If data that is changed in a transaction is not committed and the same data is viewed in another session, Oracle uses UNDO information. UNDO information is data that is already committed when viewed.	If data that is changed in a session is not committed and the same data is viewed in another session, Fujitsu Enterprise Postgres uses the old value of row data.

Note: Fujitsu Enterprise Postgres requires running VACUUM regularly. VACUUM enables the reuse of data storage spaces that are marked as used. VACUUM can also be configured to run automatically by using the autovacuum feature.

Transactions

Fujitsu Enterprise Postgres supports transactions in the same way as Oracle Database. However, there are differences in autocommit and transaction error handling.

Because commit is run in different units, application changes might be required when migrating. Oracle Database commits per statement, but Fujitsu Enterprise Postgres commits per transaction.

Table 8-4 shows the comparison of transaction processing between Oracle and FUJITSU Enterprise Postgres.

Table 8-4 Transaction processing comparison: Oracle Database and FUJITSU Enterprise Postgres

Oracle Database	FUJITSU Enterprise Postgres
Auto commit: <ul style="list-style-type: none"> ▶ A Data Definition Language (DDL) statement commits data automatically. ▶ A DML statement does not commit data automatically. 	Auto commit: <ul style="list-style-type: none"> ▶ A DDL statement does not commit data automatically. ▶ A DML statement commits automatically unless explicitly specified.
Error handling: If an error occurs within a transaction but COMMIT is run at the end of it, Oracle Database commits the data that results from successful DML statements execution.	Error handling: If an error occurs within a transaction, Fujitsu Enterprise Postgres roll s back the transaction, even if COMMIT is run at the end of the transaction.

Locking

Locks are supported on both Fujitsu Enterprise Postgres and Oracle Database. However, there are some differences in lock behavior.

Fujitsu Enterprise Postgres waits to acquire the lock unless applications have the appropriate settings. So, the applications wait for a response from FUJITSU Enterprise Postgres. Therefore, application changes might be required as part of the migration.

Table 8-5 shows a comparison of locking between an Oracle Database and a Fujitsu Enterprise Postgres database.

Table 8-5 Oracle Database and Fujitsu Enterprise Postgres locking comparison

Oracle Database	FUJITSU Enterprise Postgres
Oracle Database supports table-level locks and row-level locks.	Fujitsu Enterprise Postgres also supports table-level locks and row-level locks.
Explicit Locks are obtained by applications by using the following SQL statements: <ul style="list-style-type: none"> ▶ The LOCK TABLE statement sets a table-level lock. ▶ The SELECT statement with the FOR UPDATE clause or the FOR SHARE clause sets a row-level lock. 	Explicit Locks are obtained by applications by using the following SQL statements: <ul style="list-style-type: none"> ▶ A LOCK TABLE statement sets a table-level lock. ▶ A SELECT statement with the FOR UPDATE clause or the FOR SHARE clause sets a row-level lock.
A DDL statement sets the appropriate locks automatically. If the lock cannot be set, an error occurs.	A DDL statement sets the appropriate locks automatically. If DDL cannot set a lock, the application waits until a lock is set.

Expanding data storage capacity

The process for increasing storage capacity differs between Oracle Database and FUJITSU Enterprise Postgres. Fujitsu Enterprise Postgres requires that you copy the contents of an existing table space to a new larger replacement table space, which requires some planning.

Fujitsu Enterprise Postgres requires all the data, such as tables, to be copied to expand the capacity. Therefore, when designing the target database system, it is a best practice to consider the following items to avoid the immediate need of expanding data capacity:

- ▶ Determine the appropriate disk size and table space configuration.
- ▶ Determine whether to use partitioning.
- ▶ Determine an appropriate vacuuming strategy.

If it is necessary to expand the data capacity after going into production, plan and implement expansion while considering the data copy time.

Table 8-6 shows a side-by-side comparison of Oracle Database and Fujitsu Enterprise Postgres data storage capacity expansion.

Table 8-6 Data storage capacity expansion comparison: Oracle Database and FUJITSU Enterprise Postgres

Oracle Database	FUJITSU Enterprise Postgres
Expanding capacity is achieved by increasing the size of table space.	The size of a table space cannot be increased. To expand the capacity, a new table space must be created on a new disk. Then, all the data, such as tables and indexes, must be moved to the new table space.

Encoding

There are differences in the supported encodings between Oracle Database and FUJITSU Enterprise Postgres. If the source database uses an encoding that Fujitsu Enterprise Postgres does not support, change it to one of the supported encodings on FUJITSU Enterprise Postgres.

When migrating a database, consider which encoding to use on the target system.

Database configuration files

Configuration files that are used on Oracle Database cannot be used on FUJITSU Enterprise Postgres.

Review the parameters and connection settings on Oracle Database and set the parameters in the Fujitsu Enterprise Postgres configuration files to have the same effect as Oracle Database.

Schemas

Fujitsu Enterprise Postgres supports the concept of a schema like Oracle Database. However, there are differences in their functions.

Oracle Database automatically creates a schema with the same name as the user. Fujitsu Enterprise Postgres has a “public” schema by default. A schema with the same name as the user is not automatically created.

Because of the differences in automatically created schemas and schema search orders, design settings and definitions for Fujitsu Enterprise Postgres so that the schema can be used in the same way as database operations, such as data extraction in Oracle Database. Specifically, it is important to set the **search_path** parameter. Fujitsu Enterprise Postgres uses the **search_path** parameter to manage the schema search path, which is the list of schemas to look in. If a schema name is not specified when running queries, Fujitsu Enterprise Postgres uses the search path to determine which object is meant. The order that is specified in the search path is used to search the schema, and the first matching object is taken to be the one that is wanted, and it is used for query execution.

By default, the user who created the schema owns the schema. Therefore, appropriate privileges are required to allow other users access.

Other differences and their complexity level of migration

In a migration project, many different areas are impacted. In addition to the key differences listed earlier, you must assess the feasibility of database migration.

This section explains the migration complexities of major features from Oracle Database to Fujitsu Enterprise Postgres and a description of the differences that you should be aware of. The migration complexity level is outlined in Table 8-7, which is key in subsequent tables to indicate the migration complexity level for each task.

Table 8-7 Migration complexity levels

Migration complexity level	Description
Level 0	A Fujitsu Enterprise Postgres feature is compatible with Oracle Database. No change is required when migrating.
Level 1	A Fujitsu Enterprise Postgres feature is compatible with Oracle Database, but there are some differences, such as the interface. Some changes are required when migrating.
Level 2	Fujitsu Enterprise Postgres supports major features. However, some features, such as functions or syntax, are not compatible. Redesign and changes are required for these incompatibilities when migrating.
Level 3	A Fujitsu Enterprise Postgres feature is not compatible with Oracle Database. When migrating, design work is required to implement functions that are equivalent to Oracle Database.

The major database elements that might be impacted are outlined in Table 8-8.

Table 8-8 Major database elements

Features		Migration complexity level	Description
Encoding		2	Fujitsu Enterprise Postgres supports Unicode and EUC. Other encodings require redesign and changes.
Maximum database capacity		0	-
Quantitative limit of table		0	-
Number of indexes in a table		0	-
Table index types		2	Fujitsu Enterprise Postgres supports B-tree indexes. Other index types require redesign and changes.
Data types	Character types	2	Some character types such as CHAR and VARCHAR2 are supported, but there is a difference in how the maximum size is specified. Some character types such as CLOB and NCLOB are not supported.
	Numeric types	2	Even if Fujitsu Enterprise Postgres supports the same numeric types, there are differences in the number of significant digits and truncation for some data types.
	Date and time types	2	Even if Fujitsu Enterprise Postgres supports the same date and time types, there are differences in precision, time zone specification, and format for some data types.
	Binary data types	2	Fujitsu Enterprise Postgres does not support the same binary data types. Thus, redesign and changes are required.
	XML	2	Fujitsu Enterprise Postgres supports XML data type, but the function is not equivalent to Oracle Database.
	JSON	2	Fujitsu Enterprise Postgres supports JSON data type, but the function is not equivalent to Oracle Database.
Globalization support		1	-

The major performance elements and their migration complexity level are outlined in Table 8-9.

Table 8-9 Performance

Features	Migration complexity level	Description
Memory tuning	2	Memory setting changes are needed to suit the target system.
SQL tuning	2	Fujitsu Enterprise Postgres supports the HINT clause, but the function is not equivalent to Oracle Database.
Materialized view	2	Fujitsu Enterprise Postgres supports the materialized view, but only refreshing all rows is available.
Parallel query	1	-
In-memory columnar	1	Vertical Clustered Index is available on FUJITSU Enterprise Postgres.

The availability tasks are listed in Table 8-10.

Table 8-10 Availability

Features	Migration complexity level	Description
HA	1	For HA, Oracle Database uses RACs to configure redundant database servers, and Fujitsu Enterprise Postgres uses database multiplexing.
Disaster recovery	1	Fujitsu Enterprise Postgres enables disaster recovery with the streaming replication feature.

Table 8-11 lists the operational tasks and their migration complexity levels.

Table 8-11 Operational tasks

Features	Migration complexity level	Description
Operation management tool	2	Various tools that are available in PostgreSQL are also available in FUJITSU Enterprise Postgres.
Changing DB configuration: Adding columns or indexes	0	-
Reorganize index spaces	1	The REINDEX statement can reorganize the index spaces.
High-speed loader	1	-
Data replication	1	-

Features	Migration complexity level	Description
Database linkage for heterogeneous databases	2	Fujitsu Enterprise Postgres supports database linkage with Oracle Database. Other types of database linkage require redesign and changes.
Database Link	2	Fujitsu Enterprise Postgres can coordinate data between instances.
Applying patches	1	For a Fujitsu Enterprise Postgres clustered environment, rolling update is available during patching.

Table 8-12 lists the migration complexity level of security features.

Table 8-12 Security features

Features	Migration complexity level	Description
Data encryption	1	Fujitsu Enterprise Postgres supports TDE.
Data Masking	1	-
Row level access control	1	-
Security audit	2	Fujitsu Enterprise Postgres supports Audit Logs.

Migration complexity levels and their descriptions for application development are shown in Table 8-13.

Table 8-13 Application development

Features	Migration complexity level	Description	
Comply with SQL standards	2	Fujitsu Enterprise Postgres supports many of the major features of SQL:2016.	
Optimizer	2	Fujitsu Enterprise Postgres supports optimizer features, but there are differences in query hints and some functions of SQL query plan management.	
Interface	EmbeddedSQL (C language)	1	-
	ODBC	1	-
	JDBC	2	Fujitsu Enterprise Postgres supports JDBC standard features, but the supported versions are not the same as Oracle Database.
	.NET Framework	3	Fujitsu Enterprise Postgres does not support the .NET Framework.

Features		Migration complexity level	Description
Tools	Interactive SQL execution tool	2	SQL*Plus is available for Oracle Database, and psql is available for FUJITSU Enterprise Postgres. Usage of the tools is different.
	GUI tool for SQL execution	2	SQL Developer is available for Oracle Database, and pgAdmin is available for FUJITSU Enterprise Postgres. Usage of the tools is different.
	Development environment tool	2	The development environment tools must be changed or reconfigured to suit the target system.
Stored procedures and functions		2	To create or manipulate stored procedures and stored functions, PL/SQL is available for Oracle Database, and PL/pgSQL is available for FUJITSU Enterprise Postgres. PL/pgSQL is defined differently as PL/SQL. Fujitsu Enterprise Postgres is not compatible with Oracle Database Java. Design how to implement the equivalent function.
Access control	Lock level	0	Both databases support table-level locks and row-level locks.
	How to obtain locks	0	Both databases can obtain locks per query (command).
	Automatic deadlock detection	0	-
	Data concurrency and consistency	0	Both databases maintain data concurrency by using a multiversion model.
Connection management	Basic management	1	This feature manages access to the appropriate server during redundancy.
	Alive monitoring	2	Oracle RAC or Oracle Active Data Guard enable alive monitoring on Oracle Database. Pgpool-II or Connection Manager enable alive monitoring on FUJITSU Enterprise Postgres. However, the configuration and operation for monitoring are different.
	Application continuity	2	Oracle RAC or Oracle Active Data Guard enable alive monitoring on Oracle Database. Pgpool-II or Connection Manager enable alive monitoring on FUJITSU Enterprise Postgres. However, the configuration and operation for monitoring are different.

Differences in backup and recovery are shown in Table 8-14.

Table 8-14 Backup and recovery

Features	Migration complexity level	Description
Backup units	2	Fujitsu Enterprise Postgres backs up data per database, instance, table, or partition. However, backup per table space is not supported.
Backup settings	1	-
Recovery	1	-

The migration process

That the database system and its data are the “heart and lungs” of the organization is an analogy that serves as a good reminder of how critical data is to an organization. Like cardiac surgery, database migration requires careful consideration and the weighing of the benefits and risks, which are followed by detailed planning. These essential activities cannot be performed without accurate and detailed information regarding the following items:

- ▶ Structure and data
- ▶ Usage
- ▶ Performance
- ▶ Security and governance
- ▶ Tools and operational processes
- ▶ Integration, warranties, and support

Therefore, a quality migration assessment should be one of the first activities that an organization undertakes if they are considering a change in their DBMS.

This section first describes topics that anyone about to conduct a migration assessment should think about, particularly in terms of establishing appropriate objectives for an assessment. Then, it describes the process that is used to migrate database resources and applications.

Migration assessment considerations

In this section, we describe three important points to consider when performing a migration assessment:

- ▶ Timing

Due to a poor understanding of the extent of the areas that migrations can impact, it is common to see plans that incorporate an assessment activity immediately before a block of time for DB migration and application migration. Such plans might be an indication of poor chances of success.

Although changing database platforms might be the result of a strategic decision (such as adopting open source software or cloud computing strategy), an early understanding of the impacted areas is essential in good planning.

Although database administrators and application teams are generally the core part of a migration team, other teams such as infrastructure, DevOps, security, vendor, business, and support are all required to be involved at some point:

- Extra hardware is often required to perform a migration. Some strategies to avoid downtime involve running the old and new database platforms side by side for an extended period.
- Organizational policies often dictate how data should be secured and conformed to industry accepted security benchmarks. Often, different features must be implemented and configured to meet these goals in a new product.
- Monitoring and alerting might need to be integrated into existing systems, or retraining might be a requirement for DevOps and support teams to support the new system.
- Replacement of backup and recovery software and processes, or integration into existing archiving solutions, might be required.
- Training of staff in the new technology or administration tools is often required.

Therefore, the output of a quality migration assessment at the beginning of a migration journey allows organizations to make informed decisions and build a migration plan that ensures success by accounting for every aspect.

► Feasibility

The decision to change DBMSs should consider many factors, and the methodology to decide whether to migrate can be different for every organization. The amount of importance that an organization might allocate to a particular factor might differ considerably. However, some factors do stand out as a considerable obstacle to performing a migration.

One example is where vendors warrant their software only when it is used with a specific database. If the application that uses the database is a third-party product, check with the vendor about support for your proposed database platform because warranties might be affected.

Partnering with an experienced database migration service provider helps to identify those things that your organization should be weighing in its decision on whether migration is feasible.

► Understanding effort

Understanding the type of effort, where to expend it, and the amount that is required is important to calculate the time and cost of a migration. Here is an overview of some of the different areas where effort can be required and how you can estimate it.

You leverage automation in almost everything that you do to reduce effort, and automation can certainly help in reducing the effort that is required to migrate a database to FUJITSU Enterprise Postgres. However, there is often a large capital expense that is associated with automation investment. Fortunately, significant investment already has been made by various commercial and open source projects in this area (including Fujitsu). The focus of a substantial amount of this investment is around database structure, code, and data.

Database structure, code, and data are key foci because they are obvious things to be migrated, with clear mappings (in most cases) to a target database equivalent. However, there are other areas that should be thoroughly assessed where automated tools are not available, which are covered later in this section.

When using a tool to perform an automated assessment of the database schema, it should ideally be calibrated with the intended migration tool so that it is “aware” of the level of automation and can provide an accurate assessment of how much can be migrated automatically and how much requires manual effort.

Because most relational database implementations are based on an ANSI SQL Standard, there is a certain level of compatibility between different database vendor implementations. Therefore, tools generally work on the concept of identifying incompatibilities with the target database in the source database DDL and source files. These incompatibilities are broadly classified into those incompatibilities that can be migrated automatically and ones that require manual effort.

Manual effort is a relative (between incompatibilities) arbitrary value that is associated with each incompatibility type that can be adjusted by a multiplication factor:

- Experience of the migration team.
- Contingency buffer.
- Other complexities (like environmental).

The total effort is the total of the incompatibility manual effort that is multiplied by the multiplication factor.

For the estimate to be accurate, a good correlation between the experience of the team and the multiplication factor is required. Generally, the more experienced the team, the more accurate the factor, the shorter the estimate, and the more accurate the overall migration estimate. This reason is one to consider using a migration service provider like Fujitsu.

Another reason to consider the services of an experienced migration provider is their accumulated experience, which is typically consolidated and articulated through a knowledge base that covers best practices for resolving incompatibilities that require rewriting or technical know-how. The provider can represent significant savings compared to the same activity being conducted by a relatively inexperienced team.

For more information about the types of incompatibilities, see “Other differences and their complexity level of migration” on page 235.

Areas where automated assessment is difficult include the following ones:

- Architecture.

The database architecture focuses on the design and construction of a database system that can meet the defined requirements for the system. Such requirements cover features such as resilience, HA, security, flexibility, and performance.

DBMSs from different vendors deliver these features through different mechanisms, so architectures might vary across vendor implementations to achieve the same or equivalent functions.

HA is one area where these differences can occur. Shared storage that is used by two read/write instances might be regarded as a strength by one vendor but a weakness by another vendor (due to the single point of failure of the storage) who favors a hot standby with separate-replicated storage as a more resilient approach.

Careful consideration of what the requirements of the organization are and how they are best met by available architecture designs should be the key focus.

Often, requirements can be met by more than one architecture, so maintainability and flexibility should also be considered. Complex architectures might add risk when compared to simple ones that still deliver the needs of an organization.

- Security and governance.

Data security is a major concern for organizations with significant consequences for not complying with growing regulatory policies around the management of personal data. These consequences are financial penalties for noncompliance and the loss of trust from your customers.

Enterprise organizations have strict policies for how data must be protected and the benchmarks to be met. Features and configuration differ between vendors, and care must be taken to ensure that the configuration of target platforms continue to meet such policies and benchmarks.

- Tools.

Many commercial DBMSs come with their own brand of tools for administration, monitoring, backups, and so on. Changing DBMSs usually results in also having to use a different tool for these functions.

There are several tools that are available that provide organizations with suitable functions for these tasks. Which tools to use might depend on the specific requirements or areas of importance to them. Adequate time should be allocated for an evaluation of a suitable toolset and training in its use.

- Training.

Moving employees to a new DBMS can present some significant challenges. Staff has many years that are invested in a product and gained a level of competence that provides them with worth within an organization. Moving to a different product can create concerns for some employees about the potential loss of that investment in themselves.

Fujitsu Enterprise Postgres is like Oracle Database, and much of the existing knowledge that is possessed by Oracle DBAs can be applied to a Postgres product. However, employees should be encouraged to learn about the benefits of learning the new system.

- Licensing and subscriptions.

- Testing.

One of the most important areas of a migration is testing. Testing often makes up more than 40% of a migration project, but it is easily underestimated when assessing the migration effort.

These areas are often overlooked during migration projects and can result in the success of the project being compromised.

To accurately assess these areas requires a good understanding of the current environment (from a technical standpoint and an operational and governance perspective), and a good understanding of the target environment and how it can deliver the equivalent or better results. Therefore, migrations often involve a blended team that is made up of an organization's own subject matter experts (SMEs) and a migration partner with experience in the targeted platform.

Ensuring success

Planning for success is all about coverage and removing unknowns. A successful migration project is one that does a good job of mitigating risk, and risk mitigation is all about comprehensive scoping and detailed planning.

We mentioned in "Migration assessment considerations" on page 240 some of the different teams that should be involved in the project, such as security, DevOps, support, and business. Active participation by these teams increases coverage and reduces the risk of factors that have potential to impact business not being planned. In fact, managing a migration project from a business perspective rather than an IT one is one way that we can reduce risk.

Understanding the data being migrated and how migration activities affect customers and the impact on business in terms of loyalty and reputation is an important step to ensure that appropriate checks or backup options are in place.

Data migrations are seldom an isolated project. More often, they are part of a larger modernization or transformation project. If so, then close collaboration with the larger project can avoid many issues that are associated with waiting until the new system is complete.

Although automation helps mitigate technical risks, a thorough data verification process that runs at the data storage level as data is migrated helps to identify problems early before they impact the business. This process should be implemented in addition to user testing.

With testing, data verification, and data reconciliation, you can avoid an impact on the business through early detection of issues. However, identifying how data issues occurred can present its own set of challenges. Change Data Capture (CDC) or a data auditing capability should be built in to the migration process so that issues can be understood and quickly resolved.

Again, using the knowledge of an experienced migration service provider helps to plan for a successful migration.

Maximizing strengths

A successful migration should provide the business with new technology and the ability to use data in better ways that allows the business to respond to a fast-changing business environment. Therefore, new features of the data storage platform are a consideration during migration planning.

For example, the ability to store and access data in a JSON format is used by applications to exchange data. Should data from a source system be stored in a binary JSON column of the target database or as individual columns?

Another example that is applicable to Fujitsu Enterprise Postgres is the Vertical Columnar Index feature, which updates indexed columnar structures in memory as row data is updated. This feature allows an efficient execution of various analytical style queries. Using this feature is something that should be considered in migration planning.

These types of considerations require expert knowledge about the data and how it can be leveraged by the business, and the features of the target system and how to best leverage them.

Steps of the migration process

The Fujitsu migration process is a multi-step process that starts with assessing the source system to determine the feasibility of a migration project, as shown in Figure 8-10.

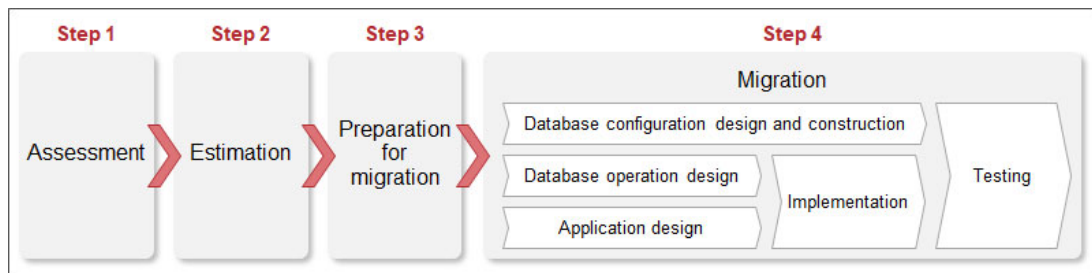


Figure 8-10 Migration process overview

The feasibility of migrating database resources is determined primarily by the level of the migration effort, which depends on the source system scale and construction. Therefore, migration projects should always start with the assessment step at the pre-migration planning stage.

The migration path to Fujitsu Enterprise Postgres includes the following main workflows, which are illustrated in Figure 8-10 on page 244:

► **Step 1: Assessment**

The source system is examined thoroughly in terms of database architecture, the size of the data, and assets such as database schemas. Then, the technical impact is analyzed, and the level of effort of the migration project is identified.

► **Step 2: Estimation**

In this step, the economic impact of a migration project is analyzed. The cost of the migration project is estimated, including testing, based on the assessment result. In practice, other costs such as the infrastructure that is required, temporary licenses, and training staff should also be considered. Based on this estimation, the project owner determines the feasibility and decides whether to proceed with this migration project.

► **Step 3: Preparation for migration**

The migration plan is created in this step. The plan includes a schedule and team structure that is determined based on the estimation result. Preparation also takes place to begin the next step by building the development environment and the production environment.

► **Step 4: Migration**

In the final step, migration is performed according to the following system development process:

- Database configuration design and construction
- Database operation design
- Application design
- Implementation
- Testing

Step 1: Assessment

Assessment is the first step of a migration project, which analyzes the technical impact and identifies the level of effort of the migration project. The level of the migration effort varies widely depending on several factors. Therefore, the source system must be explored in terms of database architecture, the size of data, and assets such as database schemas to understand what must be done in the migration step. The difficulty to complete the migration must be assessed. Assessment is required at the pre-migration planning stage.

One of the major consideration points for database migration is the impact on system performance. If system performance after migration is a key concern, performance validation is necessary during this step. Validation is done by pre-migrating some schemas and applications to evaluate whether they meet the system performance requirements.

In “Step 3: Preparation for migration” on page 247, the migration plan is created based on the skills and productivity of the engineers. Therefore, if this migration is the first time that you do a migration from Oracle Database to FUJITSU Enterprise Postgres, it is a best practice to evaluate the skills and productivity of engineers in this step.

The main exploration targets and assessment points are shown in Table 8-15.

Table 8-15 Exploration targets and assessment points

Exploration target	Assessment point
Database architecture	To analyze the technical impact of migration on the database architecture, assess the following points: <ul style="list-style-type: none"> ▶ Is the source system a single instance or a HA system? ▶ Is the source system using a database link?
SQL	To understand the volume and difficulty of SQL conversion, assess the following points: <ul style="list-style-type: none"> ▶ Data types. ▶ DDL and DML. ▶ Transaction Control Statements. ▶ Session Control Statements. ▶ System Control Statements. ▶ Operators, conditions, expressions, and functions.
PL/SQL	To understand the volume and difficulty of PL/SQL conversion, assess the following points: <ul style="list-style-type: none"> ▶ Data types. ▶ Packages. ▶ Subprograms. ▶ Transaction Processing and Control. ▶ GOTO Statement.
Application	To analyze the impact of application interface changes, check and clarify which interface is used in the source system: <ul style="list-style-type: none"> ▶ JDBC. ▶ ODBC. ▶ OCI. ▶ Pro*C.
Data	To identify how long data migration will take, assess the following points: <ul style="list-style-type: none"> ▶ The number of table constraints and indexes. ▶ Data size. ▶ Data migration strategy.
Operation	To analyze the technical impact of migration on database system operation, assess the following points: <ul style="list-style-type: none"> ▶ Database monitoring. ▶ Authentication. ▶ Backup and recovery. ▶ HA and high reliability.
Performance requirements	To identify the system performance requirements, assess the following points: <ul style="list-style-type: none"> ▶ Performance of interactive processing under normal load or under extreme load. ▶ Performance of batch processing.

Step 2: Estimation

The second step is to estimate the migration project cost and how long data migration will take. At the end of this step, the project owner determines the feasibility and decides whether to proceed with this migration project.

► Migration cost

Based on the assessment result, the migration effort and the required scope for design, implementation, and testing is determined. The cost of the migration project cost includes considerations of the amount of work that is required and the skills and productivity of the engineers performing the migration.

When estimating the testing efforts, it is a best practice to allow for sufficient time to prepare for database-migration-specific issues. When migrating a database, some differences between the source database and the target database might remain unnoticed in the design or implementation steps. For example, it is difficult to see the following differences before testing:

- Calculation results of numbers that run in SQL might differ due to the differences in rounding-up or rounding-down.
- The SQL output of date and time might differ due to the differences in the data types precision or format.

► Data migration time

It is also important to know in this step how long it takes to migrate data.

The migration time depends on the data size and the migration strategy. Even if the data volume is the same, the data migration time varies depending on the selected migration method and environment. Therefore, it is a best practice that you perform a data migration rehearsal in a test environment. The objective of the rehearsal is to verify that data migration will be successful and validate the migration time.

Step 3: Preparation for migration

The third step is to create a migration plan and prepare to begin the migration:

► Create a migration plan.

Create a migration plan that includes the project schedule and team structure based on the estimated results. The plan for go-live, such as the downtime that is required to switch to the target system, is also required.

► Preparation.

In the migration step, both the production and development environments are used. Therefore, both environments are built and set up in this step. For each environment, the equivalent architectures of source database systems and target database systems are built and configured. If some tools are going to be used for migration, the tools are also set up in this step.

- Production environment: This environment requires two systems. One is the source database system that is in use, and the other system is the target database systems that the organization uses after go-live.

- Development environment: In the migration step, the development environment is used for design, implementation such as programming, and testing:
 - Prepare both the source database system and the target database system on this environment. Build and configure the equivalent architecture of the production environment.
 - Copy all the assets that must be migrated from the source database systems on the production environment to the source database systems on the development environment. These assets include database schemas, applications, and batch files. The preparation of these assets can be simplified to reduce the time and effort that are required for preparation by focusing only on the elements that affect database migration. For example, for the development environment database, a sample data or test data that has similar characteristics to the production environment can be used.

Step 4: Migration

The final step is to migrate all the assets from Oracle Database to FUJITSU Enterprise Postgres. This step is the same as a system development process.

► Database configuration design and construction

The configuration of target databases is designed to meet the organization's system requirements and deliver a system that is equivalent to source databases. When designing, consider the differences in the database architecture between Oracle Database and FUJITSU Enterprise Postgres.

Table 8-16 shows two types of general database architecture. When designing, pay attention to the differences, especially for HA systems.

Table 8-16 General database architecture

Database architecture	Description
Single instance	One server contains a single database with one instance. There is no special consideration.
HA	A HA system consists of a group of servers that provide reliability with a minimal downtime. A clustered system is implemented with RACs on Oracle Database. This system is implemented with Database Multiplexing on FUJITSU Enterprise Postgres.

Note: For more information about HA on FUJITSU Enterprise Postgres, see 2.1.1, “Database multiplexing” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

► Database operation design

Operation and management of the target database system is designed. For more information about Fujitsu Enterprise Postgres features and implementation details for key operational requirements, see Table 8-17 on page 249.

Table 8-17 Key operational requirements

Operational requirement	Reference
Audit Logging	See 4.5, "Audit Logging" in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE</i> , SG24-8499.
Authentication	See 4.6, "Authentication" in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE</i> , SG24-8499.
HA and high reliability	See Chapter 5, "High availability and high reliability architectures" in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE</i> , SG24-8499.
Backup and recovery	See 9.1, "Backup and recovery overview" in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE</i> , SG24-8499.
Database monitoring	See 9.4, "Monitoring" in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE</i> , SG24-8499.
Database version upgrade	See Appendix A. "Version upgrade guide" in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE</i> , SG24-8499.
Database patching	See Appendix C. "Patching guide" in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE</i> , SG24-8499.

► Application design

A modification method for assets such as SQL and applications to run on the target database system is designed.

– SQL

The basic elements of SQL include data types, DDL, DML, and functions. Table 8-18 shows the key considerations of SQL migration. Table 8-19 on page 250 shows DDL statements. Table 8-20 on page 251 shows DML statements. Table 8-21 on page 252 and Table 8-22 on page 253 show functions. Table 8-22 on page 253 shows other types of SQL statements.

Note: Appendix C, "Converting SQL and PL/SQL to Fujitsu Enterprise Postgres SQL and PL/pgSQL" on page 387 provides specific examples of SQL migrations that are frequently used in Oracle Database.

Table 8-18 SQL data types in database applications

Data types	Description
Representing Character Data	<ul style="list-style-type: none"> ► Both databases support CHAR, VARCHAR2, NCHAR, and NVARCHAR2 types. However, the maximum sizes are specified differently. ► CLOB, NCLOB, and LONG types can be converted to TEXT type to store text.
Representing Numeric Data	Oracle Database and Fujitsu Enterprise Postgres support different number data types. Convert as needed to resolve the differences in significant figures and truncation.
Representing Date and Time Data	Both databases support date and time data types. However, convert as needed to resolve the differences in precision, time zone specification, and format.
Representing Specialized Data	<ul style="list-style-type: none"> ► Large Object data types, such as BLOB, store binary data. Convert to alternative data types on FUJITSU Enterprise Postgres. ► Both databases support XML data types and JSON data types, but the functions are different. Design how to convert. ► Other data types are not supported on FUJITSU Enterprise Postgres. Conversion must be accounted for.

Data types	Description
Identifying Rows by Address	Fujitsu Enterprise Postgres does not have equivalent data types for ROWID and UROWID. Convert as needed to identify the row by using the SERIAL type or SEQUENCE.
Displaying metadata for SQL Operators and Functions	Fujitsu Enterprise Postgres does not support equivalent data types for displaying metadata for SQL operators and functions such as ARGn data type. Conversion must be accounted for.

Table 8-19 DDL statements

DDL	Description
CREATE SCHEMA	The CREATE SCHEMA statement does not create a schema object on Oracle Database. This statement creates a new schema object on FUJITSU Enterprise Postgres. Specify a schema name that is different from your existing schemas. Otherwise, the database server issues an error.
CREATE DATABASE LINK	The CREATE DATABASE LINK statement creates a database link on Oracle Database that enables access to objects on another database. The Foreign Data Wrapper (FDW) function is used on Fujitsu Enterprise Postgres to implement the function to access objects on another database, which allows access to objects on Fujitsu Enterprise Postgres or Oracle Database. The CREATE EXTENSION statement enables an FDW feature that is supplied as additional modules.
CREATE TRIGGER	The CREATE TRIGGER statement creates a database trigger on both databases. However, conversion must account for the following differences. <ul style="list-style-type: none"> ▶ Syntax to write trigger functions. ▶ Languages to write trigger functions. ▶ Events that call a trigger's function.
CREATE INDEX	Oracle Database supports several types of indexes, such as B-tree indexes, bitmap indexes, and functional-based indexes. Fujitsu Enterprise Postgres supports only B-tree indexes. If Oracle Database uses indexes other than B-tree, they must be changed to B-tree indexes, or these indexes must be deleted. The difference in the length of the index keys and the data types that are specified for the index key must be considered.
CREATE MATERIALIZED VIEW	The CREATE MATERIALIZED VIEW statement creates a materialized view on both databases. However, conversion must account for the following differences: <ul style="list-style-type: none"> ▶ Features that are supported in materialized views. ▶ How to refresh materialized views. ▶ The syntax to write materialized views.
CREATE OPERATOR	The CREATE OPERATOR statement allows you to define operators on both databases. However, conversion must account for the differences in syntax.
CREATE SEQUENCE	The CREATE SEQUENCE statement creates a sequence on both databases. However, conversion must account for the differences in syntax.
CREATE FUNCTION and CREATE PROCEDURE	Both databases support the CREATE FUNCTION and CREATE PROCEDURE statements to create stored functions and stored procedures. However, conversion must account for the differences in syntax and languages.

DDL	Description
CREATE TABLE	The CREATE TABLE statement creates a relational table on both databases. However, conversion must account for the differences in the data types that are specified in the tables and syntax. After creating a table, partitions can be defined on both databases. However, conversion must account for the following differences: <ul style="list-style-type: none"> ▶ Types of partitioning. ▶ Partitioning features. ▶ Syntax to define partitions.
CREATE VIEW	The CREATE VIEW statement creates a view on both databases. However, Fujitsu Enterprise Postgres does not support the WITH READ ONLY option, so conversion must account for the differences in syntax.
CREATE ROLE	The CREATE ROLE statement create a role on both databases. However, conversion must account for the differences in syntax.
CREATE USER	The features of database users are different. A user on Fujitsu Enterprise Postgres is a type of role. Thus, conversion must account for the differences in the functions and DDL syntax. User authentication on Fujitsu Enterprise Postgres is managed in the configuration file <code>pg_hba.conf</code> .
CREATE *	Fujitsu Enterprise Postgres does not support database objects such as clusters, index-organized tables, object tables, packages, and synonyms. You must implement equivalent functions.

Table 8-20 DML statements

DML	Description
SELECT, INSERT, UPDATE, and DELETE	Both databases support basic DML statements. However, conversion must account for the differences in syntax.
MERGE	Oracle Database supports the MERGE statement to select rows from one or more sources for update or insertion into a table or view. The equivalent function in Fujitsu Enterprise Postgres is implemented by using the INSERT statement with the ON CONFLICT clause.
CALL	Both databases support the CALL statement. However, conversion must account for the differences in syntax.
EXPLAIN PLAN	The EXPLAIN PLAN statement on Oracle Database is equivalent to the EXPLAIN statement on FUJITSU Enterprise Postgres. However, conversion must account for the following differences: <ul style="list-style-type: none"> ▶ The executed results are not stored in a table. ▶ Syntax.
LOCK TABLE	Oracle Database supports the LOCK TABLE statement. The equivalent function is implemented in Fujitsu Enterprise Postgres by using the LOCK statement. Conversion must account for the differences in syntax and the name of the lock modes.

Table 8-21 Functions

Function		Description
Single-Row Functions	Numeric functions	Most numeric functions are supported on FUJITSU Enterprise Postgres. However, conversion must account for the differences in parameters and precision of return values. The following functions are not supported. Thus, other functions must be used for implementation. ▶ BITAND ▶ REMAINDER
	Character functions returning character values	Most character functions returning character values are supported on FUJITSU Enterprise Postgres. However, conversion must account for the differences in parameters. The following functions are not supported. Thus, other functions must be used for implementation. ▶ NCHR ▶ NLS_INITCAP ▶ NLS_LOWER ▶ NLS_UPPER ▶ SOUNDEX ▶ TRANSLATE ... USING
	Character functions returning number values	Most character functions returning number values are supported on FUJITSU Enterprise Postgres.
	Date and time functions	Most date and time functions are supported on FUJITSU Enterprise Postgres. However, conversion must account for the differences in precision and time zones that are available.
	General comparison functions	General comparison functions are supported on FUJITSU Enterprise Postgres.
	Conversion functions	Most conversion functions are supported on FUJITSU Enterprise Postgres. However, conversion must account for the differences in precision and parameters.
	Collection functions	CARDINALITY is not supported on FUJITSU Enterprise Postgres. The equivalent information is acquired by running SQL. Other collection functions are also not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions.
	XML functions	Most XML functions are supported on FUJITSU Enterprise Postgres. However, conversion must account for the differences in parameters.
	JSON functions	Both databases support JSON functions, but the function is different. Consider how to implement the equivalent function on FUJITSU Enterprise Postgres.
	Encoding and decoding functions	Both databases support DECODE , but the function is different. Consider how to implement an equivalent function on FUJITSU Enterprise Postgres. Other encoding and decoding functions are not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions.
	NULL -related functions	Most NULL -related functions are supported on FUJITSU Enterprise Postgres.
	Environment and identifier functions	USER is not supported on FUJITSU Enterprise Postgres. Implement the equivalent function by using an alternative function. Other environment and identifier functions are not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions.

Function		Description
	Other functions	The following functions are not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions. <ul style="list-style-type: none"> ▶ Character set functions ▶ Collation functions ▶ Large object functions ▶ Hierarchical functions ▶ Data mining functions
	Aggregate functions	Most aggregate functions are supported on FUJITSU Enterprise Postgres. However, conversion must account for the differences in return values and options.
	Analytic functions	Most analytic functions are supported on FUJITSU Enterprise Postgres. However, conversion must account for the differences in return values and options.
	Object reference functions	Object reference functions are not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions.
	Model functions	Model functions are not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions.
	OLAP functions	OLAP functions are not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions.
	Data cartridge functions	Data cartridge functions are not supported on FUJITSU Enterprise Postgres. Consider how to implement equivalent functions.
	User-defined functions	User-defined functions are supported on FUJITSU Enterprise Postgres. However, conversion must account for the differences such as syntax so that it can be run on FUITSU Enterprise Postgres.

Table 8-22 Other types of SQL statements

SQL statements		Description
Transaction Control Statements	COMMIT	Both databases support COMMIT statements. However, conversion must account for the differences in syntax.
	ROLLBACK	Both databases support ROLLBACK statements. However, conversion must account for the differences in syntax. If the TO SAVEPOINT clause is specified on the Oracle Database, use the ROLLBACK TO SAVEPOINT statement on FUJITSU Enterprise Postgres.
	SAVEPOINT	Both databases support SAVEPOINT statements. However, conversion must account for the differences in creating a save point with the same name of an existing save point.
	SET TRANSACTION	Both databases support SET TRANSACTION statements, but the function is different. If the ISOLATION LEVEL clause is specified on the Oracle Database, use the SET TRANSACTION statement on Fujitsu Enterprise Postgres to account for the differences in syntax. If other clauses are specified, consider how to implement the equivalent function.
	SET CONSTRAINT	Both databases support the SET CONSTRAINT statement function, but the statement name is different. Convert to SET CONSTRAINTS on FUJITSU Enterprise Postgres.

Session Control Statement	ALTER SESSION	The ALTER SESSION statement is not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent function.
	SET ROLE	Both databases support SET ROLE statements, but the function is different. Consider how to implement the equivalent function. SET ROLE statements on Oracle Database enable or disable a role for the current session. SET ROLE statements on Fujitsu Enterprise Postgres change the user identifier for the current session.
System Control Statement	Both databases support ALTER SYSTEM statements, but the function is different because of the differences in the database architecture. Consider how to implement the equivalent function.	
Operators	Most operators are supported on FUJITSU Enterprise Postgres. However, if Oracle Database uses the following operators, consider how to convert or implement the equivalent function: <ul style="list-style-type: none"> ▶ Hierarchical query operators such as PRIOR and CONNECT_BY_ROOT ▶ MINUS ▶ Multiset operators such as MULTISET, MULTISET EXCEPT, MULTISET INTERESCT, and MULTISET UNION Both databases support the concatenation operator ' '. However, conversion must account for the differences in behavior when specifying concatenated strings containing NULL .	
Expressions	Most expressions are supported on FUJITSU Enterprise Postgres. However, conversion must account for the differences in syntax and format of the returned value. The following expressions are not supported. Thus, consider how to implement equivalent functions. <ul style="list-style-type: none"> ▶ CURSOR expressions ▶ Model expressions ▶ Object access expressions ▶ Placeholder expressions ▶ Type constructor expressions 	
Conditions	Most conditions are supported on FUJITSU Enterprise Postgres. Comparison conditions are supported on FUJITSU Enterprise Postgres. However, an expression list for ANY , SOME , and ALL is not supported. Consider how to convert while accounting for the differences. The following expressions are not supported. Thus, consider how to implement the equivalent functions. <ul style="list-style-type: none"> ▶ '^=' used for inequality test ▶ Floating-point conditions such as IS [NOT] NAN and IS [NOT] INFINITE ▶ Model conditions such as IS ANY and IS PRESENT ▶ Multiset conditions such as IS A SET, IS EMPTY, MEMBER, and SUBMULTISET ▶ REGEXP_LIKE ▶ XML conditions such as EQUALS_PATH and UNDER_PATH ▶ IS OF type condition 	

Others	Identifier	An unquoted identifier behaves in different ways on an Oracle Database than on FUJITSU Enterprise Postgres. Quoting an identifier makes it case-sensitive, but unquoted identifiers are always folded to lowercase on FUJITSU Enterprise Postgres. Thus, convert in case-sensitive applications.
	Implicit Data Conversion	Both databases support implicit data conversion, but the function is different. The scope of implicit data conversion in Fujitsu Enterprise Postgres is smaller than Oracle Database. Consider how to implement the equivalent function by using alternatives such as explicit data conversion.
	Zero-length character value	Oracle Database handles zero-length character values as NULL , but Fujitsu Enterprise Postgres handles them as not NULL . If zero-length character values are used as NULL on an Oracle Database, consider how to implement the equivalent function on FUJITSU Enterprise Postgres.

– PL/SQL

Table 8-23 shows key considerations when migrating from Oracle Database PL/SQL to Fujitsu Enterprise Postgres PL/pgSQL.

Table 8-23 PL/SQL elements

PL/SQL elements		Description
Basic syntax	Block	Basic syntax elements of PL/SQL such as block, error handling, and variables are supported in PL/pgSQL on FUJITSU Enterprise Postgres.
	Error handling	
	Variables	
Data types	Scalar data types	For more information about SQL data types, see Table 8-18 on page 249. Both databases support the BOOLEAN data type. Other data types such as PLS_INTEGER and BINARY_INTEGER data types are not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions.
	Composite data types	Collection types are not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions, for example, by using a temporary table. Both databases support record variables. However, conversion must account for the minor differences.
Control statements	Condition selection statement	Both databases support IF and CASE statements.
	LOOP statement	Both databases support basic LOOP statements, WHILE LOOP statements, and FOR LOOP statements. However, conversion must account for the differences in the REVERSE clause of the FOR LOOP statement.
	GOTO statement	The GOTO statement is not supported on FUJITSU Enterprise Postgres. Consider how to implement the equivalent function on FUJITSU Enterprise Postgres.
Static SQL	Cursor	Both databases support cursors. However, conversion must account for the minor differences.
	Transaction processing and control	Both databases support COMMIT and ROLLBACK statements. However, conversion must account for the differences in transaction control specifications. For example, on FUJITSU Enterprise Postgres, in a block including error handling that uses the EXCEPTION clause, COMMIT and ROLLBACK statements return errors.

PL/SQL elements		Description
Dynamic SQL	EXECUTE IMMEDIATE statement	The EXECUTE IMMEDIATE statement on Oracle Database is equivalent to the EXECUTE statement on FUJITSU Enterprise Postgres.
	OPEN FOR statement	Both databases support OPEN FOR statements. However, conversion must account for the minor differences.
Subprograms		Subprograms are not supported on FUJITSU Enterprise Postgres. Thus, consider how to implement the equivalent functions.
Triggers		Both databases support triggers. However, conversion must account for the minor differences.
Packages		Packages are not supported on FUJITSU Enterprise Postgres. Thus, consider how to implement the equivalent functions.
Oracle Supplied PL/SQL packages		Some packages and procedures such as DBMS_OUTPUT, UTL_FILE, and DBMS_SQL are supported on FUJITSU Enterprise Postgres. However, conversion must account for the minor differences. Other Oracle supplied PL/SQL packages are not supported on FUJITSU Enterprise Postgres. Thus, consider how to implement the equivalent functions.

– Application

Some interfaces that are supported on Oracle Database are not supported on FUJITSU Enterprise Postgres. Even if the same interface is supported on both databases, API specifications might differ. Consider how to implement the equivalent functions on Fujitsu Enterprise Postgres while accounting for the differences in the interfaces.

Fujitsu Enterprise Postgres supports the following client interfaces.

- JDBC driver.
- ODBC driver.
- libpq - C Library.
- ECPG - Embedded SQL in C.

– Batch file for database operation

The specifications of operation commands that are written in batch files for database operations are widely different between Oracle Database and FUJITSU Enterprise Postgres. Consider how to implement the equivalent functions on FUJITSU Enterprise Postgres.

– Data

The basic steps of data migration are as follows.

- i. Extract data from the source database and store data in files.
- ii. Convert data formats that are stored in files in step i to the target database format.
- iii. Move and insert data that is converted in step ii to the target database.

When designing data migration, consider the following items:

- It is a best practice that you store data (step i on page 256) in CSV files. After data is extracted to the CSV files, it is possible to insert it by using the **COPY** statement in step iii on page 256. The **COPY** statement stores data efficiently.
- When using Oracle specific data types or external characters in an Oracle Database, consider how to extract the data.
- Pay attention to the presence of the header rows and **NULL** values when considering how to convert data formats.

► Implementation

In this process, assets in the source database systems are converted for the target database systems by following the migration approach that was created during the design process. These assets include SQL, PL/SQL, and batch files for database operations.

Also, data is migrated by following the approach that was created during the design process.

► Testing

After the implementation on the target system is complete, test to ensure that the system works as expected. It is a best practice to perform the following tests to determine the impact of database changes and assess whether the system meets your system functional and nonfunctional requirements:

– Schema verification

Verify that database objects in the source database such as tables, indexes, stored functions, and stored procedures are migrated to the target database without omission.

For example, Oracle Database obtains database object definitions that refer to system tables and system views, and Fujitsu Enterprise Postgres obtains referring system catalogs and system views. Compare the retrieved definitions to ensure that all required objects were migrated.

– Data verification

Verify that data is successfully migrated from the source database to the target database.

For example, output the table data of each database to files in the same format, such as CSV. Compare these files to confirm that there is no difference between the data before and after migration.

– Application-functional testing

Verify that the applications on the target system meet the functional requirements. Create the test case scenarios based on the functional requirements, complete all scenarios, and confirm whether the results are as expected.

– Performance testing

Verify that the target system meets performance requirements. Create the test case scenarios based on the performance requirements, complete all scenarios, and confirm whether the results are as expected.

If the target system does not meet the performance requirements, analyze the cause and resolve the issues. To resolve the performance issues, optimize the database parameters or change SQL statements as needed.

Note: For more information about performance tuning, see 8.3.2, “Performance tuning tips” on page 258.

– Operational testing

Make sure that the target system can operate the business as expected. Create the test case scenarios for each of the following steady operations, complete all scenarios, and confirm whether the results are as expected:

- Start and stop the database.
- Copy data as backups and manage them.
- Check disk usage and allocate free spaces by running **VACUUM** or rebuilding an index.
- Review the audit log information.
- Check the connection status. For example, check whether there is a connection that is connected for a long period or that occupies resources to prevent performance degradation.
- Patching.
- Switching, disconnection, and failback nodes for maintenance in a HA environment.

– Recovery testing

Make sure that business can be recovered and continue if there are abnormal problems in the target system. Create the test case scenarios for each of the following operations, complete all scenarios, and confirm whether business can be resumed immediately:

- Recovering from hardware failures, for example, disks and network equipment.
- Recovering data from backups.
- Processing during an abnormal operation of an application. For example, if there is a connection that occupies resources, disconnect to eliminate the waiting status.
- Processing while running out of disk space.
- Processing during a failover in a HA environment, which is called a failback.

8.3.2 Performance tuning tips

Enterprise systems often require high performance. Performance tuning is required to optimize the usage of resources, including CPU, memory, and disk.

This section outlines the tasks that are required for performance tuning.

Performance tuning can be organized into three perspectives: the optimal use of computer resources, minimizing I/O, and narrowing the search area. Each perspective is described in detail:

► Optimal use of computer resources

The key point is to consider the architecture of PostgreSQL itself. PostgreSQL uses a write-once architecture. To make effective use of this mechanism, the parameters of the configuration file `postgresql.conf` must be adjusted for computer resources, and resources must be distributed for optimization.

Details are provided in the following tables:

- Table 8-24 on page 259
- Table 8-25 on page 260
- Table 8-26 on page 260

► Minimizing I/O

The most important aspect of performance tuning is to reduce I/O activity that is associated with data refresh operations. Therefore, the database performs processing in memory as much as possible to improve performance. However, there are certain processes in PostgreSQL that are run to ensure the persistence of the data, such as **COMMIT** and checkpoint. **COMMIT** is the process where updates to the database are written to the disk and saved. Checkpoint is the process where data that is held in memory is written to the disk. Writing data in the memory to disks must be done efficiently or it might lead to various bottlenecks.

Details are provided in the following tables:

- Table 8-27 on page 260
- Table 8-28 on page 261
- Table 8-29 on page 261

► Narrowing the search area

The key is ensuring efficient data access by avoiding unnecessary processing and resource consumption when accessing data in the database. Specifically, SQL statements must be written and SQL must be run based on the latest statistics to perform data processing with minimum I/O processing.

Also, actively use mechanisms such as caching similar queries by using prepared statements and eliminating connection overhead with connection pooling.

Details are provided in the following tables:

- Table 8-30 on page 262
- Table 8-31 on page 262
- Table 8-32 on page 262

Table 8-24 Organizing table and index data and updating statistics

Task	Description
Tuning goal	Organize table and index data and update statistics.
Tuning objectives	<ol style="list-style-type: none"> 1. Prevent bloating of data files and increased I/O processing. 2. Prevent statistics from becoming stale and causing unnecessary I/O activity without proper execution planning.
Tuning method	<p>Consider setting up and implementing the following tasks regularly:</p> <ol style="list-style-type: none"> 1. Preventing data files from enlarging. <ul style="list-style-type: none"> – Reuse unnecessary area by VACUUM. – Remove unnecessary space with REINDEX. 2. Updating statistics to reduce unnecessary I/O activity by updating statistics with ANALYZE.
Process	Database configuration design and construction, and database operation design.
Activities	<p>In the database configuration design and construction process, consider including settings for performing autovacuuming to attend to VACUUM and ANALYZE concerns.</p> <p>REINDEX should be included in the database operations design to be implemented during maintenance work.</p>

Table 8-25 Accelerating SQL execution with parameter tuning

Task	Description
Tuning goal	Accelerate SQL execution with parameter tuning.
Tuning objectives	Adjust the parameters for the execution plan to increase the likelihood that a better execution plan is selected.
Tuning method	In <code>postgresql.conf</code> , adjust the parameters for work memory size, genetic query optimizer, planner's estimated costs, parallel processing, and conflict.
Process	Database configuration design and construction.
Activities	Design parameters during the database configuration design and construction process.

Table 8-26 Accelerating SQL execution with resource partitioning and avoiding locks

Task	Description
Tuning goal	Accelerate SQL execution with resource splitting and lock avoidance.
Tuning objectives	<ol style="list-style-type: none"> 1. Improve performance by using table spaces and partitioning to distribute or localize I/O processing. To localize I/O processing, use partition pruning to limit the partition tables that are targeted in the search. 2. Reduce the frequency of conflicts by, for example, splitting transactions to avoid lock conflicts.
Tuning method	<ol style="list-style-type: none"> 1. Leverage table spaces and partitioning. Tune SQL to use partition pruning. 2. Review application logic to shorten the database resource lock duration, such as by splitting transactions.
Process	Database configuration design and construction, and application design.
Activities	<ol style="list-style-type: none"> 1. In the database configuration design and construction process, consider using table spaces and partitioning to reduce I/O processing by their distribution or localization. In application design, consider using partition pruning in SQL tuning. 2. When designing an application, consider using the transaction unit to avoid lock contention.

Table 8-27 Parameter tuning for write assurance

Task	Description
Tuning goal	Tuning parameters to ensure writes.
Tuning objectives	Optimize I/O processing by tuning various buffer sizes, such as <code>shared_buffer</code> and <code>wal_buffers</code> , and parameters that are related to WAL and checkpoints.
Tuning method	Decide the values of various buffer sizes, including <code>shared_buffer</code> , <code>wal_buffers</code> , <code>checkpoint_timeout</code> , and <code>max_wal_size</code> and parameters for WAL and checkpoint based on the hardware specifications and description of the operation.

Task	Description
Process	Database configuration design and construction.
Activities	Decide on the appropriate values in the database configuration design and construction process based on hardware specifications and a description of the operation.

Table 8-28 Suppressing index updates

Task	Description
Tuning goal	Suppress index updates.
Tuning objectives	Updating a tuple in a table might require updating an index. To update an index, add new data to the index. This process is expensive because it is done while maintaining data relationships. Performance can be improved by ensuring that index updates do not occur.
Tuning method	<ol style="list-style-type: none"> 1. When inserting bulk data, temporarily remove indexes and create them later one at a time. 2. If there are many updates, apply a specification that provides a margin to the data storage area (FILLFACTOR). This process can streamline processing by eliminating the need for index updates.
Process	Database configuration design and construction, and database operation design.
Activities	<ol style="list-style-type: none"> 1. Consider the timing of indexing during the database operation design. 2. The value of FILLFACTOR is determined based on the description of the operation during database configuration design and construction.

Table 8-29 Storing large amounts of data in bulk and in parallel

Task	Description
Tuning goal	Large amounts of data are stored in bulk and in parallel.
Tuning objectives	Like multi-core CPUs, hardware is designed to provide maximum performance when multiple demands are made simultaneously. For this reason, when storing large amounts of data, storing each piece of data one by one does not leverage the hardware capabilities. It is beneficial to do this kind of work in bulk and in parallel to improve performance by leveraging the specifications of the hardware.
Tuning method	Store data by using a COPY command and running multiple executions in parallel.
Process	Database operation design.
Activities	During database operation design, consider the approach to storing data, and apply the method of storing it in bulk and in parallel where possible.

Table 8-30 Leveraging indexes

Task	Description
Tuning goal	Leveraging indexes.
Tuning objectives	<ol style="list-style-type: none"> 1. Replace sort and scan processes with indexes to improve SQL performance. 2. Improve performance by using indexes more efficiently.
Tuning method	<ol style="list-style-type: none"> 1. Replace processes: <ul style="list-style-type: none"> – Use index scans instead of the sort process. – Use the Index Only Scan search method. 2. Improve efficiency by using indexes: <ul style="list-style-type: none"> – Composite indexes are described in the order in which they can be filtered. – The OR condition is leveraged. – Use expression indexes.
Process	Database configuration design and construction.
Activities	Consider the indexes to use during database configuration design and construction.

Table 8-31 Evaluating and controlling execution plans

Task	Description
Tuning goal	Evaluate and control execution plans.
Tuning objectives	The performance of SQL varies greatly depending on the action plans. Ensuring that the action plan does not fluctuate is critical.
Tuning method	The default is to run ANALYZE to continuously refresh statistics because suboptimal execution plans might be created based on stale statistics. However, whenever an efficient execution plan is not selected, use <code>pg_hint_plan</code> as necessary for better control. Alternatively, stabilize performance by fixing statistics with <code>pg_dbms_stats</code> so that the execution plan does not change.
Process	Database configuration design and construction, and application design.
Activities	Consider the <code>pg_hint_plan</code> and <code>pg_dbms_stats</code> settings during database configuration design and construction and application design.

Table 8-32 Optimizing iterations and communication processing

Task	Description
Tuning goal	Optimize iterations and communication processing.
Tuning objectives	<ol style="list-style-type: none"> 1. Improve performance by caching iterations. 2. Reduce the data amount and the number of times of communication.
Tuning method	<ol style="list-style-type: none"> 1. Cache repetitive processing by using prepared statements and connection pooling. 2. Reduce the amount of data and the number of times of communication by using user-defined functions. Adjust the fetch size and reduce the number of communications.

Task	Description
Process	Database configuration design and construction, and application design.
Activities	During database configuration design and construction and application design, consider using prepared statements and connection pooling. Also, incorporate user-defined functions and fetch sizes.

8.4 Use cases: Migration from an Oracle Database system

This section explains the target system architecture (see 8.4.1, “Target system architecture” on page 263) and provides specific instructions for migrating from Oracle Database to Fujitsu Enterprise Postgres as two use cases:

- ▶ Migration scenario for large-scale legacy systems
- ▶ Migration scenario for small and medium-scale systems, which can be used for internal business

We describe the features of these two use cases and the concepts of migration. Also, we describe how to determine the target platform when migrating database systems.

8.4.1 Target system architecture

In this book, we refer to the target system architecture at three different levels of system virtualization, which are shown in Figure 8-11.

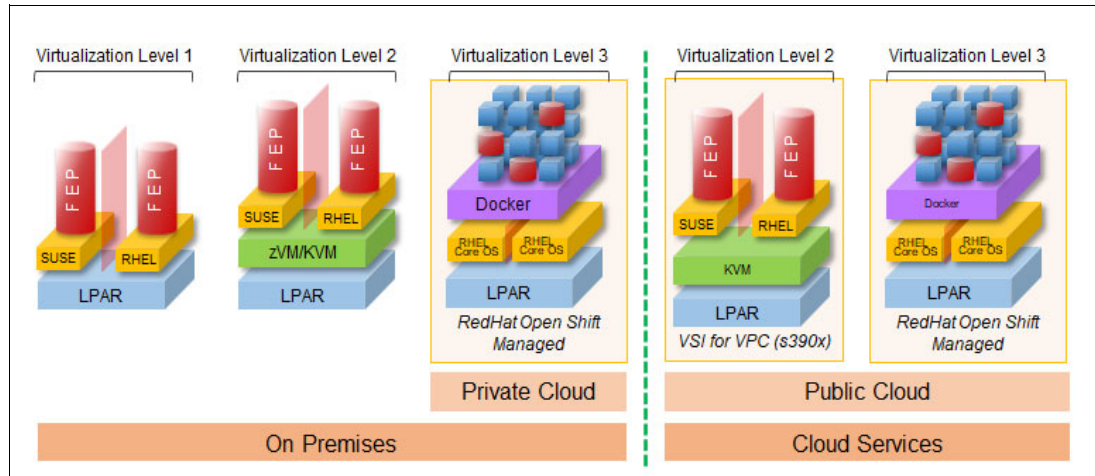


Figure 8-11 Target system architecture

- ▶ Virtualization level 1

Applications run on operating systems (OSs) that are directly hosted by server machines.

- ▶ Virtualization level 2

The infrastructure layer is virtualized. Historically, with the improvement of server machine processing power and capacity, an abstraction layer, which is known as hypervisors, were introduced. Hypervisors run between the physical hardware and virtualized machines to allow applications to efficiently use compute resources of one machine. The abstraction is realized by software, firmware, or hardware. Each virtual machine (VM) runs its own guest OS. In virtualization level 2, the applications are abstracted from the infrastructure layer.

- ▶ Virtualization level 3

The OS layer is virtualized. Applications share the OS kernel of the host system, which resolves the concerns of excess overhead in memory and storage allocation that is required for VMs. In virtualization level 3, the applications are abstracted from the OS and infrastructure layers.

Along with the virtualization of underlying layers, containerization technology can package a single application and its dependencies into a lightweight “container” to run on any OS. The advancement of container orchestration and platforms promote the adoption of containerized applications.

In the following sections, we describe the general characteristics and the migration of legacy systems, which are divided into two categories:

- ▶ Migration scenario for large-scale legacy systems

These systems are often migrated to a traditional on-premises environment with virtualization level 1 or 2.

- ▶ Migration scenario for small and medium-scale systems

These systems are often migrated to a private cloud with virtualization level 3 or a public cloud with virtualization level 3.

8.4.2 Planning and designing your migration journey

Regardless of the scale of the system and the choice of the target system environment, the modernization that migration brings to organizations has three major benefits:

- ▶ System maintenance costs are reduced by moving away from legacy hardware that uses outdated equipment and skills.
- ▶ Migrating from proprietary software to open source software reduces software license costs.
- ▶ The total cost of ownership (TCO) of future systems is reduced by building an open ecosystem.

Modernization should be strategically planned and carried out to achieve DX initiatives with maintainability and extensibility.

Fujitsu Enterprise Postgres is a database with an open interface that is based on open source PostgreSQL with extra enhanced features, such as security and HA and other features that can be used in enterprise systems. It also provides flexibility for several environments. Therefore, it can be used in either use case. In addition, there are also benefits in cost savings because HA features and security strengthening features are available as standard features.

How to determine the target platform

The target platform can realize cost savings by leveraging container technology. As a best practice, move to a container environment first because of the following advantages:

- ▶ Leverage the latest technology toward DX.
- ▶ Reduce future migration costs by leveraging hybrid clouds.
- ▶ Cost savings in operations by using automated operations.

However, the system requirements vary, so migration to a container environment might not be possible. In some cases, a traditional on-premises environment with virtualization level 1 or 2 might be best to meet the system requirements. Therefore, identify the characteristics of the systems first; decide whether migration to a container environment is possible; and then decide whether migration to a traditional on-premises environment with virtualization level 1 or 2 is wanted to determine the appropriate target system platform.

Note: For more information about container technology, contact a FUJITSU representative at the following website:

<https://www.fast.fujitsu.com/contact>

Designing the migration journey

Depending on the specific requirements of each organization, the preferred choice of platform and cloud types vary.

Section 8.4.3, “Migration scenario for large-scale legacy systems” on page 266 describes a use case of migration from large enterprise systems on Oracle RAC to Fujitsu Enterprise Postgres on a traditional on-premises environment with virtualization level 2. The test cases that are used to explain this migration is conducted on the lab environment that was set up for this book, which simulates this use case.

Section 8.4.4, “Migration scenario for small and medium-scale systems” on page 279 describes a use case of migration from small to medium-sized enterprise systems on Oracle RAC to Fujitsu Enterprise Postgres on a containerized environment with virtualization level 3. The test cases that are used to explain this migration is conducted in a lab environment that was set up for this publication, which simulates this use case.

The options that are available for the target architecture of a migration is not limited to the ones that are presented in this book. For example, to offload hardware maintenance in the target database system, some organizations might prefer to move from legacy database systems on private cloud environments to a public cloud while the database software is kept in a non-containerized form.

At the time of writing, LinuxONE is available on-premises, on a private cloud, and on IBM public cloud on z/VM.

For an experience-based look at the installation of Red Hat OpenShift Container Platform (RHOCP) on IBM LinuxONE, see [Red Hat OpenShift Installation Process Experiences on IBM Z and IBM LinuxONE](#). Platform and cloud types can be mixed and matched to cater to the nature of different database systems and workloads.

To further enhance the scalability and manageability of containerized systems, IBM offers IBM Cloud Pak for Multi Cloud Management, which offers a management layer of multiple Red Hat OpenShift clusters across private cloud and IBM public cloud.

For more information or assistance with designing the migration journey that best achieves your organization's goals, contact your IBM or Fujitsu customer service representative.

8.4.3 Migration scenario for large-scale legacy systems

This section describes the requirements for migrating large-scale Oracle legacy systems to Fujitsu Enterprise Postgres on a traditional on-premises environment with virtualization level 1 or 2 and the migration procedures in an actual case. You look at step-by-step procedures, an example of PL/SQL, which is commonly used in legacy systems, and an example of partitioning tables, which are frequently used when dealing with large amounts of data.

► **Characteristics**

Large-scale legacy systems are used for mission-critical tasks and to manage critical data within the company. Therefore, these systems have strict requirements for response time in online transaction processing, processing time for batch jobs running at night, and the downtime of a failover. In addition, the current go-live systems need higher maintenance cost and software licensing fees because they often run on older hardware such as mainframes.

► **Key considerations for migration**

It is a best practice to safely migrate to a more mature and proven platform because performance requirements are valued. Therefore, it is reasonable to migrate to a traditional on-premises environment with virtualization level 1 or 2. In addition, adopting open-source software avoids an increase in maintenance costs and software licensing fees.

Requirements for the target system

The source system is a mission-critical system and often has strict requirements for HA, strengthened security, and high performance. Therefore, the target system also must meet these requirements. The combination of Fujitsu Enterprise Postgres and IBM LinuxONE fulfills more requirements.

For more information about these requirements, see the following sections.

- 8.2.1, "Business continuity" on page 215
- 8.2.2, "Mitigating security threats" on page 217
- 8.2.3, "SQL performance tuning" on page 221

System environment

The examples that are shown in this section are validated on the following systems. Both systems use servers with the same specifications.

- **Source system**
 - OS: Red Hat Enterprise Linux
 - Database: Oracle Database 19c
- **Target system**
 - OS: Red Hat Enterprise Linux
 - Database: Fujitsu Enterprise Postgres 13

Migration scope and tools

The examples that are shown in this section assume online transaction processing and a batch job workload.

Migration scope

The migration scope includes DDL, data, and applications in an Oracle Database (PL/SQL):

► **Online processing and table data**

As an example of online transactional processing, we use TPC Benchmark C (TPC-C), which portrays the activity of a wholesale supplier such as ordering and payment. This online processing performs consecutive processing of ordering, payment, order status checks, delivery, and inventory checks.

In addition to online processing, we should migrate the table data that is used in the processing. The data size that is used for this example is ~1.5 GB.

► **Batch job: Daily processing for aggregating business data**

We created a batch job for the business process by using TPC-C databases for this example. The batch job was created by using stored procedures of PL/SQL. This process assumes a closing operation for daily daytime operations. This processing aggregates the number of orders and sales for each item for the day and inserts the aggregated data into the `daily_sales` table. This table is defined as a quarterly partitioned table. The table definition is shown in Example 8-14.

Example 8-14 Table definition of `daily_sales` in Oracle Database

```
CREATE TABLE daily_sales (
  entry_date DATE,
  i_id NUMBER(6, 0),
  total_quantity CHAR(10),
  bussiness_form BLOB
)
PARTITION BY RANGE(entry_date)
(
  PARTITION daily_sales_2021_1q VALUES LESS THAN
(TO_DATE('2021/07/01','YYYY/MM/DD'))
  TABLESPACE XEPDB1SPC01
  , PARTITION daily_sales_2021_2q VALUES LESS THAN
(TO_DATE('2021/10/01','YYYY/MM/DD'))
  TABLESPACE XEPDB1SPC01
  , PARTITION daily_sales_2021_3q VALUES LESS THAN
(TO_DATE('2022/01/01','YYYY/MM/DD'))
  TABLESPACE XEPDB1SPC01
  , PARTITION daily_sales_2021_4q VALUES LESS THAN
(TO_DATE('2022/04/01','YYYY/MM/DD'))
  TABLESPACE XEPDB1SPC01
);
```

Note: In this example, we specify 'YYYY/MM/DD' for the date format.

► **Batch job: Viewing aggregated data**

This batch job is used to view the data that as aggregated in “Migration scope”. It was created by using the stored function of PL/SQL. It retrieves and returns data, such as the sales amount for each item that was ordered on the specified date, and data such as the `daily_sales` table.

Migration tools

In this section, we describe the tools that are used in the migration. Ora2Pg is used for DDL and data migration, which is one of the migration tools that is used when migrating from Oracle Database to PostgreSQL.

► Source of the tool

This tool is open-source software and is available at no charge. It adopts the GPL license. Comply with the license policy when using it.

Note: For more information about and to download Ora2Pg, see [Ora2Pg](#).

► Required settings in `ora2pg.conf`

For prerequisites, ensure that the following settings are correct:

- Database connection settings
- Target data type setting

Set `PG_NUMERIC_TYPE` to 0. This parameter determines whether data of the numeric data type is converted to numeric or real/double precision. In this example, the TPC-C model requires highly accurate data, so any `NUMBER` data type on the Oracle Database should be converted to the numeric data type on FUJITSU Enterprise Postgres.

Migration steps for a typical OLTP processing and table data scenario

In this section, we introduce the steps to migrate DDL and table data according to the migration process that is described in 8.3.1, “Experience-based migration technical knowledge” on page 231. We assume that we already know where changes will be made and determined how to convert them during migration.

► DDL migration

We show the steps to migrate DDL when using Ora2Pg and running on the Oracle host machine.

In this example, a partitioned table is used to demonstrate DDL migration.

► Table data migration

Ora2Pg can also be used for migrating the data that is stored in the table. As a best practice, use test data to confirm that the scripts work before using them on live data. In general, table data may be migrated to a new system at a different point (before go live). To simplify the steps, DDL and table data are migrated simultaneously in this example.

In our example, Ora2Pg is installed on the Oracle database host server.

Because Ora2Pg does not have extract, transform, and load (ETL) or CDC functions, we cannot ensure data integrity when retrieving data from a running database. Therefore, we choose a simple migration method that disconnects all client connections to the Oracle Database and migrate the DDL and table data.

Note: If you need help when using a solution that uses ETL or CDC, contact a Fujitsu representative at <https://www.postgresql.fastware.com/contact>.

Complete the following steps:

1. Retrieve the DDL from the Oracle database by running **ora2pg**.

Retrieve the DDL of tables and DDL of child tables of the partitions as assets for migration, as shown in Example 8-15.

Example 8-15 Retrieving the DDL with Ora2Pg

```
[oracle@RDBKPGX1 ora2pg]$ ora2pg -c ora2pg.conf -j 4 -t TABLE -o
ora2pg_table.sql
[=====>] 10/10 tables (100.0%) end of scanning.
Retrieving table partitioning information...
[=====>] 10/10 tables (100.0%) end of table export.
[oracle@RDBKPGX1 ora2pg]$ ora2pg -c ora2pg.conf -t PARTITION -o
ora2pg_table_partition.sql
[=====>] 4/4 partitions (100.0%) end of output.
```

2. Retrieve the table data from Oracle Database by running **ora2pg**.

Retrieve the table data from Oracle Database by running **ora2pg**. Either **INSERT** or **COPY** statements can be specified as the load mechanism for the retrieved data. Using the **COPY** statement is a best practice when large amounts of data must be loaded, as shown in Example 8-16.

Example 8-16 Retrieving table data with Ora2Pg

```
[oracle@RDBKPGX1 ora2pg]$ ora2pg -c ora2pg.conf -j 4 -t COPY -o
ora2pg_copy.sql
[=====>] 10/10 tables (100.0%) end of scanning.
[=====>] 480000/480000 rows (100.0%) Table CUSTOMER (19
sec., 25263 recs/sec)
[> ] 0/1 rows (0.0%) Table DAILY_SALES_2021_1Q (0 sec.,
0 recs/sec)
[> ] 0/1 rows (0.0%) Table DAILY_SALES_2021_4Q (1 sec.,
0 recs/sec)
[=====>] 100000/1 rows (1000000.0%) Table
DAILY_SALES_2021_3Q (36 sec., 2777 recs/sec)
[> ] 0/1 rows (0.0%) Table DAILY_SALES_2021_2Q (0 sec.,
0 recs/sec)
[=====>] 160/160 rows (100.0%) Table DISTRICT (0 sec., 160
recs/sec)
[=====>] 494655/480000 rows (103.1%) Table HISTORY (8 sec.,
61831 recs/sec)
[=====>] 100000/100000 rows (100.0%) Table ITEM (2 sec.,
50000 recs/sec)
[=====>] 143845/144000 rows (99.9%) Table NEW_ORDERS (1
sec., 143845 recs/sec)
[=====>] 494505/480000 rows (103.0%) Table ORDERS (8 sec.,
61813 recs/sec)
[=====>] 4945878/4801507 rows (103.0%) Table ORDER_LINE (82
sec., 60315 recs/sec)
[=====>] 1600000/1600000 rows (100.0%) Table STOCK (45 sec.,
35555 recs/sec)
[=====>] 16/16 rows (100.0%) Table WAREHOUSE (0 sec., 16
recs/sec)
[=====>] 8359059/8085684 rows (103.4%) on total estimated
data (222 sec., avg: 37653 tuples/sec)
```

3. Divide the DDLs that were retrieved in step 1 on page 269 into “DDL before inserting data” and “DDL after inserting data”.

Generally in a table with foreign key constraints, it is necessary to determine the order in which data is inserted and consider the constraint conditions, which can be a complicated process. However, in database migration, the procedure can be simplified because the table data in the source database already satisfies foreign key constraints. In this example, insert all the data without any foreign key constraints that are defined for the target table, and then define foreign key constraints after data is loaded.

To follow this procedure, divide the DDLs of the table definition that were retrieved in step 1 on page 269 into two parts: DDLs before inserting data (without a definition of foreign key constraints), and DDLs after inserting data (the definition of foreign key constraints) by using the **ALTER TABLE** statement.

Also, divide the DDL defining indexes in the same way as for the table definition to reduce the time that is required to load data.

4. Run the “DDL before inserting data” scripts in Fujitsu Enterprise Postgres by using the **psql** utility to define tables, as shown in Example 8-17.

Example 8-17 Running “DDL before inserting data”

```
[fsepuser@rdbkpgr1 ora2pg]$ psql -d tpcc -f ora2pg_table_create.sql
SET
CREATE TABLE
...
[fsepuser@rdbkpgr1 ora2pg]$ psql -d tpcc -f ora2pg_table_partition.sql
SET
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

5. Insert the table data that was retrieved in step 2 on page 269 into FUJITSU Enterprise Postgres.

The table data that was retrieved in step 2 on page 269 is inserted into Fujitsu Enterprise Postgres by using the **psql** utility to run the script, as shown in Example 8-18.

Example 8-18 Inserting table data into the target database

```
[fsepuser@rdbkpgr1 ora2pg]$ psql -d tpcc -f ora2pg_copy.sql
BEGIN
COPY 10000
COPY 10000
...
```

6. Run “DDL after inserting data” in FUJITSU Enterprise Postgres.

After the data is loaded, run the “DDL after inserting data” script (Example 8-19) in Fujitsu Enterprise Postgres by using the **psql** utility to define foreign key constraints and indexes.

Example 8-19 Running “DDL after inserting data”

```
[fsepuser@rdbkpgr1 ora2pg]$ psql -d tpcc -f ora2pg_table_alt.sql
SET
ALTER TABLE
...
```

7. Update statistics.

As a best practice, run the **ANALYZE** command to update the statistics that are associated with the table, especially when a large amount of data was inserted. Not doing so can result in poor query plans. We used the command that is shown in Example 8-20.

Example 8-20 Updating statistics by using ANALYZE

```
[fsepuser@rdbkpg1 ora2pg]$ psql -d tpcc -c "analyze"
ANALYZE
```

Migration steps of a typical batch job scenario

In this section, we introduce the steps to migrate the PL/SQL that is used in Oracle Database batch jobs by using the migration process that is described in 8.3.1, “Experience-based migration technical knowledge” on page 231. In this example, batch job source codes are converted manually without using tools. We assume that we know where changes will be made and have determined how to convert them during migration.

In this example, there are two PL/SQL batch jobs to be migrated:

- ▶ Procedure for aggregating business data daily
- ▶ Function to display aggregated data

Note: The numbers that are used in the annotations (such as “Step 1” and “Step 2”) in the examples indicate the migration step numbers that are described in “Procedure for aggregating business data daily” on page 271.

Procedure for aggregating business data daily

This procedure is intended to be performed as a daily closing operation. Its purpose is to aggregate the number of orders per product that is ordered on that day and insert the aggregated result into a dedicated table that is named `daily_sales`. The binary column is used to store PDF versions of the sales slip with the relational data when it is inserted into the table. The sole argument of this procedure is the date on which to perform the calculation.

The procedure that we used in this book is shown in Example 8-21.

Example 8-21 Procedure definition in Oracle Database PL/SQL

```
CREATE OR REPLACE PROCEDURE daily_sales_calc (calc_day DATE)
IS --Step 1
  target_day DATE;--Step 2
  TYPE daily_sales_rec IS RECORD (i_id NUMBER(6, 0), quantity CHAR(10)); --Step 3
  var_rec daily_sales_rec;
  wk_blob BLOB;
  wk_bfile BFILE;
-- Group the orders for the dates that are specified as arguments by product ID,
and retrieve the number of orders per product
  CURSOR c1 IS SELECT s.s_i_id AS i_id, sum(oo.quantity) AS quantity--Step 4
                FROM (SELECT ol.ol_i_id AS i_id, ol.ol_supply_w_id as s_w_id,
                            ol.ol_quantity AS quantity
                FROM orders o, order_line ol
                WHERE   ol.ol_o_id = o.o_id AND
                       ol.ol_d_id = o.o_d_id AND
                       ol.ol_w_id = o.o_w_id AND
                       TRUNC(o.o_entry_d, 'DD') = target_day
                ) oo,
  stock s
```

```

        WHERE
            s.s_i_id = oo.i_id AND
            s.s_w_id = oo.s_w_id
        GROUP BY s.s_i_id ;
BEGIN
    target_day := TRUNC(calc_day, 'DD');--Step 5
    -- Delete any previous data
    DELETE FROM daily_sales WHERE TRUNC(entry_date, 'DD') = target_day;
    COMMIT;
    -- Insert aggregated records in daily_sales table
    FOR var_rec IN c1 LOOP
        INSERT INTO daily_sales VALUES
            (target_day, var_rec.i_id, var_rec.quantity, empty_blob());--Step 6
        -- Store sales slip data
        SELECT bussiness_form INTO wk_blob FROM daily_sales WHERE entry_date =
target_day AND i_id = var_rec.i_id FOR UPDATE;
        wk_bfile := BFILENAME('FILE_DIR', var_rec.i_id || '.pdf');
        DBMS_LOB.FILEOPEN(wk_bfile, DBMS_LOB.FILE_READONLY);
        DBMS_LOB.LOADFROMFILE( wk_blob, wk_bfile, DBMS_LOB.GETLENGTH( wk_bfile));
        DBMS_LOB.FILECLOSE(wk_bfile);
    END LOOP;
    COMMIT;
END;
/

```

The following steps describe how to convert this procedure in Oracle Database PL/SQL, as shown in Example 8-21 on page 271, to Fujitsu Enterprise Postgres PL/pgSQL:

1. Change the syntax of **CREATE PROCEDURE**.

Change the syntax of the **CREATE PROCEDURE** statement to accommodate the differences between Oracle and FUJITSU Enterprise Postgres. Table 8-33 shows the differences between the two procedures.

Table 8-33 The syntax of **CREATE PROCEDURE**

Oracle Database	FUJITSU Enterprise Postgres
<pre> CREATE OR REPLACE PROCEDURE daily_sales_calc (...) IS ... BEGIN ... END; </pre>	<pre> CREATE OR REPLACE PROCEDURE daily_sales_calc (...) LANGUAGE plpgsql AS \$\$ DECLARE ... BEGIN ... END; / </pre>

2. Convert the data type.

Convert the data type because the data types that are available in Oracle Database and Fujitsu Enterprise Postgres are different. Table 8-34 on page 273 shows the differences between the two conversions.

Table 8-34 Example of converting data types

Oracle Database	FUJITSU Enterprise Postgres
DATE	Timestamp without time zone
BLOB	bytes
NUMBER(6, 0)	int

3. Change the location where the record type is defined.

The record type can be defined in the declarative part that is specified as **DECLARE** in PL/SQL, but it cannot be defined in declarations in PL/pgSQL. You must define separately the declarations in PL/pgSQL by using the **CREATE TYPE** statement, as shown in Table 8-35.

Table 8-35 Changing the definition of the record type

Oracle Database	FUJITSU Enterprise Postgres
<code>TYPE daily_sales_rec IS RECORD (i_id NUMBER(6, 0), quantity CHAR(10));</code>	<code>CREATE TYPE daily_sales_rec AS? (i_id int, quantity char(10));</code>

4. Change the definition of **CURSOR**.

PL/SQL and PL/pgSQL define **CURSOR** differently, so you must change the definition. Table 8-36 shows the differences between the two record type definitions.

Table 8-36 Changing the definition of the record type

Oracle Database	FUJITSU Enterprise Postgres
<code>CURSOR c1 IS SELECT ...</code>	<code>c1 CURSOR FOR SELECT ...</code>

5. Convert the function.

Because the functions that are available in Oracle Database and Fujitsu Enterprise Postgres are different, you must convert the functions. Table 8-37 shows the differences between the two platforms.

Table 8-37 Example of converting the function

Oracle Database	FUJITSU Enterprise Postgres
<code>TRUNC(calc_day, 'DD');</code>	<code>date_trunc('day', calc_day);</code>

6. Change how binary data is handled.

Oracle Database and Fujitsu Enterprise Postgres handle binary data differently. Therefore, change the processing of handling binary data to meet the Fujitsu Enterprise Postgres specifications. The differences in handling binary data are shown in Table 8-38.

Table 8-38 How to handle binary data

Oracle Database	FUJITSU Enterprise Postgres
<pre>INSERT INTO daily_sales VALUES (target_day, var_rec.i_id, var_rec.quantity, empty_blob()); SELECT bussiness_form INTO wk_blob FROM daily_sales WHERE entry_date = target_day and i_id = var_rec.i_id FOR UPDATE; wk_bfile := BFILENAME('FILE_DIR', var_rec.i_id '.pdf'); DBMS_LOB.FILEOPEN(wk_bfile, DBMS_LOB.FILE_READONLY); DBMS_LOB.LOADFROMFILE(wk_blob, wk_bfile, DBMS_LOB.GETLENGTH(wk_bfile)); DBMS_LOB.FILECLOSE(wk_bfile);</pre>	<pre>INSERT INTO daily_sales VALUES (target_day, var_rec.i_id, var_rec.quantity, pg_read_binary_file('/tmp/data/' var_rec.i_id '.pdf'));</pre>

Following these steps facilitates the migration of the procedure from PL/SQL to PL/pgSQL, as shown in Example 8-22.

Example 8-22 Procedure definition in Fujitsu Enterprise Postgres PL/pgSQL

```
CREATE TYPE daily_sales_rec AS?
(
  i_id int,
  quantity char(10)
);
CREATE OR REPLACE PROCEDURE daily_sales_calc (calc_day timestamp without time
zone)
LANGUAGE plpgsql
AS $$
DECLARE
  target_day timestamp without time zone;
  var_rec daily_sales_rec;
-- Group the orders for the dates that are specified as arguments by product ID,
and retrieve the number of orders per product
  c1 CURSOR FOR SELECT s.s_i_id AS i_id, sum(oo.quantity) AS quantity
                FROM (SELECT ol.ol_i_id AS i_id, ol.ol_supply_w_id AS s_w_id,
                    ol.ol_quantity AS quantity
                FROM orders o, order_line ol
                WHERE  ol.ol_o_id = o.o_id AND
                    ol.ol_d_id = o.o_d_id AND
                    ol.ol_w_id = o.o_w_id AND
                    date_trunc('day', o.o_entry_d) = target_day
                ) oo,
                stock s
  WHERE
    s.s_i_id = oo.i_id AND
    s.s_w_id = oo.s_w_id
  GROUP BY s.s_i_id;
```

```

BEGIN
    target_day := date_trunc('day', calc_day);
-- Delete any previous data
    DELETE FROM daily_sales WHERE date_trunc('day', entry_date) = target_day;
    COMMIT;
-- Insert aggregated records including sales slip data into daily_sales table
    FOR var_rec IN c1 LOOP
        INSERT INTO daily_sales VALUES
            (target_day, var_rec.i_id, var_rec.quantity,
pg_read_binary_file('/tmp/data/' || var_rec.i_id || '.pdf'));
    END LOOP;
    COMMIT;
END;
$$;

```

Function to display aggregated data

This function that is shown in Example 8-23 displays the number of orders and sales amounts for each item that was ordered on the date that is specified in the argument.

Example 8-23 Function definition in Oracle Database PL/SQL

```

-- Define the types of records and tables that are used for returning value in function
CREATE OR REPLACE TYPE daily_sales_rec IS OBJECT--Step 2
(
    day char(8),
    i_name varchar(24),
    quantity NUMBER, --Step 3
    sales NUMBER
);
CREATE OR REPLACE TYPE table_daily_sales_rec IS TABLE OF daily_sales_rec; --Step 2
-- Define a function that retrieves data such as sales per item on the specified date
CREATE OR REPLACE FUNCTION daily_sales_data (getday IN DATE)
RETURN table_daily_sales_rec PIPELINED
IS --Step 1
    -- Extract the sales results for each item on the date that is specified as arguments
    CURSOR curs IS SELECT to_char(s.entry_date, 'YYYYMMDD') AS day, i.i_name, --Step 4
        nvl(s.total_quantity, 0) AS total_quantity,
        nvl(s.total_quantity, 0) * i.i_price AS total_sales --Step 5
    FROM item i, daily_sales s
    WHERE i.i_id = s.i_id(+) AND--Step 6
        TRUNC(s.entry_date, 'DD') = TRUNC(getday, 'DD');
        --Step 7
BEGIN
    -- Return the extracted data row by row
    FOR row IN curs LOOP
        PIPE ROW(daily_sales_rec(row.day, row.i_name, row.total_quantity, row.total_sales));
--Step 8
    END LOOP;
    RETURN;
END;
/

```

The following steps describe how to convert this function in Oracle Database PL/SQL (Example 8-23 on page 275) to Fujitsu Enterprise Postgres PL/pgSQL:

1. Change the syntax of **CREATE FUNCTION**.

Change the syntax of the **CREATE FUNCTION** statement because it is different between Oracle Database and Fujitsu Enterprise Postgres (Table 8-39).

Table 8-39 The syntax of **CREATE FUNCTION**

Oracle Database	FUJITSU Enterprise Postgres
<pre>CREATE OR REPLACE FUNCTION daily_sales_data (...) RETURN ... IS ... BEGIN ... END; /</pre>	<pre>CREATE OR REPLACE FUNCTION daily_sales_data (...) RETURNS ... AS \$\$ DECLARE ... BEGIN ... END; \$\$ LANGUAGE plpgsql;</pre>

2. Migrate the collection types.

Fujitsu Enterprise Postgres does not support **IS TABLE OF**, which is one of the collection types, or **OBJECT**, which is used for defining the type. Use **CREATE TYPE** as a substitute for **OBJECT**. Use the **SETOF** modifier in the **CREATE FUNCTION** statement as a substitute for **IS TABLE OF** (Table 8-40).

Table 8-40 Migrating the data type

Oracle Database	FUJITSU Enterprise Postgres
<pre>CREATE OR REPLACE TYPE daily_sales_rec IS OBJECT (day char(8), i_name varchar(24), quantity NUMBER, sales NUMBER); CREATE OR REPLACE TYPE table_daily_sales_rec IS TABLE OF daily_sales_rec; CREATE OR REPLACE FUNCTION daily_sales_data (...) RETURN table_daily_sales_rec PIPELINED IS</pre>	<pre>CREATE TYPE table_daily_sales_rec AS (day char(8), i_name varchar(24), quantity numeric, sales numeric); CREATE OR REPLACE FUNCTION daily_sales_data (...) RETURNS SETOF table_daily_sales_rec AS \$\$</pre>

3. Change the data types.

Convert the data type because the data types that are available in Oracle Database and Fujitsu Enterprise Postgres are different (Table 8-41).

Table 8-41 Example of converting data types

Oracle Database	FUJITSU Enterprise Postgres
DATE	Timestamp without time zone
NUMBER	numeric

4. Change the definition of **CURSOR**.

Change the **CURSOR** definition as described in step 4 on page 273.

5. Add explicit type conversion (Table 8-42).

Perform data type conversion because the specification of implicit data conversion is different between Oracle Database and FUJITSU Enterprise Postgres. For example, Oracle Database can convert the data type implicitly and calculate the data even if the numeric data is stored in the CHAR data type column. However, Fujitsu Enterprise Postgres cannot convert the data type implicitly, so you must convert CHAR to the numeric data type explicitly and perform the calculation.

Table 8-42 Adding an explicit type conversion

Oracle Database	FUJITSU Enterprise Postgres
<pre>SELECT to_char(s.entry_date, 'YYYYMMDD') AS day, i.i_name, nvl(s.total_quantity, 0) AS total_quantity, nvl(s.total_quantity, 0) * i.i_price AS total_sales</pre>	<pre>SELECT to_char(s.entry_date, 'YYYYMMDD')::char(8) AS date, i.i_name, coalesce(s.total_quantity::int, 0) AS total_quantity, coalesce(s.total_quantity::numeric, 0::numeric) * i.i_price::numeric AS total_sales</pre>

6. Convert the outer join by using (+) to the SQL standard description (Table 8-43).

Oracle Database supports (+) to perform an outer join. Fujitsu Enterprise Postgres uses **LEFT JOIN** to achieve the same result.

Table 8-43 Converting an outer join that is specified as (+)

Oracle Database	FUJITSU Enterprise Postgres
<pre>FROM item i, daily_sales s WHERE i.i_id = s.i_id(+)</pre>	<pre>FROM item i LEFT JOIN daily_sales s ON i.i_id = s.i_id</pre>

7. Convert the function (Table 8-44).

Convert functions as shown in step 5 on page 273.

Table 8-44 Example of converting functions

Oracle Database	FUJITSU Enterprise Postgres
<pre>TRUNC(calc_day, 'DD');</pre>	<pre>date_trunc('day', calc_day);</pre>
<pre>NVL(s.total_quantity, 0)</pre>	<pre>coalesce(s.total_quantity::numeric, 0::numeric)</pre>

8. Change the syntax for returning records (Table 8-45).

Oracle Database and Fujitsu Enterprise Postgres return the record data differently. Change the processing returning record data to meet the Fujitsu Enterprise Postgres specifications.

Table 8-45 Change how to return the record data

Oracle Database	FUJITSU Enterprise Postgres
<pre>CREATE OR REPLACE FUNCTION daily_sales_data (...) RETURN table_daily_sales_rec PIPELINED IS ... BEGIN ... ?PIPE ROW(daily_sales_rec(row.day, row.i_name, row.total_quantity, row.total_sales)); ... END; /</pre>	<pre>CREATE OR REPLACE FUNCTION daily_sales_data (...) RETURNS SETOF table_daily_sales_rec AS \$\$ DECLARE ... BEGIN ... RETURN NEXT rec; ... END; \$\$ LANGUAGE plpgsql;</pre>

Following these steps facilitates the migration of the function from PL/SQL to PL/pgSQL, as shown in Example 8-24.

Example 8-24 Function definition in Fujitsu Enterprise Postgres PL/pgSQL

```
-- Define the types of records
CREATE TYPE table_daily_sales_rec AS
(
  day char(8),
  i_name varchar(24),
  quantity numeric,
  sales numeric
);
-- Define a function that retrieves data such as sales per item on the specified
date
CREATE OR REPLACE FUNCTION daily_sales_data (getday timestamp without time zone)
RETURNS SETOF table_daily_sales_rec
AS $$
DECLARE
  rec table_daily_sales_rec;
  -- Extract the sales results for each item on the date that is specified as
arguments
  curs CURSOR FOR SELECT to_char(s.entry_date, 'YYYYMMDD')::char(8) AS date,
i.i_name,
                                coalesce(s.total_quantity::numeric, 0::numeric) AS
total_quantity,
                                coalesce(s.total_quantity::numeric, 0::numeric) *
i.i_price::numeric AS total_sales
FROM item i LEFT JOIN daily_sales s
ON i.i_id = s.i_id
WHERE date_trunc('day', s.entry_date) = date_trunc('day',
getday);
BEGIN
  -- Return the extracted data row by row
```

```
FOR rec IN curs LOOP
    RETURN NEXT rec;
END LOOP;
END;
$$ LANGUAGE plpgsql;
```

8.4.4 Migration scenario for small and medium-scale systems

This section describes the migration and consolidation of small and medium-scale Oracle databases to Fujitsu Enterprise Postgres on RHOCP. Key considerations and ways to resolve challenges are explained. Specifically, we describe the conceptual model for consolidation on a containerized environment and the step-by-step instructions for deploying security features.

We start with the characteristics and key considerations for migration of small and medium-scale systems:

- ▶ Characteristics

We assume that small or medium scale line-of-business systems are in organizations and that core systems are installed in branches or stores, such as in supermarket chains. Therefore, many systems are scattered throughout the company. These systems are operated, managed, and secured at each location and often operated primarily on small servers. The data that is handled in these systems is closed within each line-of-business, branch, and store to ensure high confidentiality. The total cost of operating and managing these systems increases proportionately to the number of systems that are scattered throughout the company.

- ▶ Key considerations for migration

Consider reducing operational and management costs by consolidating them into a container environment. There are two options about where to deploy a container: on a private cloud and on a public cloud. Choose the appropriate option depending on how sensitive the data is. In either choice of cloud, using container technology enables organizations to reduce operational costs.

When considering the migration of small and medium-scale database systems, it is also a great opportunity to consider moving to open interfaces. The benefit of replacing legacy systems with open systems is that the organization will have access to many tools and supporting software that can be implemented easily with the open interfaces. The wide variety of options serve as building blocks of DX. In addition to the benefits of open systems, Fujitsu Enterprise Postgres for Kubernetes enables the use of automation for database system operations based on Kubernetes technology.

In addition, when consolidating into a container environment, the data is transferred to an environment where multiple databases share the hardware layer and the platform layers. Therefore, it is necessary to consider enhanced security compared to the security measures that are taken for the source database systems where limited, closed access to each individual server ensured high confidentiality.

Consolidating database systems into Red Hat OpenShift Container Platform

Small and medium-scale databases that are scattered throughout an organization can be migrated to a single container environment, which reduces the cost of managing operations for multiple database systems.

RHOCP, which is a container environment that is supported on IBM LinuxONE, provides computing resources and management capabilities to run many containers in a cluster, which means that multiple database systems can be run in a cluster. Therefore, migrating database systems to RHOCP results in the consolidation of multiple database systems on a single platform.

Consolidating systems that previously operated independently in separate locations might raise various concerns. For example, organizations might have concerns about the adverse effects on one system by another system running in the same environment. Changes to configuration, operations, and applications of existing databases to reside with another database might be another concern. Also, data leakage of one database to other database administrators is a typical concern.

These concerns are addressed by the isolation capabilities of RHOCP and high security features of FUJITSU Enterprise Postgres. When different database systems are launched on RHOCP in different containers, there is no need to consider conflicting port numbers, OS usernames and database usernames, and directory configuration. Also, the RHOCP Project feature (namespaces in Kubernetes) allows database systems to be isolated, which means that with the appropriate permissions set for the database administrators, a database administrator with permission to operate a particular project cannot interfere with the database systems of other projects.

RHOCP and Fujitsu Enterprise Postgres provide role-based access control (RBAC). When the authentication and permissions features of RHOCP and Fujitsu Enterprise Postgres are configured and operational, legitimate means of attack do not influence the contents of the database. An example of an attack is using SQL from client applications to access the database.

However, there are non-legitimate means of attack, for example, stealing the storage device or file images or accessing directly the network equipment to eavesdrop on communication data. In these cases, authentication and permission settings are not a valid countermeasure. Encryption is effective to protect the data from attacks against database systems by these non-legitimate means. In addition to RBAC, Fujitsu Enterprise Postgres provides file-level encryption and encryption of communication.

Conceptual model for migration to an RHOCP environment

The conceptual model of migration for the consolidation of database systems to an RHOCP environment is shown in Figure 8-13. Multiple small to medium-scale Oracle RAC systems that are operated and maintained at each site are migrated to Fujitsu Enterprise Postgres and consolidated on a single RHOCP cluster in the cloud.

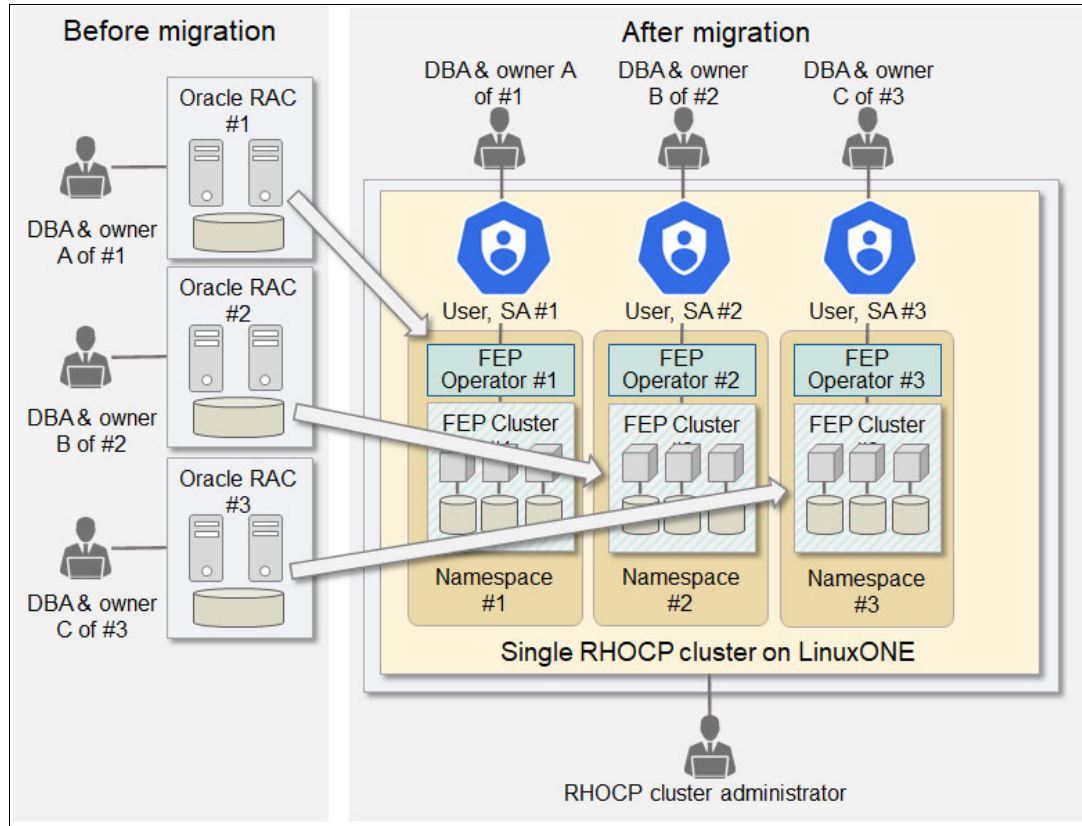


Figure 8-13 Migrating to an RHOCP environment model

The migration can run independently in each of the database systems because a namespace is allocated for each database cluster in the target architecture. Each of the small and medium-scale database systems that are scattered within an organization are migrated to one namespace on the RHOCP environment. Therefore, after migration to the container environment, organizations can continue to use the database names and database usernames that are currently used. By separating the database clusters (namespaces), the relationship between DBAs, database owners, and developers are maintained. The isolation between databases is also ensured.

In an RHOCP environment, configuration templates of Fujitsu Enterprise Postgres Operator are available for easy deployment and operations, which reduce cost. Only the necessary changes are applied to the templates.

In addition, the high capacity of IBM LinuxONE can be leveraged, so hardware resources can be used effectively.

Note: For more information about the procedures for deployment and operations of Fujitsu Enterprise Postgres by using Fujitsu Enterprise Postgres Operator, see Chapter 7, “Leveraging containers” on page 153.

Steps to migrate to an RHOCP environment

The process for migration is illustrated in Figure 8-14.

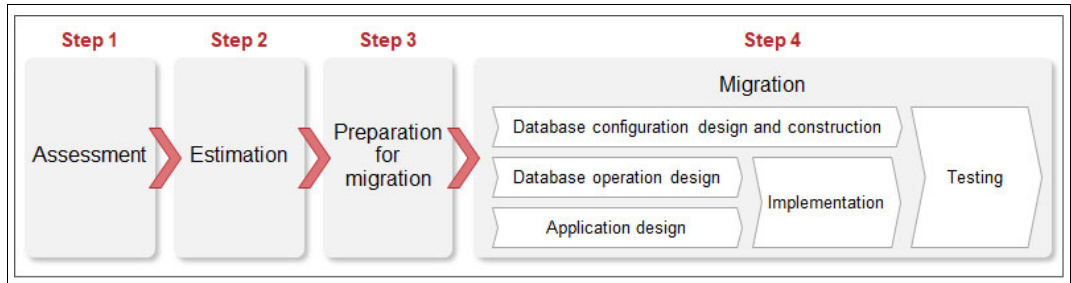


Figure 8-14 Migration process overview

This section describes the important points to remember when moving to a container environment. The work that is associated with the migration to an RHOCP environment by using Fujitsu Enterprise Postgres Operator is explained.

For more information about the migration flow, see 8.3.1, “Experience-based migration technical knowledge” on page 231. For more information about migration to FUJITSU Enterprise Postgres, see 8.4.3, “Migration scenario for large-scale legacy systems” on page 266.

This section assumes that the RHOCP environment on the cloud was created in advance. The following sections explain the migration process for databases.

Assessment

An assessment should be conducted while considering the migration.

This task is a common one when migrating to an on-premises environment. There are no special points to consider for the migration to an RHOCP environment.

Estimation

The cost and time that is required for the migration must be estimated to determine whether to perform the migration.

This task is a common one when migrating to an on-premises environment. There are no special points to consider for the migration to an RHOCP environment.

Preparing for migration

Based on the estimated work in the previous task, a migration schedule is created, and the migration team is gathered. The environments and assets that are required for the migration is prepared.

The resources (CPU, memory, and storage), projects (namespace), and administrator ID of the target RHOCP environment must be prepared.

Migration

In this task, the design and construction of the database configuration, operational design of the database, application design, implementation, and testing are performed.

Consider the following items for databases in the RHOCP environment for each activity:

- ▶ Migration activities: Database configuration design and construction:
 - Security design

Design security to ensure security in an environment with consolidated databases. There are two types of security design: permission design for accounts, and encryption of files and communication paths.

Design permissions and scope for DBAs and developers so that each person has no access to surrounding database instances. Apply permission settings to the accounts.

For more information about implementing Fujitsu Enterprise Postgres features for file-level encryption and communication-path encryption, see “Enabling security with the Fujitsu Enterprise Postgres Operator” on page 285.
 - Cluster configuration design

The cluster configuration comes in a template. The design of the cluster configuration, such as the number of replicas, can be designed based on the source system.

As a best practice, set up at least one replica for availability. Estimate the number of replicas against performance requirements.
- ▶ Migration activities: Database operation design:
 - Backup design

The backup runs automatically as configured. Design retention periods and schedules for backups (full backup and incremental backup) in accordance with current requirements.
 - Healing design

Automatic failover and automatic recovery are automated as configured, so there are no other considerations aside from configuring them.
 - Monitoring design

Configure monitoring with Grafana, Alert Manager, and Prometheus. Design the operations of monitoring to include the utilization of tools that are linked by open interfaces.

For more information, see 7.5.2, “Operation” on page 162 and 7.5.3, “Fluctuation” on page 190.
- ▶ Migration activities: Application design

Application design remains the same as for a migration to an on-premises environment. There are no special considerations for migrating to an RHOCP environment.
- ▶ Migration activities: Implementation

Implementation remains the same as a migration to an on-premises environment. There are no special considerations for migrating to an RHOCP environment.
- ▶ Migration activities: Testing

After migrating assets from the source system, perform operation verification to ensure that the target system is functioning correctly:

 - Consolidation on the RHOCP environment

Verify that the isolation and security settings between databases are sufficient.
 - Migrating to containers

Verify that the backup works as configured by setting up an automated backup.

Availability is automated in the RHOCF environment, so ensure that the applications work in a failover.

For more information about confirming the settings, see 7.5.2, “Operation” on page 162.

Enabling security with the Fujitsu Enterprise Postgres Operator

This section describes the procedures for deploying a database cluster by using Fujitsu Enterprise Postgres Operator and the following two security features:

► TDE

TDE is a feature that encrypts data at the file level and protects stored data. Encryption and decryption are transparent when reading and writing data files, so there is no need for the application to be aware of it. TDE is enabled by default when the database cluster is deployed.

► Mutual Transport Layer Security (MTLS)

The following three types of traffic can be secured by using a TLS connection that is protected by certificates:

- Postgres traffic from Client Application to FEPCluster
- Patroni RESTAPI within FEPCluster
- Postgres traffic within FEPCluster, such as replication and rewind

MTLS can be enabled by configuring it when deploying a database cluster.

Note: If these two security features must be used, perform both settings when the database cluster is deployed. It is not possible to enable security features after the database cluster is deployed. By enabling these features, it is possible to dynamically deploy encrypted table spaces and add clients by using MTLS.

Note: The setting to enable TDE and MTLS are explained in separate procedures. To enable both security features when deploying the database cluster, follow steps 1 on page 289 - 19 on page 295 in “Deploying communication path encryption by using MTLS” on page 289. Then, when setting the parameters for step 20 on page 299, include the parameter changes that are described in step 3 on page 286 in “Deploying with database encryption by using TDE” on page 286. Complete the final steps in “Deploying communication path encryption by using MTLS” on page 289.

Deploying with database encryption by using TDE

This section provides step-by-step instructions about how to build a database cluster with TDE enabled by using a template:

1. In the Red Hat OpenShift Console, click **Installed Operators**.
2. Select the Fujitsu Enterprise Postgres 13 Operator, as shown in Figure 8-15.

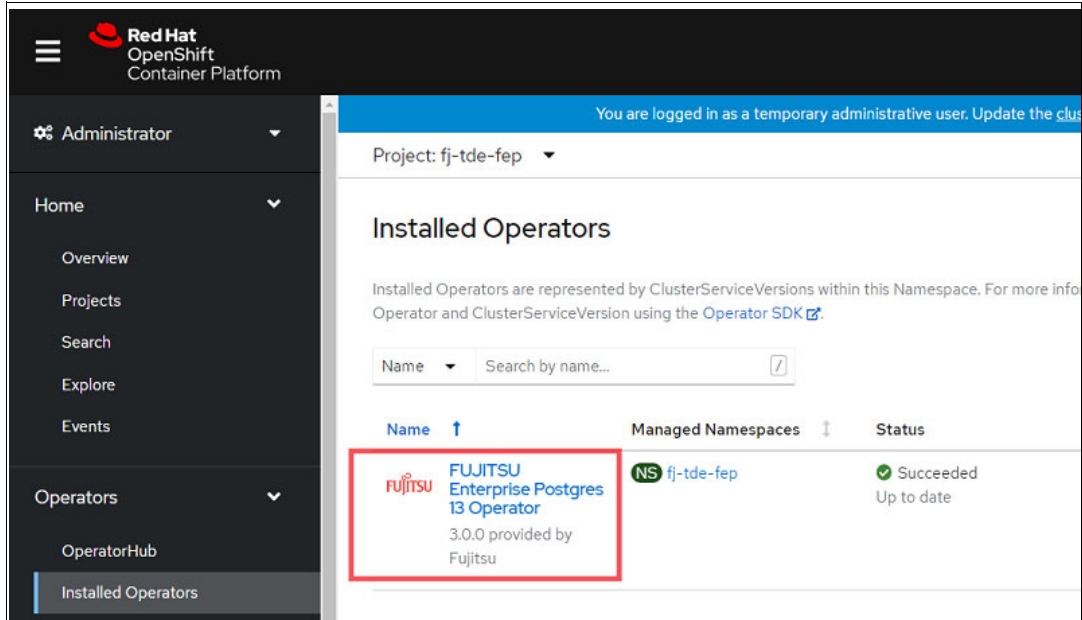


Figure 8-15 Installed Operators

3. In the Operator window, select **Create Instance**, as shown in Figure 8-16.

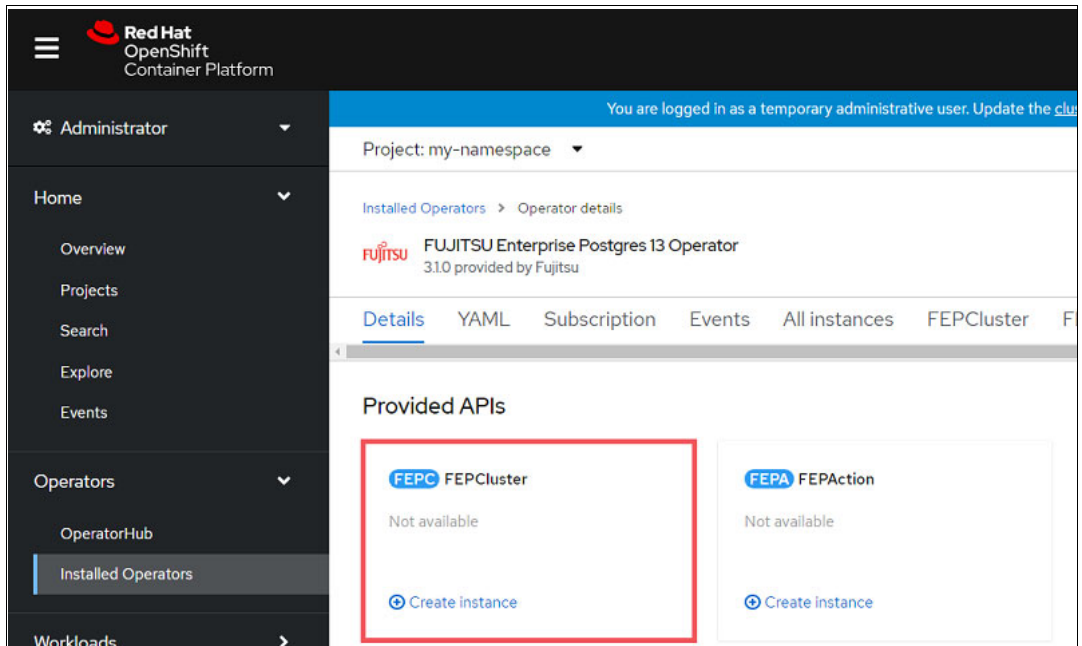


Figure 8-16 Creating a Fujitsu Enterprise Postgres cluster

In the Create FEPCluster window, click the **YAML** tab. Update the values as shown in Table 8-46 and CR configuration parameters as shown in Figure 8-17 on page 288. Update the deployment parameters and click **Create** to create a cluster.

Table 8-46 FEPCluster CR configuration file details

Field	Value	Details
metadata: name:	tde-fep	Name of the Fujitsu Enterprise Postgres cluster. Must be unique within a namespace.
spec: fep: mcSpec:	limits: cpu: 500 m memory: 700 Mi requests: cpu: 200 m memory: 512 Mi	Resource allocation to this container.
spec: fepChildCrVal: customPgHba:	host postgres postgres 10.131.0.213/32 trust	Entries to be inserted into pg_hba.conf.
spec: fepChildCrVal: sysUsers:	pgAdminPassword: admin-password	Password for postgres superuser.
	pgdb: mydb	Name of a user database that will be created.
	pguser: mydbuser	Name of user for the user database that will be created.
	pgpassword: mydbpassword	Password for pguser.
	pgrepluser: repluser	Name of a replication user. It is used for setting up replication between a primary and a replica in the Fujitsu Enterprise Postgres cluster.
	pgreplpassword: repluserpwd	Password for the user that is created for replication.
	tdepassphrase: mytdepassphrase	Passphrase for TDE. The default is tde-passphrase, so change the passphrase.

Field	Value	Details
spec: fepChildCrVal: storage:	dataVol: size: 2Gi storageClass: gold walVol: size: 1200 Mi storageClass: gold tablespaceVol: size: 512 Mi storageClass: gold archivalVol: size: 1Gi storageClass: gold logVol: size: 1Gi storageClass: gold backupVol: size: 2Gi storageClass: gold	Storage allocation to this container. For each volume, set the disk size that will be allocated and the name of the storageClass name that corresponds to the pre-provisioned storage. For more information about preparing disks, see 10.2.17 “Installing IBM Spectrum Virtualized and setting up storage class” in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE, SG24-8499</i> .

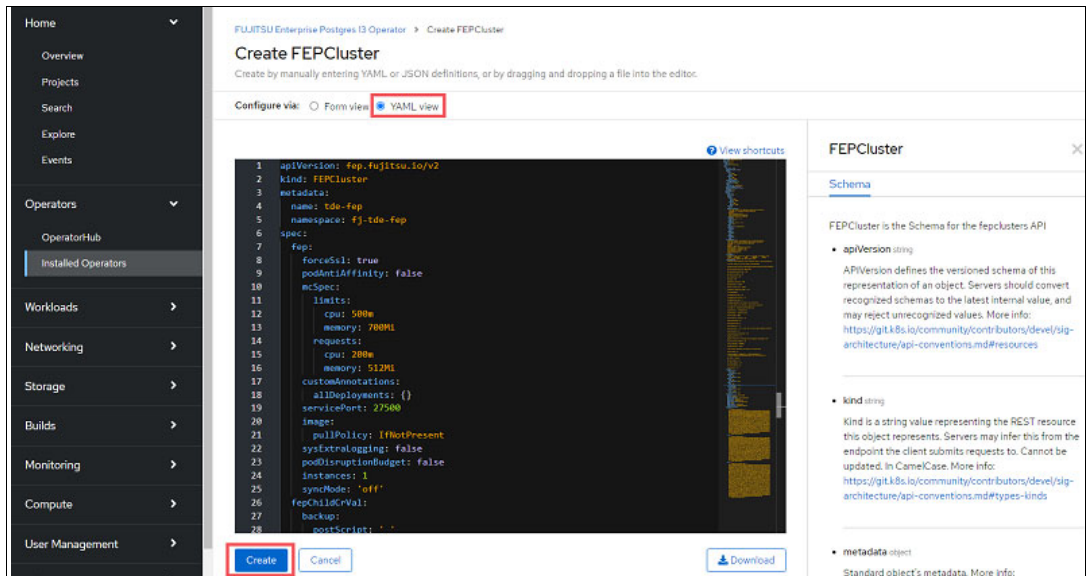


Figure 8-17 Updating the deployment parameters

Note: The TDE master encryption key can be updated by using `pgx_set_master_key`. For more information about updating the TDE master encryption key, see 4.4.10 “Managing the keystore” through “Rotating the TDE master key” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE, SG24-8499*.

- From the Fujitsu Enterprise Postgres client, connect to the postgres database and create a table space to apply TDE by using the command that is shown in Example 8-25.

Example 8-25 Creating a table space

```
$ psql -h tde-fep-primary-svc -p 27500 -U postgres -d postgres
postgres=# CREATE TABLESPACE secure_tablespace LOCATION
'/database/tablespaces/tbspace1' WITH (tablespace_encryption_algorithm =
'AES256');
```


- Verify that the created table space is the target of the encryption by using the command that is shown in Example 8-26.

Example 8-26 Sample output of pgx_tablespaces

```
postgres=# SELECT spcname, spcencalgo FROM pg_tablespace ts, pgx_tablespaces
          tsx WHERE ts.oid = tsx.spctablespace;
 spcname          | spcencalgo
-----+-----
 pg_default       | none
 pg_global        | none
 secure_tablespace | AES256
(3 rows)
```

- Create a TDE-protected database, as shown in Example 8-27.

Example 8-27 Creating a TDE-protected database

```
postgres=# CREATE DATABASE securedb TABLESPACE secure_tablespace;
```

- From the Fujitsu Enterprise Postgres client, connect to the securedb database and create secure_table. Then, insert the data 'Hello World' (Example 8-28).

Example 8-28 Creating a table and inserting data

```
$ psql -h tde-fep-primary-svc -p 27500 -U postgres -d securedb
securedb=# CREATE TABLE secure_table (msg text);
securedb=# INSERT INTO secure_table VALUES ('Hello World');
```

- Verify that the encrypted table can be referenced transparently (Example 8-29).

Example 8-29 Sample output of secure_table

```
securedb=# SELECT * FROM secure_table;
 msg
-----
 Hello World
(1 row)
```

Deploying communication path encryption by using MTLS

This section provides step-by-step instructions about how to deploy a database cluster with MTLS enabled. This task consists of three main tasks:

- Create certificates.

The administrator of the certificate authority (CA) certificates or server certificates creates the certificates that are required by MTLS.

The list of certificates, certificate file names, private key file names, and passphrases that are used in the deployment procedure is listed in Table 8-47.

Table 8-47 List of certificates that is used in the deployment procedure

Certificate	Certificate file name	Private key file name	Passphrase
CA Certificate	myca.pem	myca.key	0okm9ijn8uhb7ygv
Fujitsu Enterprise Postgres server certificate	fep.pem	fep.key	abcdefghijkl
Patroni certificate	patroni.pem	patroni.key	-
postgres user certificate	postgres.pem	postgres.key	-
repluser certificate	(Any value)	(Any value)	-
rewinduser certificate	(Any value)	(Any value)	-

2. Create a ConfigMap and Secrets.

The DBA creates the ConfigMap and Secret, which correspond to private keys and passphrases.

The list of ConfigMap and names of Secrets that are used in the deployment procedure is listed in Table 8-48.

Table 8-48 ConfigMap and Secrets that are used in the deployment procedure

Certificate	ConfigMap Name	Secret name for certificates and private keys	Secret name for the passphrase
CA certificate	cacert	-	-
Fujitsu Enterprise Postgres server certificate	-	mydb-fep-cert	mydb-fep-private-key-password
Patroni certificate	-	mydb-patroni-cert	-
postgres user certificates	-	mydb-postgres-cert	-
repluser certificate	-	mydb-repluser-cert	-
rewinduser certificate	-	mydb-rewinduser-cert	-

3. Create a database cluster.

The DBA deploys an MTLs-enabled database cluster that is based on the ConfigMap and Secrets created.

Here are the step-by-step instructions for the deployment procedure:

1. On the Red Hat OpenShift client machine, create a self-signed certificate as a CA. In this example, we used the command that is shown in Example 8-30. The output of that command is shown in Example 8-31.

Example 8-30 Sample output of openssl

```
$ openssl genrsa -aes256 -out myca.key 4096
Generating RSA private key, 4096-bit long modulus (2 primes)
.....++++
.....++++
e is 65537 (0x010001)
Enter pass phrase for myca.key: 0okm9ijn8uhb7ygv
Verifying - Enter pass phrase for myca.key: 0okm9ijn8uhb7ygv
```

Example 8-31 Sample output of openssl

```
$ cat << EOF > ca.cnf
[req]
distinguished_name=req_distinguished_name
x509_extensions=v3_ca
[v3_ca]
basicConstraints = critical, CA:true
keyUsage=critical,keyCertSign,digitalSignature,cRLSign
[req_distinguished_name]
commonName=Common Name
EOF
$ openssl req -x509 -new -nodes -key myca.key -days 3650 -out myca.pem -subj
"/O=My Organization/OU=CA /CN=My Organization Certificate Authority" -config
ca.cnf
Enter pass phrase for myca.key: 0okm9ijn8uhb7ygv
```

Note: For the migration procedure that is presented in this publication, the self-signed certificates are provided by a private CA. In a production environment, set up a CA that is trusted by the administrators and users.

2. Create a ConfigMap to store the CA certificate. We used the command that is shown in Example 8-32.

Example 8-32 Creating ConfigMap

```
$ oc create configmap cacert --from-file=ca.crt=myca.pem -n my-namespace
```

3. Create a password to protect the Fujitsu Enterprise Postgres server private key. We used the command that is shown in Example 8-33.

Example 8-33 Creating a password

```
$ oc create secret generic mydb-fep-private-key-password
--from-literal=keypassword=abcdefghijkl -n my-namespace
```

4. Create the Fujitsu Enterprise Postgres server private key (Example 8-34).

Example 8-34 Sample openssl command to create a server private key with output

```
$ openssl genrsa -aes256 -out fep.key 2048
Generating RSA private key, 2048-bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for fep.key: abcdefghijk
Verifying - Enter pass phrase for fep.key: abcdefghijk
```

5. Create a server certificate signing request. We used the command that is shown in Example 8-35.

Example 8-35 Creating a certificate signing request

```
$ cat << EOF > san.cnf
[SAN]
subjectAltName = @alt_names
[alt_names]
DNS.1 = *.my-namespace.pod
DNS.2 = *.my-namespace.pod.cluster.local
DNS.3 = mydb-primary-svc
DNS.4 = mydb-primary-svc.my-namespace
DNS.5 = mydb-primary-svc.my-namespace.svc
DNS.6 = mydb-primary-svc.my-namespace.svc.cluster.local
DNS.7 = mydb-replica-svc
DNS.8 = mydb-replica-svc.my-namespace
DNS.9 = mydb-replica-svc.my-namespace.svc
DNS.10 = mydb-replica-svc.my-namespace.svc.cluster.local
EOF
$ openssl req -new -key fep.key -out fep.csr -subj "/CN=mydb-headless-svc"
-reqexts SAN -config <(cat /etc/pki/tls/openssl.cnf <(cat san.cnf))
```

6. Create the server certificate that is signed by a CA (Example 8-36).

Example 8-36 Creating a signed server certificate

```
$ openssl x509 -req -in fep.csr -CA myca.pem -CAkey myca.key -out fep.pem -days
365 -extfile <(cat /etc/pki/tls/openssl.cnf <(cat san.cnf)) -extensions SAN
-CAcreateserial
Signature ok
subject=/CN=mydb-headless-svc
Getting CA Private Key
Enter pass phrase for myca.key: 0okm9ijn8uhb7ygv
```

7. Create the TLS secret to store the server certificate and key (Example 8-37).

Example 8-37 Creating the TLS secret

```
$ oc create secret generic mydb-fep-cert --from-file=tls.crt=fep.pem
--from-file=tls.key=fep.key -n my-namespace
```

8. Create a private key for Patroni (Example 8-38).

Example 8-38 Creating a private key for Patroni

```
$ openssl genrsa -out patroni.key 2048
Generating RSA private key, 2048-bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
```

Note: At the time of writing, the Fujitsu Enterprise Postgres container does not support a password protected private key for Patroni.

9. Create a certificate signing request for Patroni (Example 8-39).

Example 8-39 Creating a certificate signing request

```
$ cat << EOF > san.cnf
[SAN]
subjectAltName = @alt_names
[alt_names]
DNS.1 = *.my-namespace.pod
DNS.2 = *.my-namespace.pod.cluster.local
DNS.3 = mydb-primary-svc
DNS.4 = mydb-primary-svc.my-namespace
DNS.5 = mydb-replica-svc
DNS.6 = mydb-replica-svc.my-namespace
DNS.7 = mydb-headless-svc
DNS.8 = mydb-headless-svc.my-namespace
EOF
$ openssl req -new -key patroni.key -out patroni.csr -subj
"/CN=mydb-headless-svc" -reqexts SAN -config <(cat /etc/pki/tls/openssl.cnf
<(cat san.cnf))
```

10. Create a certificate signed by a CA for Patroni (Example 8-40).

Example 8-40 Creating a certificate that is signed by a CA

```
$ openssl x509 -req -in patroni.csr -CA myca.pem -CAkey myca.key -out
patroni.pem -days 365 -extfile <(cat /etc/pki/tls/openssl.cnf <(cat san.cnf))
-extensions SAN -CAcreateserial
Signature ok
subject=/CN=mydb-headless-svc
Getting CA Private Key
Enter pass phrase for myca.key: 0okm9ijn8uhb7ygv
```

11. Create a TLS secret to store the Patroni certificate and key (Example 8-41).

Example 8-41 Creating a TLS secret

```
$ oc create secret tls mydb-patroni-cert --cert=patroni.pem --key=patroni.key
-n my-namespace
```

12. Create a private key for a postgres user client certificate (Table 8-42).

Example 8-42 Creating a private key for the postgres user client certificate

```
$ openssl genrsa -out postgres.key 2048
Generating RSA private key, 2048-bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
```

Note: At the time of writing, the SQL client inside the Fujitsu Enterprise Postgres server container does not support password-protected certificate.

13. Create a certificate signing request for the postgres user client certificate (Example 8-43).

Example 8-43 Creating a certificate signing request

```
$ openssl req -new -key postgres.key -out postgres.csr -subj "/CN=postgres"
```

14. Create a client certificate for the postgres user (Example 8-44).

Example 8-44 Creating a client certificate

```
$ openssl x509 -req -in postgres.csr -CA myca.pem -CAkey myca.key -out
postgres.pem -days 365
```

15. Create a TLS secret to store the postgres user certificate and key (Example 8-45).

Example 8-45 Creating a TLS secret

```
$ oc create secret tls mydb-postgres-cert --cert=postgres.pem
--key=postgres.key -n my-namespace
```

16. Repeat steps 13 - 15 for repluser and rewinduser.

17. In the Red Hat OpenShift Console, click **Installed Operators**.

18. Select the Fujitsu Enterprise Postgres 13 Operator, as shown in Figure 8-18 on page 295.

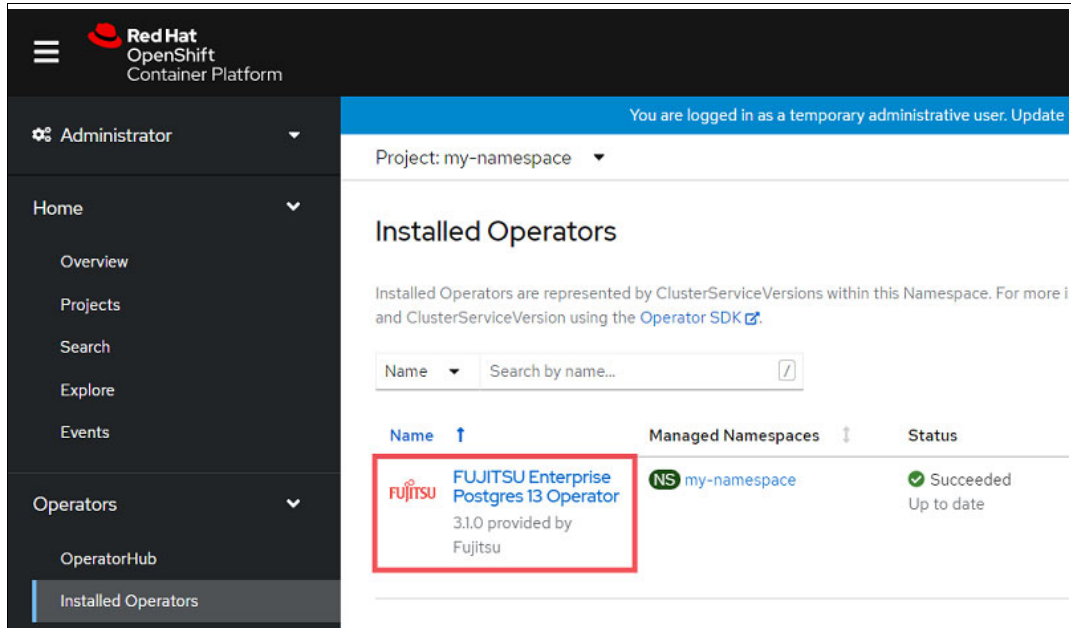


Figure 8-18 Installed Operators

19. In the Operator details window, select **Create Instance**, as shown in Figure 8-19.

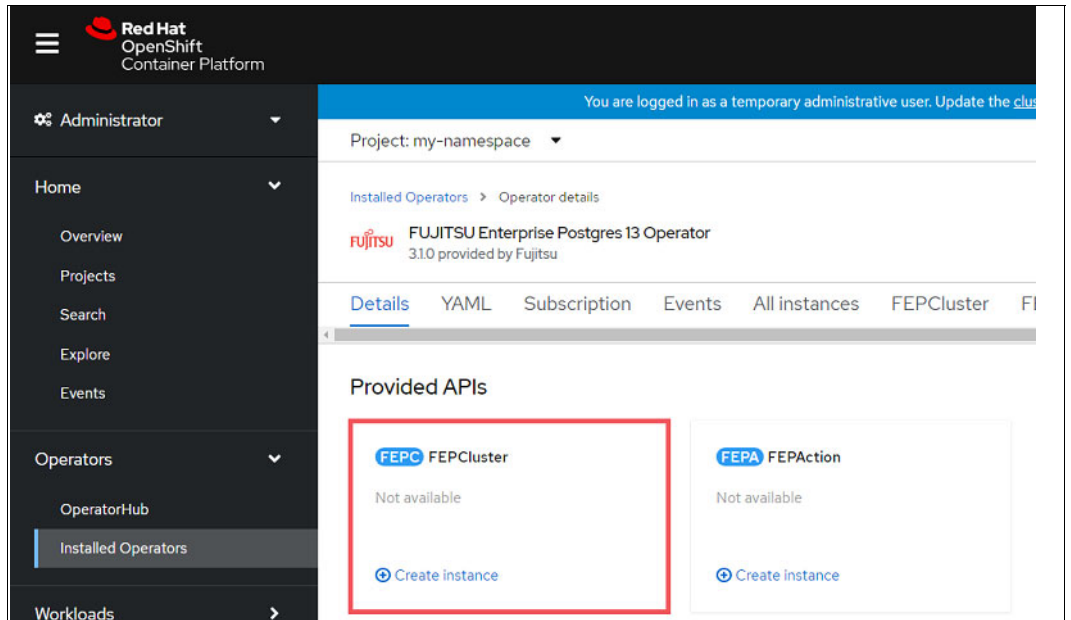


Figure 8-19 Creating a Fujitsu Enterprise Postgres cluster

In the Create FEPCluster window, click the **YAML** tab. Update the values as shown in Table 8-49. Update the deployment parameters and click **Create** to create a cluster, as shown in Figure 8-20 on page 299.

Table 8-49 FEPCluster CR configuration file details

Field	Value	Details
metadata: name:	mydb	Name of the Fujitsu Enterprise Postgres Cluster. Must be unique within a namespace.
metadata: namespace:	my-namespace	Name of the namespace.
spec: fep: usePodName:	true	Setting this key to true makes internal pod communication, both Patroni and Postgres, use a hostname instead of an IP address. This parameter is important for TLS because the hostname of the pod is predictable and can be used to create a server certificate, but an IP address is unpredictable and cannot be used to create certificates.
spec: fep: patroni: tls:	certificateName: mydb-patroni-cert	This parameter points to the Kubernetes TLS Secret that contains the certificate for Patroni. The certificate itself is stored in the key <code>tls.crt</code> .
	caName: cacert	This parameter points to the Kubernetes ConfigMap that contains an extra CA that Patroni uses to verify the client. The CA is stored in the key <code>ca.crt</code> .
spec: fep: postgres: tls:	certificateName: mydb-fep-cert	This parameter points to the Kubernetes TLS Secret that contains the certificate for Fujitsu Enterprise Postgres server. The certificate itself is stored in the key <code>tls.crt</code> .
	caName: cacert	This parameter points to the Kubernetes ConfigMap that contains an extra CA that the Fujitsu Enterprise Postgres server uses to verify the client. The CA is stored in the key <code>ca.crt</code> .
	privateKeyPassword: mydb-fep-private-key-passwo rd	This parameter points to the Kubernetes Secret that contains the password for the private key.
spec: fep: forceSsl:	true	This parameter ensures that the communication to the server is only through SSL. Changes are reflected in <code>pg_hba.conf</code> .
spec: fep: mcSpec:	limits: cpu: 500m memory: 700Mi requests: cpu: 200m memory: 512Mi	Resource allocation to this container.

Field	Value	Details
spec: fep: instances:	3	The number of Fujitsu Enterprise Postgres pods in the cluster.
spec: fepChildCrVal: customPgHba:	hostssl all all 0.0.0.0/0 cert hostssl replication all 0.0.0.0/0 cert	Entries to be inserted into pg_hba.conf.
spec: fepChildCrVal: sysUsers:	pgAdminPassword: admin-password	Password for postgres superuser.
	pgdb: mydb	Name of a user database to be created.
	pguser: mydbuser	Name of a user for the user database that will be created.
	pgpassword: mydbpassword	Password for pguser.
	pgrepluser: repluser	Name of a replication user. It is used to set up replication between the primary and replica in Fujitsu Enterprise Postgres Cluster.
	pgreplpassword: repluserpwd	Password for the user that is created for replication.
	tdepassphrase: tde-passphrase	Passphrase for TDE.
spec: fepChildCrVal: sysUsers: pgAdminTls:	certificateName: mydb-postgres-cert	This parameter points to the Kubernetes TLS Secret that contains the certificate of the Postgres user postgres. Patroni uses this certificate for certificate authentication. The certificate itself is stored in the key tls.crt.
	caName: cacert	This parameter points to the Kubernetes ConfigMap that contains an extra CA that the client uses to verify a server certificate. The CA is stored in the key ca.crt.
	sslMode: verify-full	Specifies the type of TLS negotiation with the server.

Field	Value	Details
spec: fepChildCrVal: sysUsers: pgrepluserTls:	certificateName: mydb-repluser-cert	This parameter points to the Kubernetes TLS Secret that contains the certificate of the Postgres user repluser. Patroni uses this certificate for certificate authentication. The certificate itself is stored in the key tls.crt.
	caName: cacert	This parameter points to the Kubernetes ConfigMap that contains an extra CA that the client uses to verify a server certificate. The CA is stored in the key ca.crt.
	sslMode: verify-full	Specifies the type of TLS negotiation with the server.
spec: fepChildCrVal: sysUsers: pgRewindUserTls:	certificateName: mydb-rewinduser-cert	This parameter points to the Kubernetes TLS Secret that contains the certificate of Postgres user rewinduser. Patroni uses this certificate for certificate authentication. The certificate itself is stored in the key tls.crt.
	caName: cacert	This parameter points to the Kubernetes ConfigMap that contains an extra CA that the client uses to verify a server certificate. The CA is stored in the key ca.crt.
	sslMode: verify-full	Specifies the type of TLS negotiation with the server.
spec: fepChildCrVal: storage:	dataVol: size: 2Gi storageClass: gold walVol: size: 1200Mi storageClass: gold tablespaceVol: size: 512Mi storageClass: gold archivewalVol: size: 1Gi storageClass: gold logVol: size: 1Gi storageClass: gold backupVol: size: 2Gi storageClass: gold	Storage allocation to this container. For each volume, set the disk size that will be allocated and the name of the storageClass name that corresponds to the pre-provisioned storage. For more information about preparing disks, see 10.2.17 “Installing IBM Spectrum Virtualized and setting up storage class” in <i>Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE, SG24-8499</i> .

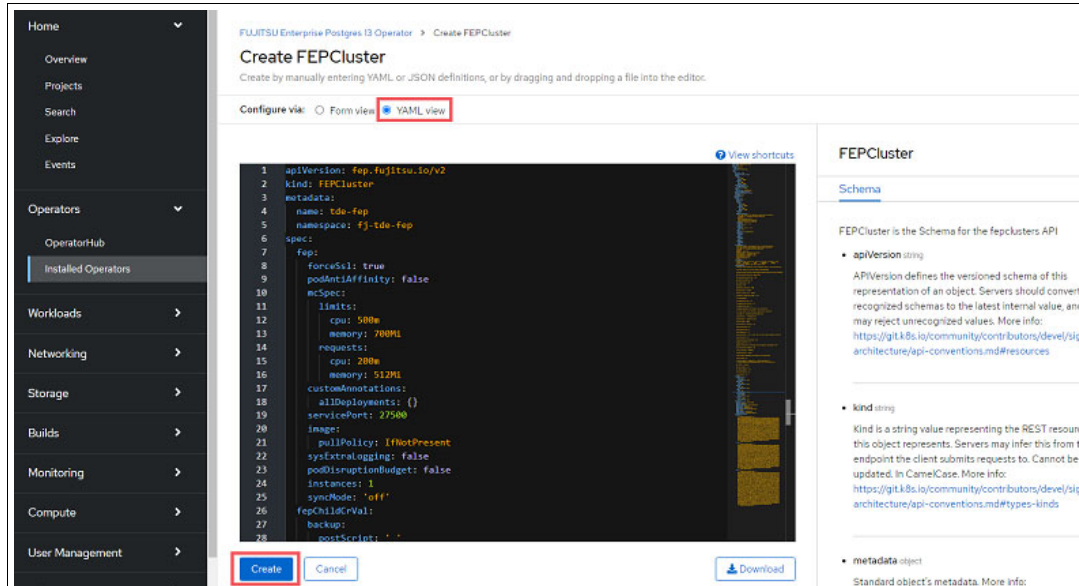


Figure 8-20 Updating deployment parameters

20. The database cluster is deployed, and the deployment status can be checked by selecting **Workloads** → **Pods**, as shown in Figure 8-21. The status shows *Running* after the cluster is ready.

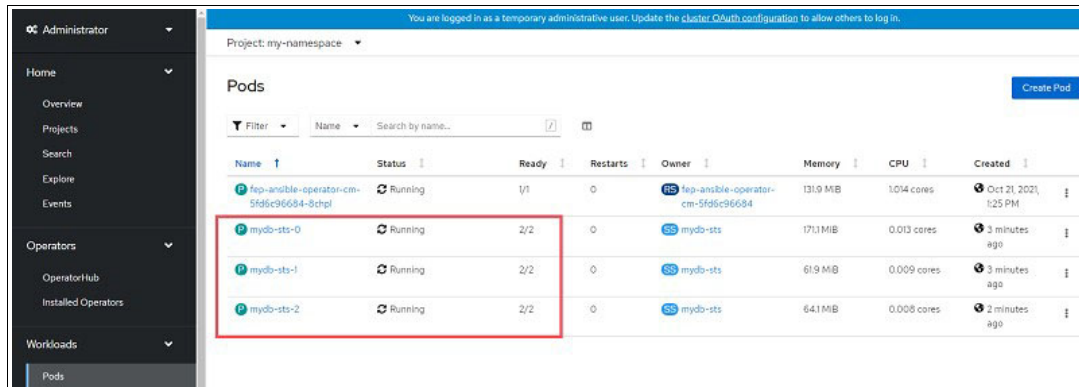


Figure 8-21 Database cluster deployment result

All Fujitsu Enterprise Postgres pods must show the status as *Running*.

You successfully installed Fujitsu Enterprise Postgres on Red Hat OpenShift Cluster on an IBM LinuxONE server platform with MTLS-enabled communication path.

Note: For more information about the parameters of the Fujitsu Enterprise Postgres cluster FEPCluster CR configuration, see 1.1, “FEPCluster Parameter” in [Fujitsu Enterprise Postgres 13 for Kubernetes Reference Guide](#).

MTLS connection by the Fujitsu Enterprise Postgres client

This section provides step-by-step instructions about how to connect to an MTLS-enabled database cluster by using the Fujitsu Enterprise Postgres client:

1. The administrator of the server certificate distributes the server root certificates (`myca.pem`) that are created when deploying the MTLS-enabled database cluster to the clients (application developers).
2. The client administrator creates a private key for a client certificate and requests a client certificate from a CA. Then, the client certificate and the private key must be distributed to the clients (application developers).

Note: If the server and client root certificates are different, the DBA must update the `spec.fep.postgres.tls.caName` parameter. For more information, see the [Fujitsu Enterprise Postgres 13 for Kubernetes Reference Guide](#).

3. The clients (application developers) use the server root certificate (`myca.pem`), client certificate (`tls.crt`), and private key (`tls.key`) to connect to the database cluster, as shown in Example 8-46.

Example 8-46 Connecting to the database cluster

```
$ psql 'host= mydb-primary-svc.my-namespace port=27500 user=aplluser
sslcert=/client/tls.crt sslkey=/client/tls.key sslrootcert=/client/myca.pem
sslmode=verify-full'
```



Application use case-geospatial data

This chapter provides a use case that uses PostgreSQL for the storage, mapping, and representation of geospatial data.

Additional LinuxONE use cases can be found in Appendix A of the IBM Redbooks publication, [Practical Migration from x86 to LinuxONE](#), SG24-8377.

For more information about other use cases, see the [IBM LinuxONE client web page](#).

8.1 Geospatial data

This section describes the importance of geospatial data and how it is being increasingly used to evaluate and predict trends in socioeconomic data. Geospatial information systems (GISs) relate specifically to the physical mapping of data within a visual representation. Figure 8-1 shows typical use-cases involving financial, healthcare, retail, energy, commercial and environmental studies.

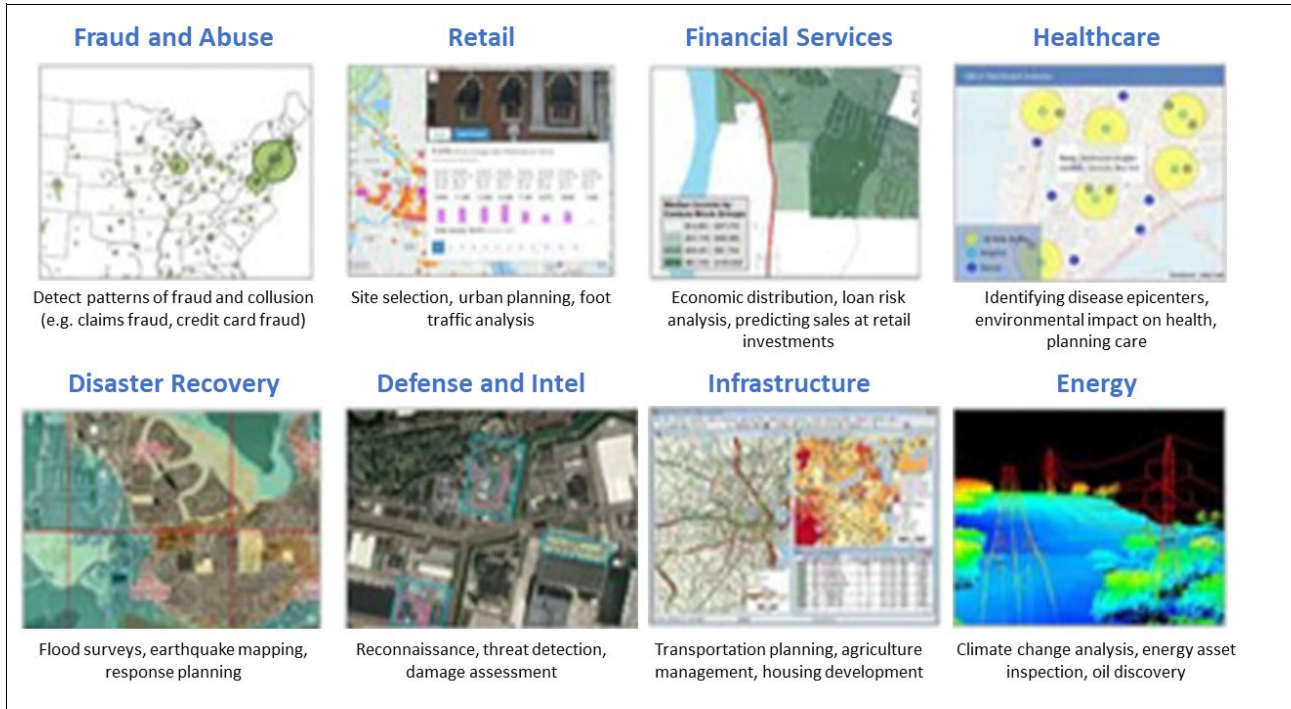


Figure 8-1 Geospatial data source examples

8.1.1 Introducing geospatial information systems and geospatial data

One of the growing trends of using PostgreSQL is the storage, mapping, and representation of geospatial data. From shipping, finance, supply chain, and delivery requirements to urban, environmental, and ecological studies, the precise location and imagery that are associated with a building, street, or object such as a tree, lake, river, or even a pond is becoming more important.

Location data is no longer only an address or GPS coordinate within a 2D plane. In a geospatial or 3D world, this data is represented by points, vectors, and depth, area; continuous data such as temperature and elevation; or spectral data such as satellite images, aerial photographs, and digital pictures. Non-geographical data is also captured within a geospatial reference that is known as *raster* data. Raster data represents real-world phenomena: socioeconomic data, such as financial tables, population densities, health records, weather, and traffic data; and places of interest and collections, such as the number of trees, plants, and animal species that are found in a specific area.

Location data is captured in a standard geometric format that is usually based on the Geographic Coordinate System (GCS), which uses latitude and longitude. These calculations are based on the 360 degrees of the Earth with the equator representing the positive and negative division of the latitude degrees and the Prime Meridian (Greenwich Observatory, London) representing the start and end of the longitude degrees. This measurement system

evolved to a standard known as ISO 6709, but there are many variants and influences that are based on the requirements to measure and plot above or below ground or beyond Earth, that is, outer space.

Having deployed PostGIS within the Fujitsu PostgreSQL Enterprise Server, these coordinates are converted and stored as either geometry or geography data types and then associated with spatial reference systems (SRSs). Unlike other database management systems (DBMSs), PostGIS supports multiple SRS IDs instead of the usual EPSG:4326, which is a worldwide system that is used by GPS systems. These values consist of components that describe a series of 3D geographic parameters, such as the orientation, latitude, longitude, and elevation in reference to geographic objects, which define coordinate systems and spatial properties on a map.

Figure 8-2 on page 303 provides an example of the Tower of London, which is a Central London Place of Interest.

- ▶ Latitude: 51.508530 DMS Lat: 51° 30' 30.7080" N
- ▶ Longitude: -0.07702 DMS Long: 0° 4' 34.0752" W

These coordinates are represented as a GIS or geometry value of "0020000001000010E94049 C11782D38477BFFD05FAEBC408D9" that is based on an spatial reference identifier (SRID) of 4326.



Figure 8-2 Geospatial coordinates

Additionally, *geocoding* is the process of transforming a description of a location, such as a pair of coordinates, an address, or a name of a place, to a location on the earth's surface. You can geocode by entering one location description at a time or by providing many of them at once in a table. *What3words* is an example of a geocoding service that transforms locations into three specific words, for example, `///gallons.pinch.sketch` provides the W3W reference for the Tower of London.

So, in summary, data from geocoding systems, whether GPS devices, postal code systems, or mapping services such as What3words, and imagery data are stored, converted, and rendered as visual models in graphical mapping solutions. Openstreetmap.org is a common open-source application that graphically maps the location by converting geocoding data into GCS coordinates and providing overlays such as satellite imagery that are embedded into many commercial applications. These coordinates vary in accuracy based on the precision of the data that includes GPS coordinates, where each degree represents an area of 111 km^2 . You can use up to eight decimal places, which represent an accuracy to within 1.11 mm^2 . Typically, postal addresses contain four decimal places, and Google Maps uses seven decimal places, which represent an accuracy of 11.1 meters^2 .

For more information about this topic, see [Precision](#) and [Address geocoding](#).

8.1.2 Using FUJITSU Enterprise Postgres server for geospatial data

There are many geospatial services that are deployed as extensions to database platforms that store location-based data and offer a range of tools to query and manage the data. PostGIS is an Open Spatial Consortium (OSC)-compliant extender for FUJITSU Enterprise Postgres that offers the widest range of geospatial functions, such as distance, area, union, and intersection, and specialty geometry and raster data. Additionally, pgRouting is an extension that adds routing between and around locations and other network analysis functions to PostGIS and Fujitsu PostgreSQL databases to provide shortest path search and other graph analysis functions. Fujitsu offers other extensions and supports multiple SRS IDs, so a single database can store worldwide addresses, geometries, and their associated SRS IDs to map them, which are key when running low-level granularity on imagery down to the 1.11 m for eight-decimal place location analysis.

The common debate among data scientists is whether to run models, predictions, or R/Python programs locally by using client tools such as Quantum Geographic Information System (QGIS) or OpenJump against files, or use databases and specifically PostGIS functions. Raster data is the complex data that provides the second or third dimension of geospatial queries, often overlaying socioeconomic data, such as population densities and weather patterns, on top of geographical maps. As models are developed, scientists use their PCs and dedicated x86 server platforms to provide compute- and memory-intensive processing, but they can operate only at small volumes and often run out of space or memory. Therefore, the scientists must break the models into much smaller executable units and then stitch the results back together. Using public clouds solves some of the compute and memory challenges, but unpredictable processing models often result in unforeseen billing costs, which make this experiment expensive.

As you can imagine, this geospatial processing involves large volumes of data and intensive data processing that combines large mapping data sources with socioeconomic data. You need a high-performing database that can process the complex analytical queries while inferring the properties of several data types. FUJITSU Enterprise Postgres on IBM LinuxONE combines those key properties to ensure a robust, secure, and high-performing geospatial platform.

8.1.3 Key PostGIS functions

FUJITSU Enterprise Postgres and PostGIS provide a rich library of data types, casts, extensions, and functions to explore geospatial data (PostGIS V3.2 provides over 400 dedicated functions). All these functions are accessible by using SQL and Open Database Connectivity (ODBC) coding, but some might be explicit and therefore require casting, and

others might involve complex pre-built functions that are written in C, PL/SQL, or other languages.

In this section, we describe some of the most commonly used features and functions with comments about the performance requirements where available. For more information about these features and functions, see the [PostGIS Reference guide](#).

Base data types

Here are the key data types for storing geospatial data:

- ▶ [box2d](#): Represents a 2-dimensional bounding box.
- ▶ [box3d](#): Represents a 3-dimensional bounding box.
- ▶ [geometry](#): Represents spatial features with planar coordinate systems.
- ▶ [geometry_dump](#): A composite type that is used to describe the parts of complex geometry.
- ▶ [geography](#): Represents spatial features with geodetic (ellipsoidal) coordinate systems.

The casts that are shown in Figure 8-3 transform data types from one format to another one.

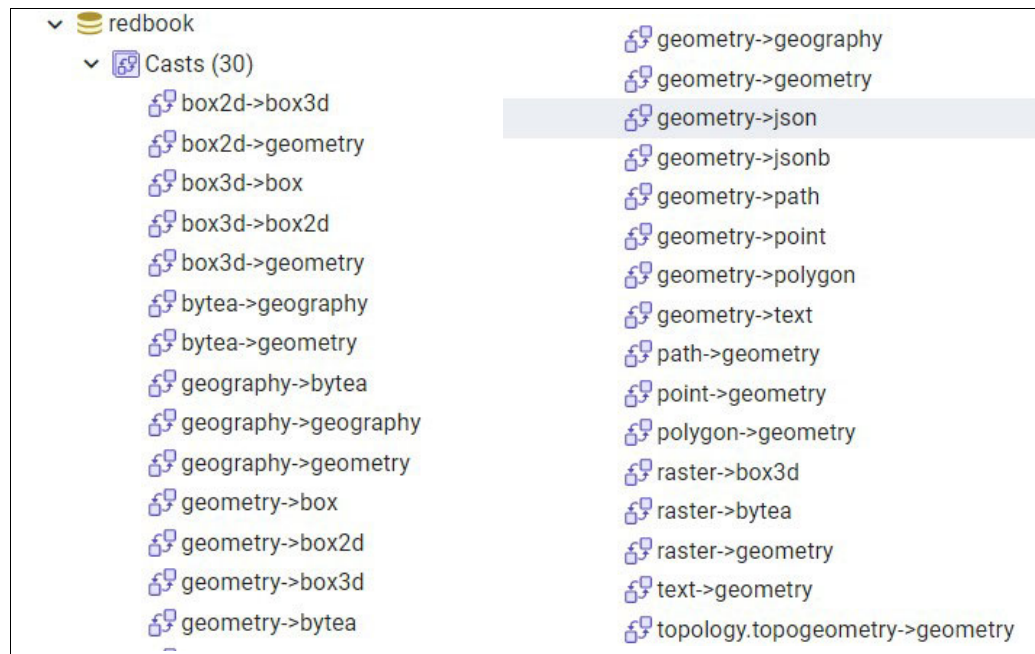


Figure 8-3 Casts showing data type transformations

Here are the associated data type functions, which are also known as geometry accessors and constructors and editors.

- ▶ [AddGeometryColumn](#): Adds a geometry column to an existing table.
- ▶ [DropGeometryColumn](#): Removes a geometry column from a spatial table.
- ▶ [DropGeometryTable](#): Drops a table and all its references into `geometry_columns`.
- ▶ [Find_SRID](#): Returns the SRID that is defined for a geometry column.
- ▶ [Populate_Geometry_Columns](#): Ensures that geometry columns are defined with type modifiers or have the appropriate spatial constraints.
- ▶ [UpdateGeometrySRID](#): Updates the SRID of all features in a geometry column, and the table metadata.

Constructors

Here are some of the constructor functions that are used to create geometries:

- ▶ **ST_Point**: Creates a point with the provided coordinate values. Alias for ST_MakePoint.
- ▶ **ST_PointZ**: Creates a point with the provided coordinate and SRID values.
- ▶ **ST_PointM**: Creates a point with the provided coordinate and SRID values.
- ▶ **ST_PointZM**: Creates a point with the provided coordinate and SRID values.
- ▶ **ST_Polygon**: Creates a Polygon from a linestring with a specified SRID.
- ▶ **ST_TileEnvelope**: Creates a rectangular polygon in Web Mercator (SRID:3857) by using the XYZ tile system.
- ▶ **ST_HexagonGrid**: Returns a set of hexagons and cell indexes that cover the bounds of the geometry argument.
- ▶ **ST_Hexagon**: Returns a single hexagon that uses the provided edge size and cell coordinate within the hexagon grid space.
- ▶ **ST_SquareGrid**: Returns a set of grid squares and cell indexes that cover the bounds of the geometry argument.
- ▶ **ST_Square**: Returns a single square that uses the provided edge size and cell coordinate within the square grid space.

Accessors

Here are some of the accessor functions:

- ▶ **ST_Area**: Returns the area of the surface if it is a polygon or multi-polygon. For a “geometry” type, the area is in SRID units. For a “geography” type, the area is in square meters.
- ▶ **ST_Boundary**: Returns the boundary of a geometry.
- ▶ **ST_Distance**: For a geometry type, returns the 2-dimensional Cartesian minimum distance (based on the spatial reference) between two geometries in projected units. For a geography type, defaults to a return spheroidal minimum distance between two geographies in meters.
- ▶ **ST_Intersection**: (T) Returns a geometry that represents the shared portion of geomA and geomB. The geography implementation does a transform to geometry to do the intersection and then transform back to WGS84.
- ▶ **ST_Intersects**: Returns TRUE if the geometries or geography spatially intersect in 2D (share any portion of space) and returns FALSE if they do not (they are disjointed). For geography, the tolerance is 0.00001 meters, so any points that close are considered to intersect.
- ▶ **ST_Length**: Returns the 2D length of the geometry if it is a linestring or multilinestring. Geometry is in units of spatial reference, and geography is in meters (default spheroid).
- ▶ **ST_Perimeter**: Returns the length measurement of the boundary of an ST_Surface or ST_MultiSurface for geometry or geography (polygon or multipolygon). The geometry measurement is in units of spatial reference, and geography is in meters.

Spatial relationships

Spatial functions model data into objects that can be represented graphically. When combined with geometry or geography data types, they model information onto maps, such as population densities or number of fast-food restaurants in an area.

Topological relationships

The PostGIS topology types and functions are used to manage topological objects such as faces, edges, and nodes. Among these types and functions are the following ones:

- ▶ **ST_3DIntersects**: Returns true if two geometries spatially intersect in 3D. Only for points, linestrings, polygons, and polyhedral surfaces (area).
- ▶ **ST_Contains**: Returns true if no points of B lie in the exterior of A, and A and B have at least one interior point in common.
- ▶ **ST_ContainsProperly**: Returns true if B intersects the interior of A but not the boundary or exterior.
- ▶ **ST_CoveredBy**: Returns true if no point in A is outside B.
- ▶ **ST_Covers**: Returns true if no point in B is outside A.
- ▶ **ST_Crosses**: Returns true if two geometries have some, but not all, interior points in common.
- ▶ **ST_LineCrossingDirection**: Returns a number indicating the crossing behavior of two linestrings.
- ▶ **ST_Disjoint**: Returns true if two geometries do not intersect (they have no point in common).
- ▶ **ST_Equals**: Returns true if two geometries include the same set of points.
- ▶ **ST_Intersects**: Returns true if two geometries intersect (they have at least one point in common).
- ▶ **ST_OrderingEquals**: Returns true if two geometries represent the same geometry and have points in the same directional order.
- ▶ **ST_Overlaps**: Returns true if two geometries intersect and have the same dimension but are not contained by each other.
- ▶ **ST_Relate**: Tests whether two geometries have a topological relationship matching an Intersection Matrix pattern or computes their Intersection Matrix.
- ▶ **ST_RelateMatch**: Tests whether a DE-9IM Intersection Matrix matches an Intersection Matrix pattern.
- ▶ **ST_Touches**: Returns true if two geometries have at least one point in common, but their interiors do not intersect.
- ▶ **ST_Within**: Returns true if no points of A lie in the exterior of B, and A and B have at least one interior point in common.

Distance relationships

The PostGIS distance types and functions provide spatial distance relationships between geometries:

- ▶ **ST_3DDWithin**: Returns true if two 3D geometries are within a certain 3D distance.
- ▶ **ST_3DDFullyWithin**: Returns true if two 3D geometries are entirely within a certain 3D distance.
- ▶ **ST_DFullyWithin**: Returns true if two geometries are entirely within a certain distance.
- ▶ **ST_PointInsideCircle**: Tests whether a point geometry is inside a circle that is defined by a center and radius.
- ▶ **ST_DWithin**: Returns true if the geometries are within a given distance.

Example 8-1 shows an example of using the function `ST_DWithin`, and Figure 8-4 provides the results of the example. This example provides an answer to the question, “How many fast-food restaurants within 1 mile of a US highway?”¹

Example 8-1 ST_DWithin example

```

SELECT f.franchise
, COUNT(DISTINCT r.id) As total -- <1>
FROM ch01.restaurants As r
  INNER JOIN ch01.lu_franchises As f ON r.franchise = f.id
    INNER JOIN ch01.highways As h
      ON ST_DWithin(r.geom, h.geom, 1609) -- <2>
GROUP BY f.franchise
ORDER BY total DESC;

```

	franchise	total
	character varying (30)	bigint
1	McDonald	5343
2	Burger King	3049
3	Pizza Hut	2920
4	Wendys	2446
5	Taco Bell	2428
6	Kentucky Fried Chicken	2371
7	Hardee	1077
8	Jack in the Box	509
9	Carl's Jr	224
10	In-N-Out	44

Figure 8-4 PostGIS function example of ST_DWithin

Measurement functions

The following functions compute measurements of distance, area, and angles. There are also functions to compute geometry values that are determined by measurements.

- ▶ **ST_Area**: Returns the area of polygonal geometry.
- ▶ **ST_BuildArea**: Creates a polygonal geometry that is formed by the linework of a geometry.
- ▶ **ST_Azimuth**: Returns the north-based azimuth as the angle in radians as measured clockwise from the vertical on pointA to pointB.
- ▶ **ST_Angle**: Returns the angle between three points, or between two vectors (four points or two lines).
- ▶ **ST_ClosestPoint**: Returns the 2D point on g1 that is closest to g2. That point is the first point of the shortest line.
- ▶ **ST_3DClosestPoint**: Returns the 3D point on g1 that is closest to g2. That point is the first point of the 3D shortest line.
- ▶ **ST_Distance**: Returns the distance between two geometry or geography values.

¹ Sources: <http://www.fastfoodmaps.com>; US highways maps, found at <https://gisgeography.com/us-road-map/>; *PostGIS In Action*, found at <https://www.manning.com/books/postgis-in-action-third-edition>.

- ▶ **ST_3DDistance**: Returns the 3D Cartesian minimum distance (based on the spatial reference) between two geometries in projected units.
- ▶ **ST_DistanceSphere**: Returns the minimum distance in meters between two longitude and latitude geometries by using a spherical earth model.
- ▶ **ST_DistanceSpheroid**: Returns the minimum distance between two longitude and latitude geometries by using a spheroidal earth model.
- ▶ **ST_FrechetDistance**: Returns the Fréchet distance between two geometries.
- ▶ **ST_HausdorffDistance**: Returns the Hausdorff distance between two geometries.
- ▶ **ST_Length**: Returns the 2D length of a linear geometry.
- ▶ **ST_Length2D**: Returns the 2D length of a linear geometry. It is an alias for ST_Length.
- ▶ **ST_3DLength**: Returns the 3D length of a linear geometry.
- ▶ **ST_LengthSpheroid**: Returns the 2D or 3D length or perimeter of a longitude and latitude geometry on a spheroid.
- ▶ **ST_LongestLine**: Returns the 2D longest line between two geometries.
- ▶ **ST_3DLongestLine**: Returns the 3D longest line between two geometries.
- ▶ **ST_MaxDistance**: Returns the 2D largest distance between two geometries in projected units.
- ▶ **ST_3DMaxDistance**: Returns the 3D Cartesian maximum distance (based on a spatial reference) between two geometries in projected units.
- ▶ **ST_MinimumClearance**: Returns the minimum clearance of a geometry, which is a measure of a geometry's robustness.
- ▶ **ST_MinimumClearanceLine**: Returns the two-point linestring spanning a geometry's minimum clearance.
- ▶ **ST_Perimeter**: Returns the length of the boundary of a polygonal geometry or geography.
- ▶ **ST_Perimeter2D**: Returns the 2D perimeter of a polygonal geometry. It is an alias for ST_Perimeter.
- ▶ **ST_3DPerimeter**: Returns the 3D perimeter of a polygonal geometry.
- ▶ **ST_Project**: Returns a point that is projected from a start point by distance and bearing (azimuth).
- ▶ **ST_ShortestLine**: Returns the 2D shortest line between two geometries.
- ▶ **ST_3DShortestLine**: Returns the 3D shortest line between two geometries.

As an example, you want to know the population density of an area outside of the town center, which is a ring road in effect, where you are interested in locating your retail business. The function that is shown in Example 8-2 creates an areal geometry that is formed by the constituent linework of a geometry. The return type can be a polygon or multi-polygon, depending on the input.

Example 8-2 ST_BuildArea example

```
SELECT ST_BuildArea(ST_Collect(smallc,bigc))
FROM (SELECT
ST_Buffer( ST_GeomFromText('POINT(100 90)'), 25) As smallc,
ST_Buffer( ST_GeomFromText('POINT(100 90)'), 50) As bigc) As foo;
```

The resultant data set is shown in Example 8-3.

Example 8-3 Results of STBuildArea

```
Result dataset (2,146 chars)
"000000000300000002000000214062C0000.....
DB010CD765405F40000000000405680000000000"
```

The resultant graph is a “donut”, as shown in Figure 8-5.

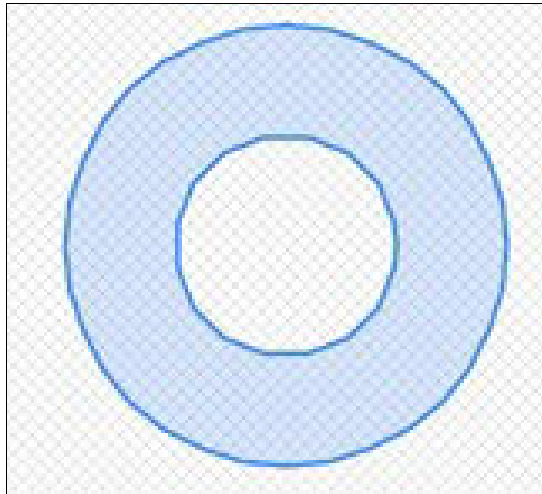


Figure 8-5 PostGIS function *ST_BuildArea*

Now, consider the effect of applying that query to a population density raster data set and identifying the number of inhabitants (and potential clients) that are based within an area. That area might vary in a sparsely populated terrain versus a large city. The data results and data processing requirements would vary considerably.

Raster functions

Raster data provides *a representation of the world as a surface divided up into a regular grid array of cells*², where each of these cells has an associated value. When transferred into a GIS setting, the cells in a raster grid can potentially represent other data values, such as temperature, rainfall, or elevation. Each raster has one or more tiles, each having a set of pixel values that are grouped into chunks. Rasters can be georeferenced. There are a number of constructors, editors, accessors, and management functions that are related to raster data sets, but in this section we describe the processing functions that transpose socioeconomic data onto geospatial maps.

Figure 8-6 on page 311 provides an example of vector and raster grid models.

² <https://spatialvision.com.au/blog-raster-and-vector-data-in-gis/>

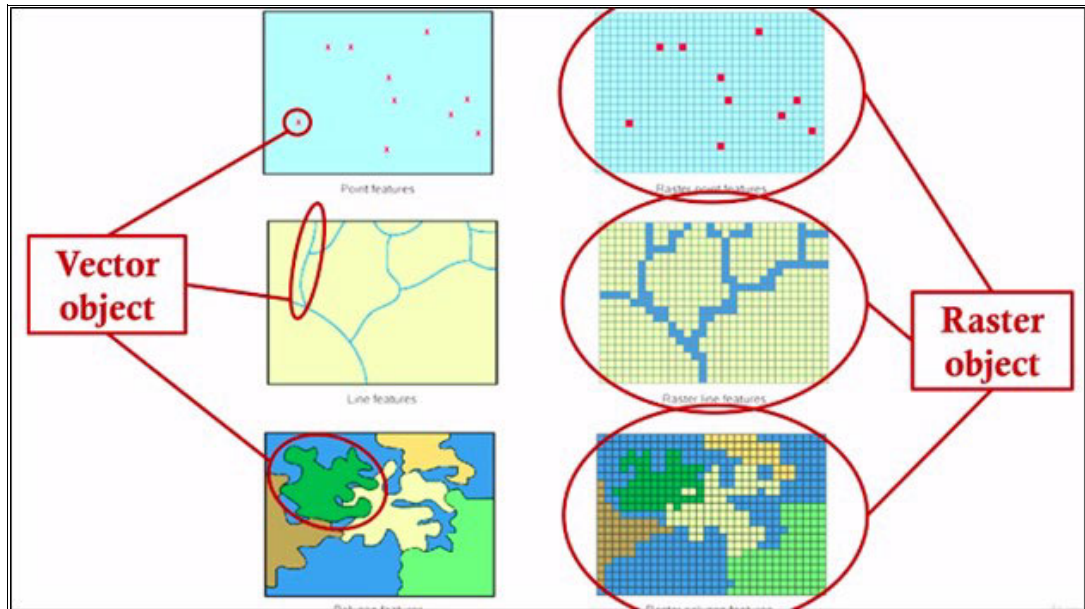


Figure 8-6 Vector and raster grid models

- ▶ **Box3D**: Returns the box 3D representation of the enclosing box of the raster.
- ▶ **ST_Clip**: Returns the raster that is clipped by the input geometry. If no band is specified, all bands are returned. If crop is not specified, true is assumed, which means that the output raster is cropped.
- ▶ **ST_ConvexHull**: Returns the convex hull geometry of the raster, including pixel values that are equal to BandNoDataValue. For regular-shaped and non-skewed rasters, it provides the same result as ST_Envelope, so it is useful only for irregularly shaped or skewed rasters.
- ▶ **ST_DumpAsPolygons**: Returns a set of geometry value (geomval) rows from a raster band. If no band number is specified, the band number defaults to 1.
- ▶ **ST_Envelope**: Returns the polygon representation of the extent of the raster.
- ▶ **ST_HillShade**: Returns the hypothetical illumination of an elevation raster band by using the provided azimuth, altitude, brightness, and elevation scale inputs. Useful for visualizing terrain.
- ▶ **ST_Aspect**: Returns the surface aspect of an elevation raster band. Useful for analyzing terrain.
- ▶ **ST_Slope**: Returns the surface slope of an elevation raster band. Useful for analyzing terrain.
- ▶ **ST_Intersection**: Returns a raster or a set of geometry-pixel value pairs representing the shared portion of two rasters or the geometrical intersection of a vectorization of the raster and a geometry.
- ▶ **ST_Polygon**: Returns a polygon geometry that is formed by the union of pixels that have a pixel value that does not have a data value. If no band number is specified, the band number defaults to 1.

- ▶ **ST_Reclass**: Creates a raster that is composed of band types that are reclassified from original. The nband is the band to be changed. If nband is not specified, it is assumed to be 1. All other bands are returned unchanged. For example, you can convert a 16BUI band to an 8BUI for simpler rendering as viewable formats.
- ▶ **ST_Union**: Returns the union of a set of raster tiles into a single raster that is composed of one band. If no band is specified for union, band number 1 is assumed. The resulting raster's extent is the extent of the whole set. In intersection, the resulting value is defined by p_expression, which is one of the following values: LAST (the default when none is specified), MEAN, SUM, FIRST, MAX, and MIN.

Figure 8-7 shows raster data that is overlaid onto maps.

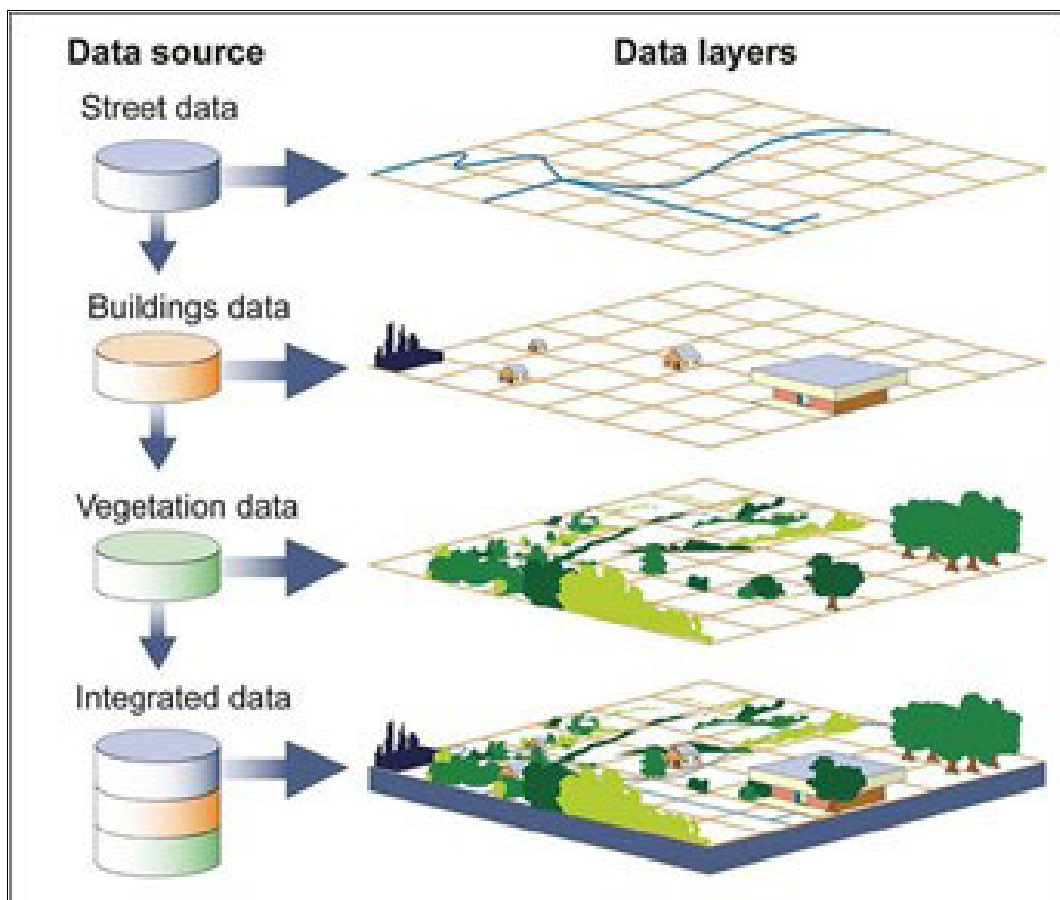


Figure 8-7 Raster data overlaid onto maps

Working with other API or GIS formats

Functions can also be run against external data formats, for example, binary images and maps, file strings, and JSON representations.

- ▶ **ST_Box2dFromGeoHash**: Returns a BOX2D object from a GeoHash string
- ▶ **ST_GeomFromGeoHash**: Returns a geometry object from a GeoHash string.
- ▶ **ST_GeomFromGeoJSON**: Takes as input a geojson representation of a geometry and outputs a PostGIS geometry object.
- ▶ **ST_GMLToSQL**: Returns a specified ST_Geometry value from GML representation. This name is an alias for ST_GeomFromGML.
- ▶ **ST_LineFromEncodedPolyline**: Creates a linestring from an encoded polyline.

- ▶ [ST_PointFromGeoHash](#): Returns a point from a GeoHash string.
- ▶ [ST_FromFlatGeobuf](#): Reads FlatGeobuf data.

Summary

Many of the sample functions that we have shown here are complex in nature, and they cast or explicitly call C programs to read, compare, transform, and process several geospatial data sources. They are intended to compliment or even replace the traditional data science approach of using statistical analysis in classifying data, identifying similarities, and predicting trends. Using FUJITSU Enterprise Postgres to support PostGIS functions on IBM LinuxONE provides a high-performance platform that is scalable and has advanced security.

8.1.4 Urban landscaping use case

Many organizations require detailed geospatial data to evaluate the potential location for a new business or decide how best to maximize the local facilities, predict traffic flows during peak and non-peak times, and promote their business to the correct clientele in the correct areas.

One such service provider in the UK, Space Syntax Ltd, developed a rich PostgreSQL-based platform with extensive socioeconomic data running on IBM LinuxONE. This platform contains ordnance survey data, a rich and detailed mapping source for the UK, and non-UK mapping data sources to support projects around the world. This platform is embellished with social-economic data, which includes population information such as demographics, travel, education, welfare, healthcare, and financial-related data.

Figure 8-8 shows a cycle network analysis that was done by Space Syntax for the Department for Transport.

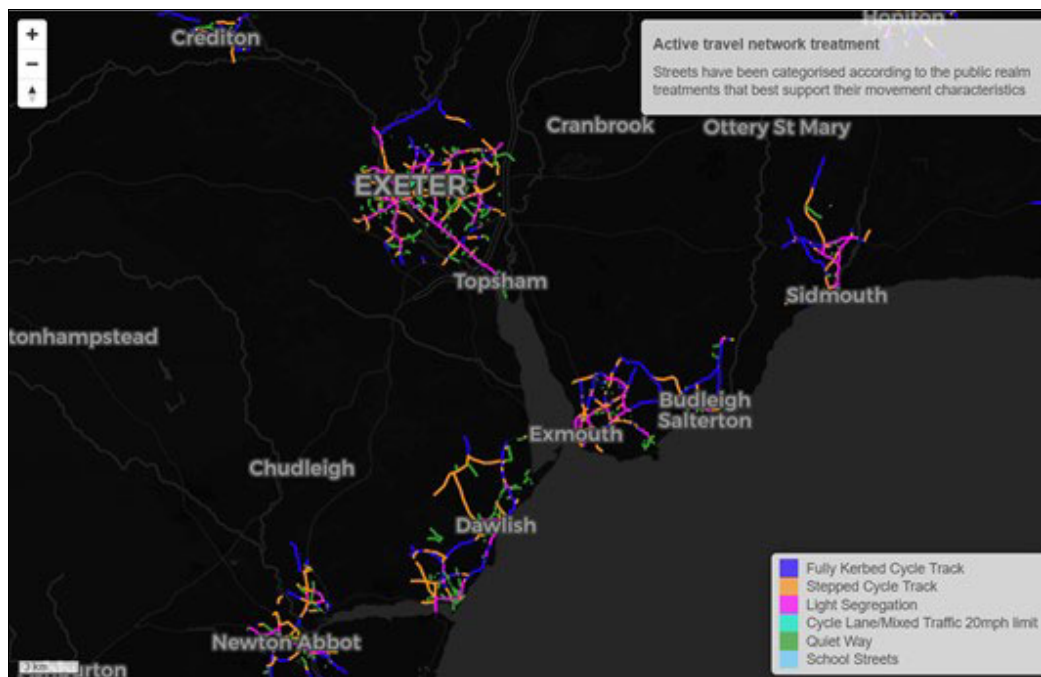


Figure 8-8 Space Syntax: Department for Transport cycle network analysis

Using a combination of queries that uses PostGIS spatial functions, Figure 8-9 shows how government-defined cycle network treatments are assigned to specific parts of the street network based on how likely each street segment is to be used for journeys by bike, a mix of adjacent land uses, and the number of traffic pedestrians.

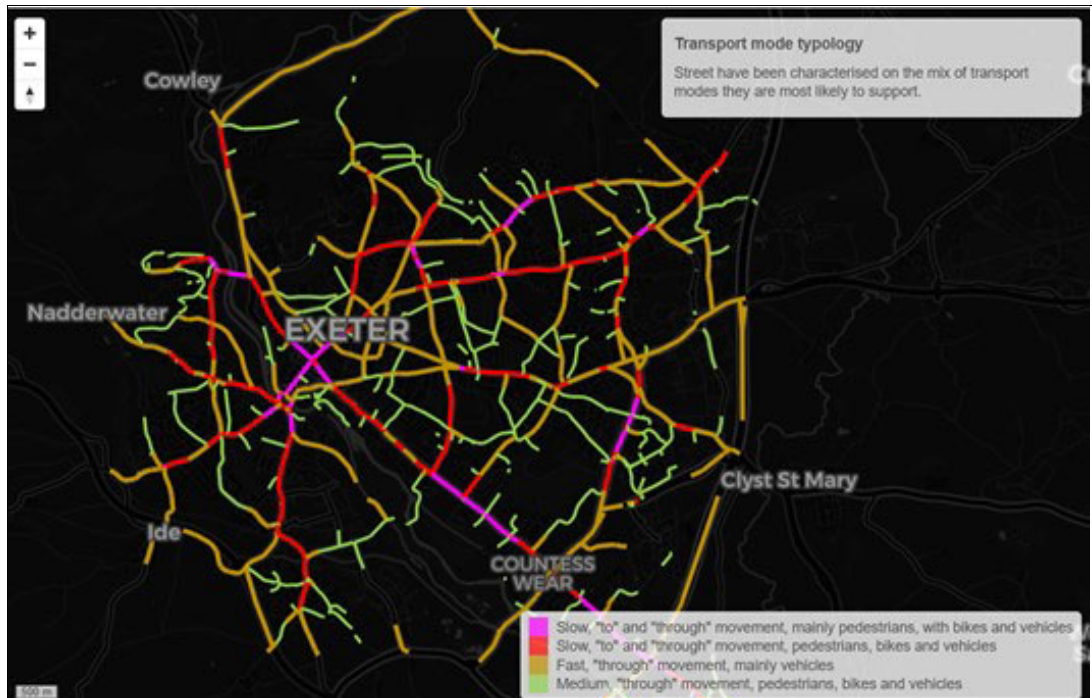


Figure 8-9 Space Syntax: Cycle network that is categorized according to character of movement

The queries that are shown in Figure 8-10 on page 315 are spatial queries that run by using multiple data sources that are available exclusively for approved partners by the UK Geospatial Commission. These sources include active travel routes and transport data, social data that is based on census polls, and Ordnance Survey Mastermap data. The Mastermap includes the core location identifiers (Unique Property Reference Numbers (UPRNs), Unique Street Reference Numbers (USRNs), and the Topographic Object Identifier (TOID)) that provide a golden thread to link a wide range of data sets together to provide insights that otherwise are not possible.

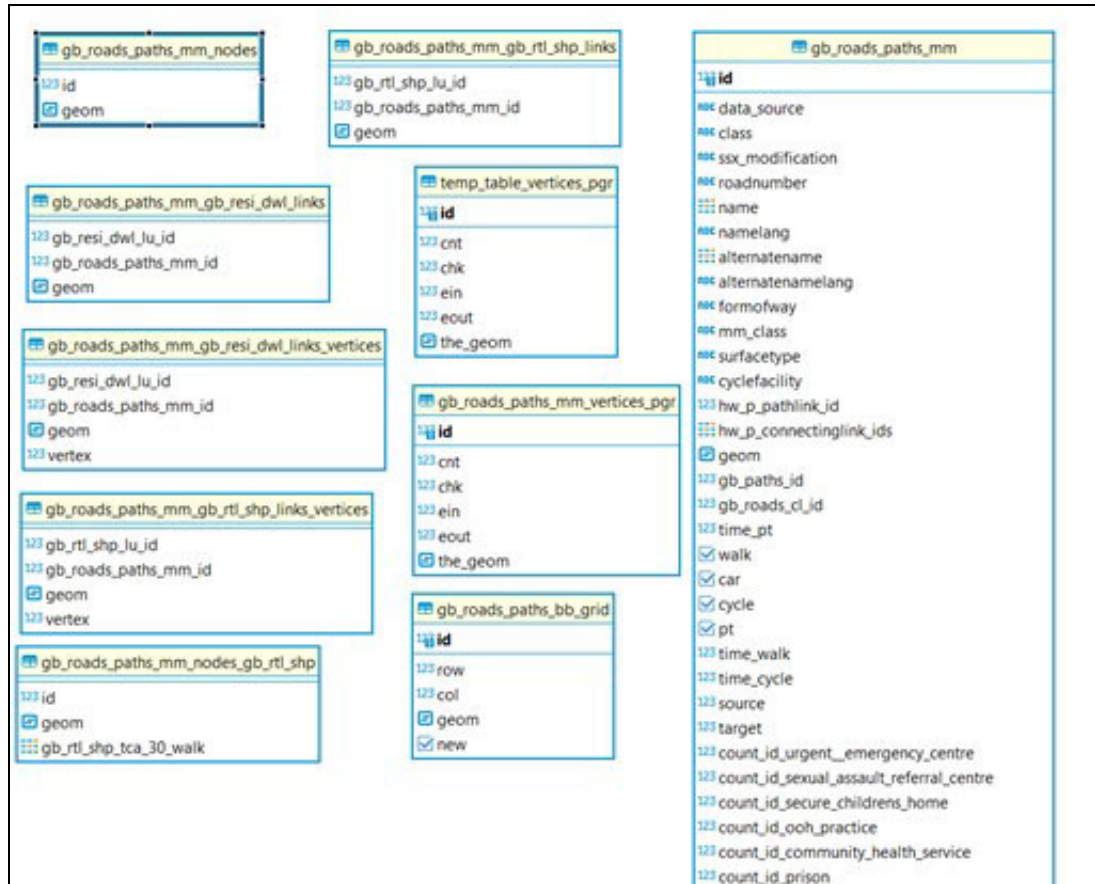


Figure 8-10 UK-based geospatial data model

The large tables contain the core road, pavement, cycle pathways, and commercial buildings data, which contains over 100 million combined records. To determine the quickest routes between point A -> B in a certain period, geospatial reference queries and functions were developed. The location grid coordinates are provided, and the query plots the geometrical positions and then calculates, based on all the available data, the potential distance by road, cycle route, or pavement within the given period. Results vary from seconds to hours based on the location, such as rural, semi-rural, town, or city, and the given time, such as 15 minutes to several hours.

While implementing these tables, the key geometry fields are added to spatial indexes. With the spotty temporal library, you can use functions to index points within a region, on a region containing points, and points within a radius to enable fast queries on this data during location analysis. As the SQL shows in Example 8-4 on page 316, the sample query uses an index scan within the `ST_Intersects` PostGIS function to improve the comparison between two `geom` columns. This spatial index was created with the following statement:

```
CREATE INDEX sidx_gb_rtl_shp_geom ON uk_landuse.gb_rtl_shp USING gist (geom)
```

Example 8-4 shows a portion of the code that is required to interpret this geospatial data (the remainder is commercially sensitive, so it is not shown here).

Example 8-4 Sample query portion to interpret geospatial data

```
SELECT array_agg(distinct vertex)
      FROM (
        SELECT vertex
        FROM uk_landuse.'|| current_setting('vars.myvar2')||' AS o
        INNER JOIN uk_landuse.'|| current_setting('vars.myvar2')||'_jobs_walk_30
AS g
        ON o.geom&&g.geom AND ST_Intersects(o.geom, g.geom)
        INNER JOIN uk_road.gb_roads_paths_mm.'||
current_setting('vars.myvar2')||'_links_vertices 1
        ON o.lu_id = 1.'|| current_setting('vars.myvar2')||'_lu_id
        WHERE g.id = '|| current_setting('vars.myvar')||'
        ORDER BY o.lu_id, vertex
        OFFSET ('||chunk_seq_var||' - 1)*'||chunk_size||'*2.0
        LIMIT '||chunk_size||'*2.0) a;'
```

Figure 8-11 shows the results of the program.

#	Node	Plan	rows
1.	→ Aggregate (cost=473.43..473.44 rows=1 width=32)		1
2.	→ Limit (cost=473.41..473.41 rows=1 width=8)		1
3.	→ Sort (cost=469.26..473.41 rows=1659 width=8)		1659
4.	→ Nested Loop Inner Join (cost=0.71..380.54 rows=1659 width=8)		1659
5.	→ Nested Loop Inner Join (cost=0.29..56.78 rows=416 width=4)		416
6.	→ Seq Scan on uk_landuse.gb_rtl_shp_jobs_walk_30 as g (cost=0.23..46 rows=1 width=120) Filter: (g.id = 7)		1
7.	→ Index Scan using sidx_gb_rtl_shp_geom on uk_landuse.gb_rtl_shp as o (cost=0.29..33.3 rows=1 wi... Filter: st_intersects(o.geom, g.geom) Index Cond: ((o.geom && g.geom) AND (o.geom && g.geom))		1
8.	→ Index Scan using sidx_gb_roads_paths_mm_gb_rtl_shp_links_vertices_lu_id on uk_road.gb_roads_paths... Index Cond: (l.gb_rtl_shp_lu_id = o.lu_id)		4

Figure 8-11 Query results

The results were captured as data sets from which individual locations were transposed into graphical representations (Figure 8-12).

tile_id	geom	chunk_seq	chunk_size	count_origin	error	minutes	id
114	002000000300006C3400000001000000054	1	2500	1179		2.008836	1
114	002000000300006C3400000001000000054	2	2500	1147		2.792562	2
114	002000000300006C3400000001000000054	3	2500	1389		5.370199	3
114	002000000300006C3400000001000000054	4	2500	1407		6.513321	4
114	002000000300006C3400000001000000054	5	2500	1554		6.451666	5

Figure 8-12 Results data set

Summary

In summary, this analysis is complex: It intersects multiple data sources and places them into a positional framework, breaks down each geometric location into tiles and chunks, and then applies the socioeconomic data patterns.

FUJITSU Enterprise Postgres with the PostGIS extension running on IBM LinuxONE provides the ideal environment to run this code against the geospatial data to ensure strong performance, resilience, data integrity, and security.

8.2 MongoDB as a Service

Has continued growth for new and existing services pushed current infrastructures to their limits? Did the chip shortage slow growth potential? Is recovery from corruption easy? Is recovery from ransomware type events possible with existing systems? Can you maintain the cost of cloud services while protecting the data that is required to complete all transactions? Does it look like carbon-neutral targets can be achieved by 2025 while maintaining the demand that is created by your current server sprawl?

What if it is possible to resolve all these business issues by using today's mainframe technology?

This section describes how IBM LinuxONE, when combined with IBM Storage, provides superior availability, performance, and security than competing platforms by using a sample anonymized client environment. With the potential to reduce power requirements by approximately 70% while decreasing the footprint that is required to run equivalent services on x86 servers by up to 50% in a study that is associated with this type of consolidation effort, it is easy to see the start of your potential savings along with the reduced carbon footprint that this solution provides³.

Figure 8-13 shows the sample comparison that is used for this chapter.

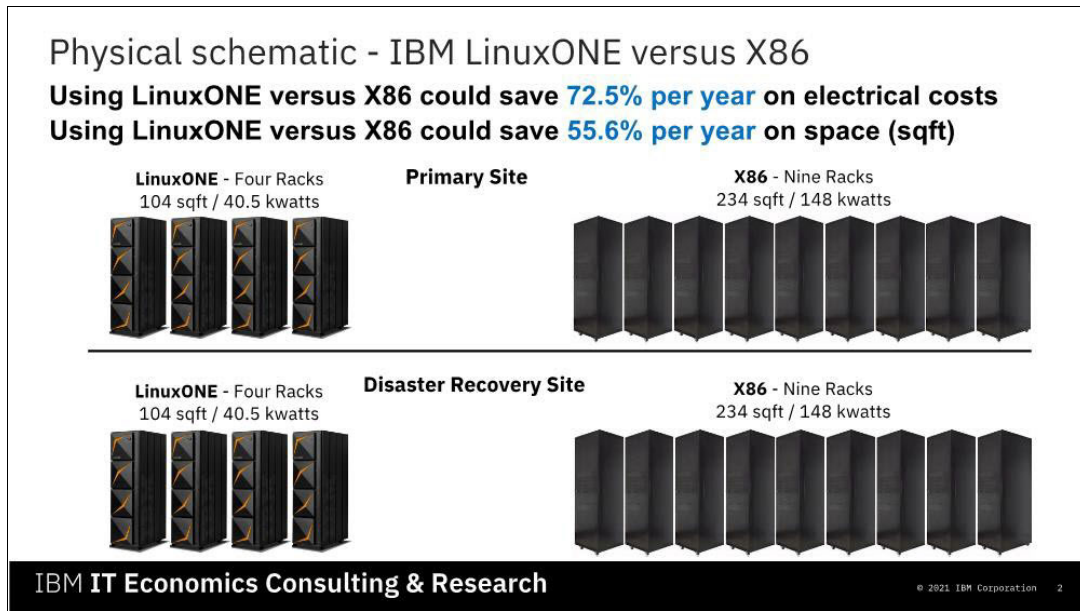


Figure 8-13 IBM LinuxONE versus x86

8.2.1 IBM lab environment

IBM and Sine Nomine Associates (SNA) built an internal environment that mimicked a simplified version of a single-cluster client environment. This environment was hosted at the IBM Systems Client Engineering group (previously referred to as Garage for Systems). The environment is available as a demonstration outside the IBM lab. The environment was replicated on a much larger scaler at several client sites.

³ The IBM Economics Consulting and Research team provided slightly better numbers for this specific consolidation effort. Refer to the link to find your savings based on your quantifiable metrics. You can save costs and gain other benefits by consolidating servers on IBM LinuxONE systems.

Figure 8-14 shows a representation of the emulated data center.

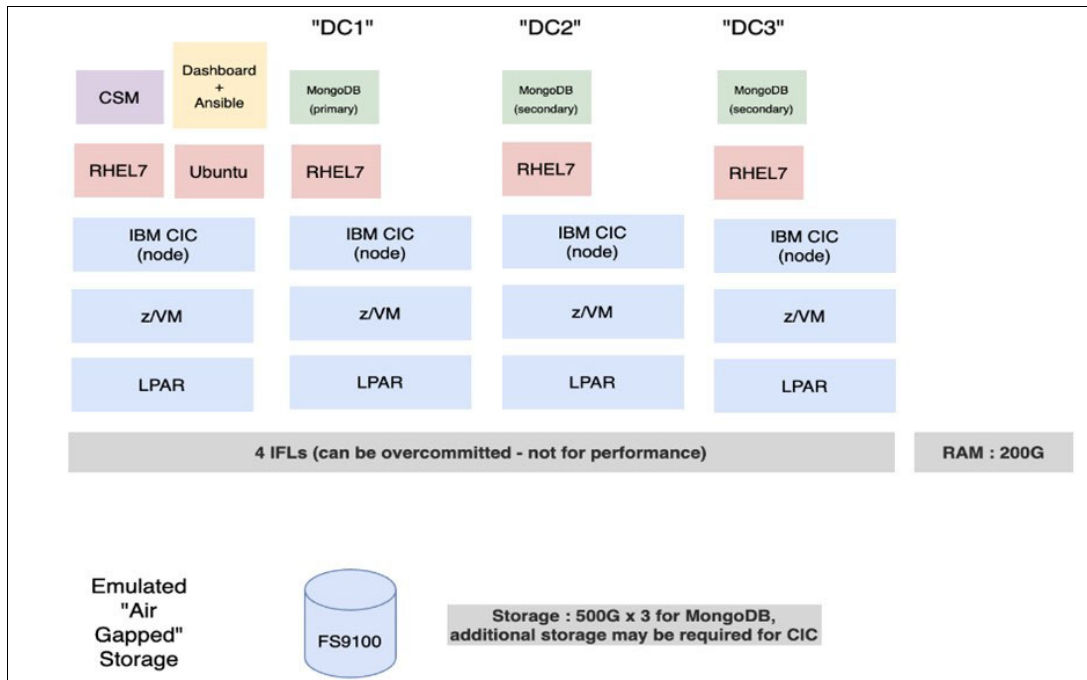


Figure 8-14 Emulated data center

Figure 8-15 shows how the environment was expanded to three active data centers for our use: Poughkeepsie, New York, US (POK), Montpellier France (MOP), and the Washington System Center (WSC) in Herndon, Virginia, US.

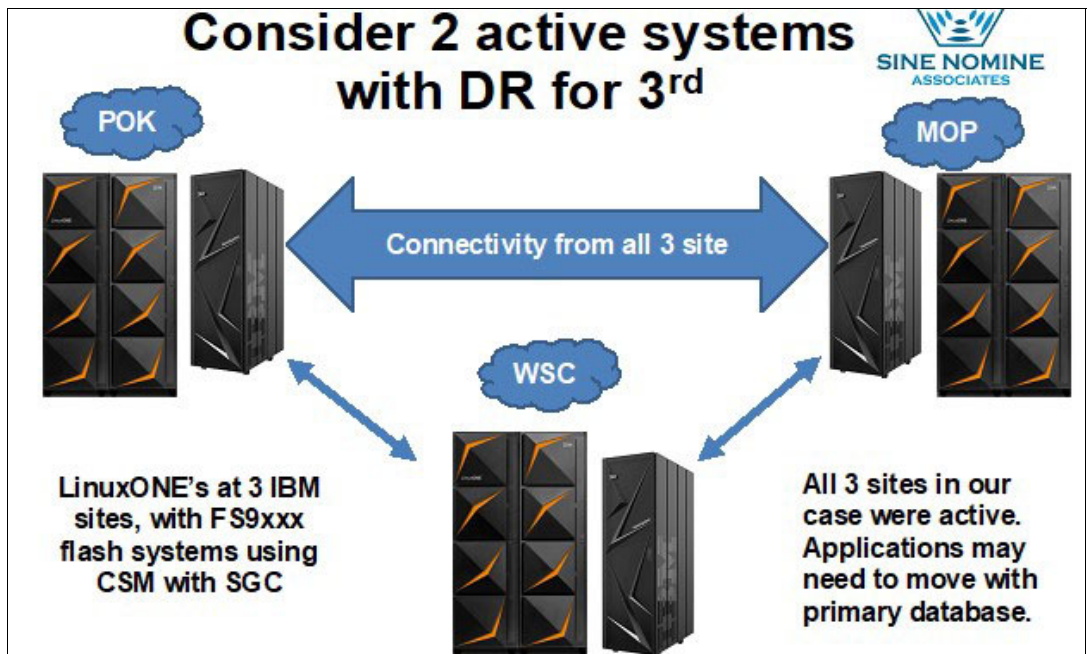


Figure 8-15 Expanded lab environment

The three MongoDB database instances were deployed with IBM Cloud Infrastructure Center (IBM CIC) at each data center while keeping all the nodes in a cluster geographically dispersed. Connectivity was possible by using multiple virtual private networks (VPNs) for increased visibility and availability into the collective systems. A mix of IBM FlashSystem® 9100 and IBM FlashSystem 9200 were used for storage, with all of them configured to levels that included IBM Safeguarded Copy (IBM SGC) V8.4.0 for the controllers. The IBM SGC volumes are managed and accessed with CSM for recovery, but creation was handled by automated policies on the storage devices.

IBM CIC does not contain a native mechanism for immutable snapshots because that capability is provided by the IBM FlashSystem 9x controller and Copy Services Manager.

Figure 8-16 shows a sample GUI that you can use to complete the following tasks:

- ▶ Select the size of the MongoDB database that you are creating based on standard T-Shirt sizes.
- ▶ Select a checkbox to add Appendix J (App J) to the MongoDB databases that you are creating that require it.

Now, when a new MongoDB cluster is created, the App J Compliant checkbox can be selected and IBM SGC copies start based on the policy that are defined for them.

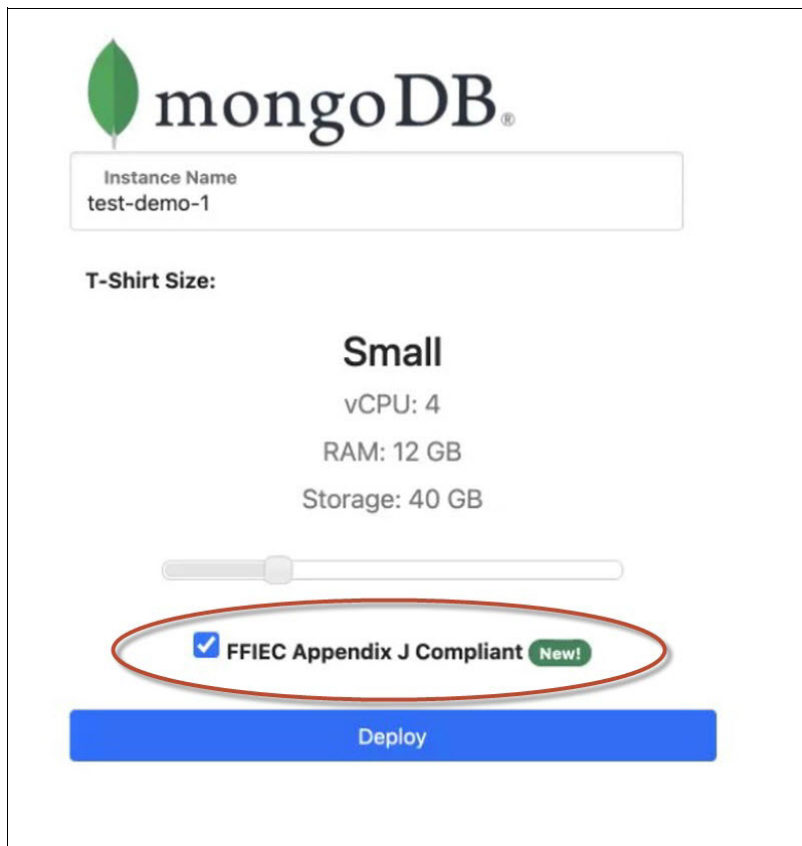


Figure 8-16 Selecting the Appendix J Compliant checkbox

Federal Financial Institutions Examination Council Appendix J

The Federal Financial Institutions Examination Council (FFIEC) is a regulatory body for financial institutions in the US. As part of its regulations, App J was added as an additional appendix that sets standards for the resilience of outsourced technology. As with other regulations, App J is open to interpretation, and the definition of being FFIEC App J compliant

changes from organization to organization. Work with your organization's compliance group to determine what their technical and process checklist entails.

Appendix J, IBM Safeguarded Copy, and data serving

For our sample client, App J compliance meant building a cyberattack resilient environment. A key part of the implementation was the ability to generate immutable "air-gapped" copies of the live production data that was insulated from production access and protected from attack vectors such as malware and ransomware. IBM SGC provides a hardware-based solution that ensures the immutability of data. For our sample client use case, IBM SGC was sufficient to satisfy the air-gapped immutability aspect of the compliance document.

► Frequency and retention period

The lowest frequency for IBM SGC copies is 1 minute with a 1-day retention. However, using this frequency would deplete rapidly the storage that is needed for immutable snapshots based on write-rates, and so this frequency is impractical for real-world use cases. Most frequencies are set to a 30-min to 4-hour frequency with multi-day retention ranging from 3 days to several months (for less frequent snapshots), so a cyberattack might not be noticeable for months. As a best practice, use longer retention periods or complement on-device immutable copies with "cold" storage-immutable copies that are less performant and slower to recover, but are much more cost-effective.

In this case of MongoDB, the MongoDB live data (typically on `/var/lib/mongo` by default) required immutable air-gapping from its live production systems. In addition to the data being unmodifiable, the addition constraint of being un-erasable was also included under the definition of "immutable".

For the sample environment, we defined a 30-minute recovery time objective (RTO) and recovery point objective (RPO). By using a round-robin algorithm, each database was backed up 10 minutes apart to keep the backups within the defined range so that in the worst case, only 20 minutes of stale data would exist (within our 30-minute RPO).

► Application consistency versus crash consistency

Application consistency assumes that the application layer cleanly quiesces the application or DB before shutdown. In a real cyberattack, it is imprudent to expect or depend on application consistency. However, crash consistency treats failures as though power from the server was disconnected and caused an instantaneous or forced shutdown of the entire stack. MongoDB, with its journaling and checkpointing capabilities, offers both application consistency (with manual scripting and synchronization) and crash consistency (with a storage layer snapshot). With the default setting, the MongoDB built-in crash consistency (a loss of a few seconds at most) is sufficient to meet most RPO (for example, 30 minutes) where MongoDB is used. Zero RPO is possible, but at the expense of performance, and there are better databases that are suited for that use case.

Figure 8-17 provides a quick outline of the two main types of attacks against data today, along with the recovery efforts that are needed. Although it is possible to use an IBM SGC copy of the data for normal database restoration or recovery, these efforts are part of normal backup and restore practices so that the application and Linux teams can recover from general inconsistencies that are discovered during normal operations.



Figure 8-17 Main types of attacks with recovery efforts

The following automation scripts are available online from MongoDB to help with recovery:

- ▶ [Deploy Automatically with GitHub](#)
- ▶ [Backup and Restore with Filesystem Snapshots](#)

The [Configure LVM logical volumes](#) Ansible playbook was used to create the Logical Volume Management (LVM) and snapshot area.

We completed the following tasks:

1. Three-node cluster deployment automation.
2. Enablement of IBM SGC copies.
3. Validation of a “clean” (untainted) copy or snapshot.
4. Intentional corruption of live data.
5. Recovery.
6. Business continuity returned.

In the sample environment, we did not include automation to determine which copy was tainted because that task was beyond the scope of the project. It is not a hard prerequisite for App J compliance. In typical real-world scenarios, an organization has several options for this task:

- ▶ Validation before snapshot creation.
- ▶ Asynchronous validation.
- ▶ Validation on detection of a cyberattack in another workload (post-mortem).

IBM has solutions for all these options, but they are beyond the scope of this book.

Figure 8-18 on page 322 shows a diagram of IBM Copy Services Manager with a 5-minute frequency and 1-day retention. 5 minutes was set up for demonstration purposes because waiting 30 minutes or multiple hours is not practical to showcase this technology.

The screenshot displays the IBM Copy Services Manager interface for a session named 'mongoDB_SGC_LBSSVC6'. The session is in a 'Normal' state, protected, and is a 'Safeguarded Copy' on host 'H1'. It is recoverable and has a description of 'Automatically created Safeguarded Copy session(modify)'. The backup schedule is set to 'Every 5 mins', with the last recoverable backup occurring on '2021-09-13 20:06:39 MST' in the 'mongoDB_SGC' volume group.

Below the session details, there is a table showing backup information:

Backup Time	Backup ID	Recoverable	Copy Sets	Last Result	Expiration
2021-09-13 16:56:34 MST	1631577600	Yes	4	✓ IWNR2800I	2021-09-20 16:56:34 ...
2021-09-13 17:01:34 MST	1631577900	Yes	4	✓ IWNR2800I	2021-09-20 17:01:34 ...
2021-09-13 17:03:09 MST	1631577992	Yes	4	✓ IWNR2800I	2021-09-14 17:03:09 ...
2021-09-13 17:06:34 MST	1631578200	Yes	4	✓ IWNR2800I	2021-09-20 17:06:34 ...
2021-09-13 17:11:34 MST	1631578500	Yes	4	✓ IWNR2800I	2021-09-20 17:11:34 ...
2021-09-13 17:16:34 MST	1631578800	Yes	4	✓ IWNR2800I	2021-09-20 17:16:34 ...
2021-09-13 17:21:34 MST	1631579100	Yes	4	✓ IWNR2800I	2021-09-20 17:21:34 ...
2021-09-13 17:26:34 MST	1631579400	Yes	4	✓ IWNR2800I	2021-09-20 17:26:34 ...
2021-09-13 17:31:34 MST	1631579700	Yes	4	✓ IWNR2800I	2021-09-20 17:31:34 ...

Figure 8-18 IBM Copy Services Manager with 5-minute frequency and 1-day retention

8.2.2 Deployment automation overview

The lifecycle of replica sets in MongoDB consists of several steps, which are briefly described here:

1. Create a base image that is used in all deployments (, “Base image deployment overview” on page 323).
2. Deploying a set of virtual machines (VMs) that make up a replica set (, “Replica set virtual machine instantiation overview” on page 323) or a shadow set of VMs (, “Shadow overview” on page 323).
3. Deleting a replica set or its shadows.

In addition, there are several supporting playbooks to perform tasks such as:

1. Quiescing and resuming the database so that backups may be taken.
2. Terminating the replica set gracefully.

In general, the process of enabling IBM SGC copies for a general environment is as follows:

1. Gather a list of volumes and respective storage devices.
2. Ensure that the storage devices have the Safeguarded Policy available.
3. Ensure that there is sufficient space for Safeguarded Pools.
4. Create a volume group (with Safeguarded Policy attached).
5. Attach volumes to a volume group.
6. (optional) Validate enablement in CSM (in about 2 - 3 minutes or by logging in to the storage device and inspecting the Safeguarded pool).

Base image deployment overview

This section describes the procedures for creating the base image for MongoDB deployment. Creating such an image provides a common base from which to work.

The process is controlled by a playbook with a parameter file that defines locations, credentials, and other important information.

- ▶ **Playbook:** The playbook performs the following tasks:
 - a. Updates the host's address from the deployment data.
 - b. Adds the name servers as specified in the parameter file.
 - c. Refreshes the Red Hat Enterprise Linux subscription manager data.
 - d. Updates the system and installs some extra packages.
 - e. Cleans the YUM cache.
 - f. Captures the VM as an image.
- ▶ **Parameter file:** This JSON file contains parameters that are used by the playbook to construct a base image.
- ▶ **Base inventory file:** This trivial file is used by the playbook so that you can create an IP address for the instantiated VM:

```
base_image
```

Replica set virtual machine instantiation overview

The provisioning of a replica set in MongoDB is a multistep process. The first step is provisioning the following items:

1. VMs

Ansible uses the OpenStack module to create three VMs for use as a Mongo replica set.

2. Network resources

The network parameters are used to associate a network with the provisioned VMs.

3. Data volume

Ansible uses the OpenStack module to define and attach a Mongo data volume to each of the provisioned VMs.

These tasks are performed by using Ansible playbooks. In addition, there are playbooks for creating shadow VMs, which can be used for recovery. The process is almost identical to the steps that are outlined in this section. The differences are described in , “Shadow instance deployment” on page 336.

The Ansible configuration consists of a playbook and a set of tasks that is repeated for each of the VMs that are being provisioned.

Shadow overview

When recovering a replica, here are the three options:

1. Use the existing VMs and replace the Mongo data volume.
2. Create VMs that mimic the size of the original.
3. Create shadows of the original (asynchronously during deployment).

Table 8-1 provides a list of the pros and cons of each option.

Table 8-1 Comparison table of the three options:

Mechanism	Pros	Cons
Reuse VMs.	<ul style="list-style-type: none"> ▶ Single set of VMs. ▶ No networking changes are required. 	<ul style="list-style-type: none"> ▶ Elegant malware is hard to detect, and a “clean state” might be impossible to determine. ▶ If the host is still compromised, the freshly recovered data might be compromised again. ▶ False recovery: Recovery and attachment might seem successful, but the malware might stay dormant and impact the data silently later (weeks or months).
Fresh VM deployment.	A clean state that is clear of malware.	<ul style="list-style-type: none"> ▶ Time-consuming (< 5 minutes) but reasonable given longer RTO windows.
Shadowing.	<ul style="list-style-type: none"> ▶ A clean state that is clear of malware. ▶ Quicker than spinning up a fresh VM because starting the instances is near instantaneous. 	<ul style="list-style-type: none"> ▶ Networking addresses or names need changing to match what the Mongo data expects.

MongoDB deployment overview

In this section, we describe several Ansible playbooks and supporting files that perform tasks that affect and effect the operation of MongoDB.

The mongo-operations playbooks were created to facilitate the deployment and operation of Mongo instances.

Required files

There are two base files that are used to define and control the Ansible environment:

1. `ansible.cfg`
2. `hosts`

The `hosts` file is created by running a script that is invoked by the playbook, which passes the IP names and addresses of the nodes.

Ansible configuration

Certain settings in Ansible are adjustable through a configuration file. By default, this file is `/etc/ansible/ansible.cfg`, and for this project, a simple file is in a local directory, which is shown in Example 8-5.

Example 8-5 Configuration settings in Ansible

```
[defaults]
inventory = hosts
host_key_checking = False
```

The important entry here is `inventory`, which tells Ansible where to find a definition of all the endpoints to be made known to Ansible.

Hosts

Hosts can be defined in Ansible in many different ways. In our simple implementation, we use a flat file in the `.ini` format that is created by a script that is invoked from a playbook. An

example is show in , “Hosts file” on page 338. The file is named after the replica set. This file is used to create shadows, terminate the replica set, and destroy the replica set.

Deployment

Deployment consists of running a deployment playbook against the hosts that are defined in the hosts file.

Playbooks

The deployment playbook takes a base RHEL 7 system and performs the following tasks:

1. Install software, which includes MongoDB and supporting software.
2. Install configuration files.
3. Define an admin user for Mongo.
4. Enable operation in an SELinux enforcing environment.

Tasks

The deployment playbook (, “MongoDB deployment overview” on page 324) prepares the environment for a working MongoDB installation.

Supporting files

The following files are used to support the deployment process:

► SELinux files

Two policies must be created and installed:

- a. Enable Full Time Diagnostic Data Capture (FTDC).
- b. Allow access to `/sys/fs/cgroup`.

► Proc policy

The current SELinux policy does not allow the MongoDB process to open and read `/proc/net/netstat`, which is required for FTDC.

To create the policy that is used by the playbook, run the script that is shown in Example 8-6 to create `mongodb_proc_net.te`.

Example 8-6 Creating the policy

```

module mongodb_proc_net 1.0;
require {
    type proc_net_t;
    type mongod_t;
    class file { open read };
}
#===== mongod_t =====
allow mongod_t proc_net_t:file { open read };

```

Convert the policy into an SE module by running the following commands:

- **checkmodule -M -m -o mongodb_proc_net.mod mongodb_proc_net.te**
- **semodule_package -o mongodb_proc_net.pp -m mongodb_proc_net.mod**

► CGroup memory policy

The current SELinux Policy does not allow the MongoDB process to access `/sys/fs/cgroup`, which is required to determine the available memory on your system.

To create the policy that is used by the playbook, run the script that is shown in Example 8-7.

Example 8-7 Creating the CGroup memory policy

```
mongodb_cgroup_memory.te:
module mongodb_cgroup_memory 1.0;
require {
    type cgroup_t;
    type mongod_t;
    class dir search;
    class file { getattr open read };
}
#===== mongod_t =====
allow mongod_t cgroup_t:dir search;
allow mongod_t cgroup_t:file { getattr open read };
```

Convert the policy into an SE module by running the following commands:

- **checkmodule -M -m -o mongodb_cgroup_memory.mod mongodb_cgroup_memory.te**
- **semodule_package -o mongodb_cgroup_memory.pp -m mongodb_cgroup_memory.mod**

► JavaScript files

- rs.js

The JavaScript file that is shown in Example 8-8 is generated by a script that is invoked from the playbook and used to initiate the replica set.

Example 8-8 The rs.js file

```
try {
    if (rs.status().code != "") {
        rs.initiate( {
            _id: "rs0",
            members: [
                { _id: 0, host: "MONGO-RS1-1:27017" },
                { _id: 1, host: "MONGO-RS1-2:27017" },
                { _id: 2, host: "MONGO-RS1-3:27017" }
            ]
        })
    }
} catch {
}
```

– admin.js

The JavaScript file that is shown in Example 8-9 adds the user `admin` to the `admin` database. The `admin` password is defined in this script and should be changed to meet your requirements.

Example 8-9 admin.js

```
try {
  rs.secondaryOk()
  db = connect('admin')
  adminUser = db.system.users.find({user:'admin'}).count()
  if (adminUser == 0) {
    db.createUser( {
      user: "admin",
      pwd: "xxxxxxx",
      roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
    }
  )
}
} catch {
}
```

Other playbooks

Here are some other relevant playbooks:

► Quiesce

The quiesce playbook (, “Quiescing” on page 346) locks the database to prevent it from being updated. You are prompted for the `admin` password.

► Resume

The resume playbook (, “Resuming” on page 346) unlocks the database to allow updates to proceed. You are prompted for the `admin` password.

► Shutdown

The shutdown playbook (8.2.5, “Terminating” on page 347) uses `systemd` to gracefully shut down a replica set.

Destroying replica sets

The decommissioning of a MongoDB replica set is a multi-step process, the first step of which is the provisioning of the following items:

► VMs

► Data volume

These tasks are performed by using Ansible playbooks.

Virtual machine deletion

Ansible uses the `OpenStack` module to delete the nodes that are defined in the inventory file that is specified.

Data volume

Ansible uses the `OpenStack` module to delete the MongoDB data volume of each provisioned VM.

Playbook and tasks

The Ansible configuration consists of a playbook and a set of tasks that is repeated for each of the VMs being provisioned. For more information, see 8.2.5, “Terminating” on page 347.

8.2.3 Playbooks

This section provides an annotated description of the playbooks that are used to create images and deploy MongoDB replica sets.

Base deployment

This playbook creates a base image for use by other deployment processes. Example 8-10 shows our sample playbook. The line numbers are for this and the following examples only so that you can easily reference the explanations that follow.

Example 8-10 Playbook for base deployment

```
[000] ---
[001]
[002] - name: Create a Base Image
[003]   hosts: all
[004]   gather_facts: no
[005]
[006]   tasks:
[007]     - name: Check for existing image
[008]       register: image
[009]       local_action:
[010]         module: openstack.cloud.image_info
[011]         image: "{{ cic_base_name }}-IMAGE"
[012]
[013]     - name: Set Image Data
[014]       local_action:
[015]         module: set_fact
[016]         id: "% if image.openstack_image %>{{ image.openstack_image.name }}{%
else %>{{ cic_rhel_image }}{% endif %}"
[017]
[018]     - name: Deploy a Starting Image
[019]       register: deployed_vm
[020]       local_action:
[021]         module: openstack.cloud.server
[022]         name: "{{ cic_base_name }}"
[023]         image: "{{ id }}"
[024]         key_name: "{{ cic_key_name }}"
[025]         availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[026]         flavor: "{{ cic_flavor }}"
[027]         security_groups: default
[028]         network: "{{ cic_vlan }}"
[029]         volume_size: 5
[030]         timeout: 1800
[031]         wait: true
[032]
[033]     - name: Update Inventory IP address
[034]       set_fact:
[035]         ansible_ssh_host: "{{ deployed_vm.openstack.public_v4 }}"
[036]
[037]     - name: Remove /etc/resolv.conf
```



```
[038] file:
[039]   path: /etc/resolv.conf
[040]   state: absent
[041]
[042] - name: Create new /etc/resolv.conf
[043]   file:
[044]     path: /etc/resolv.conf
[045]     state: touch
[046]     owner: root
[047]     group: root
[048]     mode: 0644
[049]
[050] - name: Add content to /etc/resolv.conf
[051]   blockinfile:
[052]     path: /etc/resolv.conf
[053]     block: |
[054]       {% for dns in cic_dns %}
[055]         nameserver {{ dns }}
[056]       {% endfor %}
[057]
[058] - name: Clean Subscription Manager
[059]   command: subscription-manager clean
[060]
[061] - name: Remove Old Subscription Manager
[062]   yum:
[063]     name: katello-ca-*
[064]     state: absent
[065]     update_cache: yes
[066]
[067] - name: Add New Subscription Manager
[068]   yum:
[069]     name: "{{ sub_rpm }}"
[070]     state: present
[071]     update_cache: yes
[072]
[073] - name: Register and auto-subscribe
[074]   community.general.redhat_subscription:
[075]     state: present
[076]     org_id: "{{ sub_org }}"
[077]     activationkey: "{{ sub_key }}"
[078]     ignore_errors: yes
[079]
[080] - name: Upgrade System
[081]   yum:
[082]     name=*
[083]     state=latest
[084]
[085] - name: Install Tools
[086]   yum:
[087]     name: "{{ item }}"
[088]     state: present
[089]   with_items:
[090]     - vim
[091]     - yum-utils
[092]     - net-tools
```

```

[093]     - lvm2
[094]
[095]   - name: Update .bashrc
[096]     become: true
[097]     blockinfile:
[098]       path: .bashrc
[099]       block: |
[100]         alias vi=vim
[101]
[102]   - name: Clean yum cache
[103]     command: yum clean all
[104]
[105]   - name: Cleanup
[106]     file:
[107]       path: "{{ item }}"
[108]       state: absent
[109]     with_items:
[110]       - /var/cache/yum/s390x/7Server
[111]
[112]   - name: Create Image Snapshot
[113]     local_action:
[114]       module: command
[115]       cmd: ./capture.py -v "{{ this_file }}" -i "{{ deployed_vm.openstack.id
[116]     register: result

```

Here are the line numbers from Example 8-10 on page 328 and their descriptions:

- ▶ [007] - [011]: Check whether the image that you are building exists.
- ▶ [013] - [016]: If the image exists, then use it or use the one from the parameter file.
- ▶ [018] - [031]: From the localhost, deploy a starting image.
- ▶ [033] - [035]: Update the host's address from the deployment data. All references to 'base' now resolve to this address.
- ▶ [037] - [040]: Erase the existing `/etc/resolv.conf` file.
- ▶ [042] - [048]: Create an empty `/etc/resolv.conf`.
- ▶ [050] - [056]: Add the name servers as specified in the parameter file.
- ▶ [058] - [059]: Clean any subscription manager configuration.
- ▶ [061] - [065]: Install the subscription manager parameter package.
- ▶ [067] - [071]: Install the subscription manager parameter package as specified in the parameter file.
- ▶ [073] - [078]: Activate the subscription.
- ▶ [080] - [083]: Update the system.
- ▶ [085] - [093]: Install some extra packages.
- ▶ [095] - [100]: Update `.bashrc` to include an alias for `vim`.
- ▶ [102] - [110]: Clean the YUM cache.
- ▶ [112] - [116]: Capture the VM as an image.

Parameters

The JSON file that is shown in Example 8-11 contains parameters that are used by the playbook to construct a base image. The line numbers are to make it easier for reference in this example only and should not be included in your parameter file.

Example 8-11 Parameters

```
[000] {
[001]   "cic_base_name" : "[base_image_name]",
[002]   "cic_url" : "https://[ip]:5000",
[003]   "cic_user" : "XXXXXXXX",
[004]   "cic_password" : "*****",
[005]   "cic_project" : "CDemo",
[006]   "cic_cacert" : "[certlocation]/[certfile].crt",
[007]   "cic_flavor" : "tiny",
[008]   "cic_rhel_image" : "rhel_image_7.7",
[009]   "cic_vlan" : "Vlan133",
[010]   "cic_key_name" : "[key_name]",
[011]   "cic_availability_zone" : "Default Group",
[012]   "cic_host" : "[cic-host-name]",
[013]   "cic_dns" : [ "[ip]" ]
[014]   "sub_org" : "Default_Oranization",
[015]   "sub_key" : "*****",
[016]   "sub_rpm" : "https://example.org/sub-manager-latest.noarch.rpm",
[017]   "this_file" : base_pok.json
[018] }
```

Here are the line numbers from Example 8-11 and their descriptions:

- ▶ [001] - `cic_base_name`: The name of the image to be produced.
- ▶ [002] - `cic_url`: The URL of the IBM CIC management node.
- ▶ [003] - `cic_user`: The user ID that is used for connecting to the IBM CIC host.
- ▶ [004] - `cic_password`: The password that is associated with `cic_user`.
- ▶ [005] - `cic_project`: The project under which the IBM CIC work is performed.
- ▶ [006] - `cic_cert`: The certificate that is used if IBM CIC uses a self-signed certificate.
- ▶ [007] - `cic_flavor`: The flavor of the deployed VM.
- ▶ [008] - `cic_rhel_image`: The image on which ours will be based.
- ▶ [009] - `cic_vlan`: The LAN to be associated with the deployed VM.
- ▶ [010] - `cic_key_name`: The name of the key to be placed in `/root/.ssh/authorized_keys`. It must be uploaded before deployment.
- ▶ [011] - `cic_availability_zone`: A way to create logical groupings of hosts.
- ▶ [012] - `cic_host`: The name of the node on which to create this VM.
- ▶ [013] - `cic_dns`: A list of DNS addresses to be placed into `/etc/resolv.conf`.
- ▶ [014] - `sub_org`: The organization to use with subscription manager registration.
- ▶ [015] - `sub_key`: The subscription manager activation key.
- ▶ [016] - `sub_rpm`: The URL of the subscription manager satellite RPM.
- ▶ [017]: The parameter file name that is used by the playbook to invoke the capture process.

Replica set virtual machine instantiation

A replica set consists of three or more VMs. They are created by using a master playbook (Example 8-12) that invokes an image and a task playbook (Example 8-13 on page 333) for each member of the replica set. Variables that are required by the `deploy-hosts.yml` playbook and `deploy-image.yml` tasks are shown in Example 8-14 on page 334.

Example 8-12 shows the `deploy-hosts.yml` playbook, which controls the provisioning process. The line numbers in this example are for reference only for this book and would not appear in the playbook.

Example 8-12 Master playbook

```
[000] ---
[001]
[002] - name: Launch a compute instance
[003]   hosts: localhost
[004]   collections:
[005]     - ibm.spectrum_virtualize
[006]   vars:
[007]     nodes: []
[008]     prep: ""
[009]
[010]   tasks:
[011]     - include_tasks: deploy-image.yml
[012]       with_items: "{{ cic_instances }}"
[013]       loop_control:
[014]         loop_var: replica_number
[015]
[016]     - name: Build Args
[017]       set_fact:
[018]         prep: "{{ prep }} -n {{ item.name }}:{{ item.vmName }}:{{ item.IP
[019] }}:{{ item.WWN }}:{{ item.volId }}:{{ item.volName }}"
[017]       with_items: "{{ nodes }}"
[019]
[020]     - name: Prepare Mongo
[021]       shell: ./prepareMongo {{ prep }} -p {{ mongodb_instance_name }}
```

Here are the line numbers from Example 8-12 and their descriptions:

- ▶ [002 - 008]: The playbook runs on the localhost and captures data in variables.
- ▶ [019 - 025]: The playbook invokes the `deploy-image.yml` tasks for each node in the replica set.
- ▶ [027 - 030]: The playbook prepares the parameters to be passed to the `prepareMongo` script.
- ▶ [032 - 033]: The playbook invokes the `prepareMongo` script, which creates supporting files for the `deploy-mongo.yml` playbook.

The `deploy-image.yml` file that is called in Example 8-13 contains the tasks that are required to provision a single VM and data volume.

Example 8-13 Image and volume tasks playbook

```
[000] ---
[001]   - name: Launch a compute instance
[002]     register: deployed_vm
[003]     openstack.cloud.server:
[004]       state: present
[005]       name: "{{ mongodb_instance_name }}-{{ replica_number }}"
[006]       image: "{{ cic_rhel_image }}"
[007]       key_name: "{{ cic_key_name }}"
[008]       availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[009]       flavor: tiny # Fix t-shirt size
[010]       security_groups: default
[011]       network: "{{ cic_vlan }}"
[012]       timeout: 1800
[013]       wait: true
[014]
[015]   - name: Create a volume
[016]     register: new_volume
[017]     openstack.cloud.volume:
[018]       state: present
[019]       name: "{{ mongodb_instance_name }}-{{ replica_number }}-data"
[020]       #availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[021]       size: 1
[022]       metadata:
[023]         "capabilities:volume_backend_name": "{{ cic_host }}"
[024]         "drivers:storage_pool": "{{ cic_storage_pool }}"
[025]
[026]   - name: Attach the volume
[027]     os_server_volume:
[028]       state: present
[029]       server: "{{ deployed_vm.openstack.name }}"
[030]       volume: "{{ new_volume.id }}"
[031]       device: /dev/vdmdv
[032]
[033]   - name: Query volume
[034]     openstack.cloud.volume_info:
[035]       name: "{{ mongodb_instance_name }}-{{ replica_number }}-data"
[036]       details: yes
[037]       register: result
[038]
[039]   - name: Extract Volume Serial Number
[040]     shell: echo "{{ result.volumes[0].id }}" | cut -b 1-13
[041]     register: serial
[042]
[043]   - name: Form volume name
[044]     set_fact:
[045]       volname: "volume-{{ mongodb_instance_name }}-{{ replica_number }}-data-{{ serial.stdout }}"
[046]
[047]   - name: Create Volume Group
```

```

[048]     shell: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port }} {{ csm_cluster}} mkvolumegroup -name {{ mongodb_instance_name }}-{{
replica_number }} | true
[049]     register: result
[050]
[051]     - name: Associate Policy with Group
[052]     command: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port }} {{ csm_cluster}} chvolumegroup -safeguardedpolicy {{ svc_policy }} {{
mongodb_instance_name }}-{{ replica_number }}
[053]     register: result
[054]
[055]     - name: Add volume to the volumegroup
[056]     command: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port}} {{ csm_cluster}} chvdisk -volumegroup {{ csm_volgroup }} {{
mongodb_instance_name }}-{{ replica_number }}
[057]     register: volgroup
[058]
[059]     - name: Add deployment information
[060]     set_fact:
[061]         nodes: "{{ nodes }}" + [ { 'name': '{{deployed_vm.openstack.name
}}', 'IP': '{{deployed_vm.openstack.public_v4}}', 'WWN':
'{{new_volume.volume.metadata.volume_wwn}}', 'volId': '{{ result.volumes[0].id
}}', 'volName': '{{ volname }}', 'vmName': '{{ deployed_vm.openstack.instance_name
}}' } ]"

```

Here are the line numbers from Example 8-13 on page 333 and their descriptions:

- ▶ [001 - 013]: Deploy a VM by using the parameters that are passed in `extra_args.json`.
- ▶ [015 - 024]: Create a data volume for Mongo.
- ▶ [026 - 031]: Attach the volume to the VM.
- ▶ [033 - 037]: Query the volume that was created in lines [015 - 024].
- ▶ [039 - 041]: Form the volume ID that the IBM SAN Volume Controller uses from the volume information.
- ▶ [043 - 045]: Define the volume name based on the volume data that you extracted.
- ▶ [047 - 059]: Send the `mkvolumegroup` command to the SAN Volume Controller and ignore any errors (`volumegroup` might already be defined).
- ▶ [051 - 053]: Send the `chvolumegroup` command to the SAN Volume Controller to associate the policy with `volumegroup`.
- ▶ [055 - 057]: Send the `chvdisk` command to the SAN Volume Controller to include it in the `volumegroup`.
- ▶ [059 - 061]: Set the facts that are used by the `prepareMongo` script.

The file that is shown in Example 8-14 provides the variables that are required by the `deploy-hosts.yml` playbook and `deploy-image.yml` tasks.

Example 8-14 Extra variable files

```

[000] {
[001]     "mongodb_instance_name" : "MONGO-RS0",
[002]     "cic_flavor" : "X-Small",
[003]     "cic_rhel_image" : "MOP_BASE_Image",
[004]     "cic_vlan" : "VLAN710-MOP",
[005]     "cic_storage_pool" : "MOPFS9110",

```

```

[006]     "cic_key_name" : "[cic_key_name]",
[007]     "cic_availability_zone" : "Default Group",
[008]     "cic_host" : "MYHOST",
[009]     "cic_instances": [1, 2, 3],
[010]     "svc_cluster" : "10.7.10.19",
[011]     "svc_user" : "[cic-user]",
[012]     "svc_policy" : "my-policy",
[013]     "svc_volgroup" : "mongo-volgroup",
[014]     "svc_port" : 22
[015] }

```

Here are the line numbers from Example 8-14 on page 334 and their descriptions:

- ▶ [001]: Name of the nodes and hostnames.
- ▶ [002]: Flavor of image that will be deployed.
- ▶ [003]: Name of the image that will be deployed.
- ▶ [004]: Name of the LAN that will be associated with the nodes.
- ▶ [005]: Storage pool for data volumes.
- ▶ [006]: Key to use to authenticate SSH connections.
- ▶ [007]: Zone that will be used for deployment.
- ▶ [008]: Host on which nodes are run.
- ▶ [009]: Instance numbers that will be used in hostnames
- ▶ [010]: IP name or address of the SAN Volume Controller.
- ▶ [011]: SAN Volume Controller username, and the public key that was specified when user was created.
- ▶ [012]: Name of the policy to apply to volumegroup.
- ▶ [013]: Name of volumegroup.
- ▶ [014]: Port to SSH for SAN Volume Controller.

OpenStack support

The file that is shown in Example 8-15 is used by the OpenStack module to authenticate against the IBM CIC host.

Example 8-15 Authenticating against the IBM CIC host

```

[000] ---
[001] clouds:
[002]   openstack:
[003]     auth:
[004]       auth_url: 'https://[ip]:5000/v3/'
[005]       username: "cdemo"
[006]       password: "XXXXXXXX"
[007]       project_name: "CDemo"
[008]       cacert: [cert_location]/[cert_name].cert
[009]       interface: public
[010]       identity_api_version: 3
[011]       domain_name: default

```

Here are the line numbers from Example 8-15 on page 335 and their descriptions:

- ▶ [004]: URL of the IBM CIC host.
- ▶ [005]: Username to authenticate.
- ▶ [006]: Password that will be used in authentication.
- ▶ [007]: Project that will be accessed.
- ▶ [008]: Certificate that is used in connection authentication.

Shadow instance deployment

If you choose to use a shadowing backup and recovery mechanism, the playbooks that are shown in Example 8-16 and Example 8-17 on page 337 may be used to create “shadows” of a specified MongoDB replica set (Example 8-16) by using the hosts file (Example 8-17 on page 337) that is created when the replica set was created.

Example 8-16 Deploying a shadow replica set

```
[000] ---
[001]
[002] - name: Launch a compute instance
[003]   hosts: localhost
[004]   collections:
[005]     - ibm.spectrum_virtualize
[006]   vars:
[007]     nodes: []
[008]     prep: ""
[009]
[010]   tasks:
[011]     - include_tasks: deploy-umbra.yml
[012]       with_items: "{{ groups['mongo_nodes'] }}"
[013]       loop_control:
[014]         loop_var: shadow
[015]
[016]     - name: Build Args
[017]       set_fact:
[018]         prep: "{{ prep }} -n {{ item.name }}:{{ item.vmName }}:{{ item.IP
[019] }}:{{ item.WWN }}:{{ item.volId }}:{{ item.volName }}"
[020]         with_items: "{{ nodes }}"
[021]
[022]     - name: Prepare Mongo
[023]       shell: ./prepareMongo {{ prep }} -p {{ mongodb_instance_name }}-S -s
[024]
[025]     - name: Append Shadow to Inventory
[026]       blockinfile:
[027]         path: "{{ mongodb_instance_name }}-S-hosts"
[028]         block: |
[029]           {% for host in groups['mongo_nodes'] %}
[030]             {{ hostvars[host] }}
[031]           {% endfor %}
```

Example 8-17 Deploying a shadow host

```

[000] ---
[001]   - name: Launch a compute instance
[002]     register: deployed_vm
[003]     openstack.cloud.server:
[004]       state: present
[005]       name: "{{ shadow }}-S"
[006]       image: "{{ cic_rhel_image }}"
[007]       key_name: "{{ cic_key_name }}"
[008]       availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[009]       flavor: tiny # Fix t-shirt size
[010]       security_groups: default
[011]       network: "{{ cic_vlan }}"
[012]       timeout: 1800
[013]       wait: true
[014]
[015]   - name: Create a volume
[016]     register: new_volume
[017]     openstack.cloud.volume:
[018]       state: present
[019]       name: "{{ shadow }}-data-S"
[020]       #availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[021]       size: 1
[022]       metadata:
[023]         "capabilities:volume_backend_name": "{{ cic_host }}"
[024]         "drivers:storage_pool": "{{ cic_storage_pool }}"
[025]
[026]   - name: Attach the volume
[027]     os_server_volume:
[028]       state: present
[029]       server: "{{ deployed_vm.openstack.name }}"
[030]       volume: "{{ new_volume.id }}"
[031]       device: /dev/vdmdv
[032]
[033]   - name: Query volume
[034]     openstack.cloud.volume_info:
[035]       name: "{{ shadow }}-data-S"
[036]       details: yes
[037]     register: result
[038]
[039]   - name: Extract Volume Serial Number
[040]     shell: echo "{{ result.volumes[0].id }}" | cut -b 1-13
[041]     register: serial
[042]
[043]   - name: Form volume name
[044]     set_fact:
[045]       volname: "volume-{{ shadow }}-data-S-{{ serial.stdout }}"
[046]
[011]   - name: Create Volume Group
[012]     shell: ssh -l {{ svc_user }} -o "StrictHostKeyChecking=no" -p {{
svc_port }} {{ svc_cluster }} mkvolume group -name {{ shadow }} || true
[013]     register: result
[014]
[015]   - name: Associate Policy with Group

```

```

[016]     command: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port }} {{ csm_cluster}} chvolumegroup -safeguardedpolicy {{ svc_policy }} {{
shadow }}
[017]     register: result
[018]
[047]     - name: Add volume to the volumegroup
[048]     command: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port}} {{ csm_cluster}} chvdisk -volumegroup {{ shadow }} {{ volname }}
[049]     register: volgroup
[050]
[051]     - name: Add deployment information
[052]     set_fact:
[053]         nodes: "{{ nodes }}" + [ { 'name': '{{deployed_vm.openstack.name
}}', 'IP': '{{deployed_vm.openstack.public_v4}}', 'WWN':
'{{new_volume.volume.metadata.volume_wwn}}', 'volId': '{{ result.volumes[0].id
}}', 'volName': '{{ volname }}', 'vmName': '{{ deployed_vm.openstack.instance_name
}}' } ]"

```

Hosts file

The file that is shown in Example 8-18 is created by the deployment playbooks and contains all the information that is needed for a successful deployment.

Example 8-18 Hosts file

```

[all:vars]

[mongo_nodes]
MONGO-RS0-1 ansible_ssh_host="[ip-1]" shadow="0" vmName="cic003c1" wwn="[wwn]"
volId="2a40014a-f233-46a7-9dfa-07370bb604fe"
volName="volume-MONGO-RS0-1-data-2a40014a-f233" ansible_user=root
ansible_python_interpreter="/usr/bin/env python2"
MONGO-RS0-2 ansible_ssh_host="[ip-2]" shadow="0" vmName="cic003c2" wwn="[wwn]"
volId="01d40a84-1409-47e3-be49-429104d8c8fa"
volName="volume-MONGO-RS0-2-data-01d40a84-1409" ansible_user=root
ansible_python_interpreter="/usr/bin/env python2"
MONGO-RS0-3 ansible_ssh_host="[ip-3]" shadow="0" vmName="cic003c3" wwn="[wwn]"
volId="146d8c19-1073-41e6-91aa-937e26eb73d2"
volName="volume-MONGO-RS0-3-data-146d8c19-1073" ansible_user=root
ansible_python_interpreter="/usr/bin/env python2"

[mongo_master]
MONGO-RS0-1 ansible_ssh_host=[ip-1] ansible_user=root
ansible_python_interpreter="/usr/bin/env python2"

```

Under `[mongo_nodes]`, add the hosts that will participate in the replica set. Under `[mongo_master]`, specify one of the nodes to become the master.

In Example 8-18 on page 338, our hosts file defines a category of hosts that is called `mongo_nodes` that consists of three hosts that are defined by the IP addresses, for example, `129.40.186.[215,218,220]`. These addresses identify servers that Ansible may manage. An IP name also can be used. The second parameter defines which user is used when contacting that host (any public keys for the Ansible player must be present on that host). The third parameter defines which Python interpreter to use. The YUM tasks that used by the deployment playbook require Python 2.7.

The `mongo_master` parameter defines one of the nodes as the master in the replica set.

8.2.4 MongoDB deployment

MongoDB deployment is performed by using a playbook (, “Controller” on page 339) that invokes another playbook for each member of the replica set. In this section, we describe each of those playbooks.

Controller

The controller playbook (shown in Example 8-19) is a sort of master playbook that is used to invoke the Tasks playbook (, “Tasks” on page 339 playbook).

Example 8-19 Controller playbook

```
[000] ---
[001] - name: install
[002]   hosts:
[003]     - mongo_nodes
[004]   roles:
[005]     - mongoddb
[006]   vars:
[007]     - nodeCount: "{{ groups['mongo_nodes'] | length }}"
[008]     - force: 0
[009]
[010] - name: Terminate Shadows
[011]   hosts:
[012]     - localhost
[013]
[014]   tasks:
[015]     - include_tasks: terminate-host.yml
[016]       with_items: "{{ groups['mongo_nodes'] }}"
[017]       loop_control:
[018]         loop_var: node
```

Here are the line numbers from Example 8-19 and their descriptions:

- ▶ [002] - [003]: Act on those hosts in the inventory file within the 'mongo_nodes' group.
- ▶ [004] - [005]: The Mongo deployment happens in the `mongoddb/tasks/main.yml` file.
- ▶ [006] - [008]: Set the node count based on the number of hosts that is defined. Do not force termination of non-shadow hosts.
- ▶ [010] - [018]: When Mongo is installed and working, terminate the shadow virtual machines.

Tasks

Tasks let you configure and save frequently run jobs so you can later run them with one click.

Example 8-20 Tasks file

```

[000] ---
[001] # tasks file for MongoDB setup
[002] #
[003] - name: Add mongo repo
[004]   yum_repository:
[005]     name: MongoDB
[006]     description: MongoDB 4.4 s390x Repository
[007]     baseurl:
https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.4/s390x/
[008]     gpgcheck: 1
[009]     enabled: 1
[010]     gpgkey: https://www.mongodb.org/static/pgp/server-4.4.asc
[011]
[012] - name: Refresh subscription manager
[013]   command: subscription-manager refresh
[014]
[015] - name: Install mongoDB and supporting programs
[016]   yum:
[017]     name:          "{{ packages }}"
[018]     state:         "present"
[019]     update_cache: yes
[020]   vars:
[021]     packages:
[022]       - 'mongodb-org'
[023]       - 'checkpolicy'
[024]       - 'policycoreutils-python'
[025]       - 'net-snmp'
[026]
[027] - name: Add node names to /etc/hosts (real)
[028]   become: true
[029]   blockinfile:
[030]     path: /etc/hosts
[031]     block: |
[032]         {% for host in groups['mongo_nodes'] %}
[033]         {{ hostvars[host].ansible_ssh_host }} {{ host }}
[034]         {% endfor %}
[035]   when: shadow == 0
[036]
[037] - name: Add node names to /etc/hosts (shadow)
[038]   become: true
[039]   blockinfile:
[040]     path: /etc/hosts
[041]     block: |
[042]         {% for host in groups['mongo_nodes'] %}
[043]         {{ hostvars[host].ansible_ssh_host }} {{ host }}
[044]         {% endfor %}
[045]         {% for host in groups['shadows'] %}
[046]         {{ hostvars[host].ansible_ssh_host }} {{ host }}
[047]         {% endfor %}
[048]   when: shadow == 1
[049]
[050] #
[051] # Check if we have a data volume and prepare it if found
[052] #

```

```
[053] - name: Get path
[054]   stat:
[055]     path: "/dev/vdmdv"
[056]   register: dev
[057]
[058] - name: partition data device
[059]   parted:
[060]     device: "/dev/vdmdv"
[061]     number: 1
[062]     state: present
[063]   when: dev.stat.exists
[064]
[065] - name: Get path to partition
[066]   find:
[067]     recurse: no
[068]     paths: "/dev/mapper"
[069]     file_type: link
[070]     follow: no
[071]     patterns: "*{{ wwn }}*1"
[072]   register: info
[073]   when: dev.stat.exists
[074]
[075] - name: Create Mongo data volume group
[076]   lvg:
[077]     vg: vg_mongo
[078]     pvs: "{{ info.files[0].path }}"
[079]   when: dev.stat.exists
[080]
[081] - name: Create LVM
[082]   lvol:
[083]     vg: vg_mongo
[084]     lv: data
[085]     size: 100%VG
[086]   when: dev.stat.exists
[087]
[088] - name: Create an xfs file system
[089]   filesystem:
[090]     fstype: xfs
[091]     dev: /dev/mapper/vg_mongo-data
[092]   when: dev.stat.exists
[093]
[094] - name: Make mount point
[095]   file:
[096]     path: /var/lib/mongo
[097]     state: directory
[098]     owner: mongod
[099]     group: mongod
[100]     mode: '0755'
[101]   when: dev.stat.exists
[102]
[103] - name: Add mount point for Mongo volume
[104]   ansible.posix.mount:
[105]     backup: yes
[106]     path: /var/lib/mongo
[107]     src: /dev/mapper/vg_mongo-data
```

```

[108]     fstype: xfs
[109]     boot: yes
[110]     dump: '1'
[111]     passno: '2'
[112]     state: present
[113]     when: dev.stat.exists
[114]
[115] - name: Mount the volume
[116]     command: mount -a
[117]     when: dev.stat.exists
[118]
[119] - name: Change ownership of data volume
[120]     file:
[121]         path: /var/lib/mongo
[122]         state: directory
[123]         owner: mongod
[124]         group: mongod
[125]         recurse: yes
[126]     when: dev.stat.exists
[127]
[128] #
[129] # SELinux processing
[130] #
[131] - name: Copy SELinux proc policy file
[132]     copy:
[133]         src: mongod_proc_net.pp
[134]         dest: /tmp/mongod_proc_net.pp
[135]         owner: root
[136]         group: root
[137]         mode: 0644
[138]
[139] - name: Copy SELinux cgroup policy file
[140]     copy:
[141]         src: mongod_cgroup_memory.pp
[142]         dest: /tmp/mongod_cgroup_memory.pp
[143]         owner: root
[144]         group: root
[145]         mode: 0644
[146]
[147] - name: Update SELinux proc file policy
[148]     command: semodule -i /tmp/mongod_proc_net.pp
[149]
[150] - name: Update SELinux cgroup memory policy
[151]     command: semodule -i /tmp/mongod_cgroup_memory.pp
[152]
[153] - name: Apply SELinux policies
[154]     community.general.sefcontext:
[155]         target: "{{ item.target }}"
[156]         setype: "{{ item.set_type }}"
[157]         state: "{{ item.state }}"
[158]     with_items:
[159]         - { state: 'present', set_type: 'mongod_var_lib_t', target:
'/var/lib/mongo.*' }
[160]         - { state: 'present', set_type: 'mongod_log_t', target:
'/var/log/mongod.*' }

```

```

[161]     - { state: 'present', set_type: 'mongod_var_run_t', target:
'/var/run/mongodb.*' }
[162]     register: filecontext
[163]     notify:
[164]     - Run restore context to reload selinux
[165]
[166] - name: Update user policy - data
[167]   command: chcon -Rv -u system_u -t mongod_var_lib_t /var/lib/mongo
[168]
[169] - name: Update user policy - logs
[170]   command: chcon -Rv -u system_u -t mongod_log_t /var/log/mongodb
[171]
[172] - name: Update user policy - run
[173]   command: chcon -Rv -u system_u -t mongod_var_run_t /var/run/mongodb
[174]
[175] #
[176] # Open required firewall ports
[177] #
[178] - name: Open firewall to Mongo port
[179]   firewallld:
[180]     service:  "{{{ item.service }}"
[181]     state:    "{{{ item.state }}"
[182]     permanent: "{{{ item.permanent }}"
[183]     immediate: "{{{ item.immediate }}"
[184]   with_items:
[185]     - { permanent: 'yes', immediate: 'yes', state: 'enabled', service:
'mongodb' }
[186]
[187] #
[188] # Configure Mongo
[189] #
[190] - name: Copy mongodb config file
[191]   copy:
[192]     src: mongod.conf
[193]     dest: /etc/mongod.conf
[194]     owner: root
[195]     group: root
[196]     mode: 0644
[197]
[198] - name: Create /etc/security/limits.d/mongodb.conf
[199]   copy:
[200]     src: security-mongodb.conf
[201]     dest: /etc/security/limits.d/mongodb.conf
[202]     owner: root
[203]     group: root
[204]     mode: 0644
[205]
[206] #
[207] # Update sysctl according to Mongo recommendations
[208] #
[209] - name: configure sysctl settings
[210]   sysctl:
[211]     name:  "{{{ item.name }}"
[212]     value: "{{{ item.value }}"
[213]     state: "{{{ item.state }}"

```

```
[214] with_items:
[215]   - { name: 'vm.dirty_ratio',          value: '15',  state:
'present' }
[216]   - { name: 'vm.dirty_background_ratio', value: '5',   state:
'present' }
[217]   - { name: 'vm.swappiness',          value: '10',  state:
'present' }
[218]   - { name: 'net.core.somaxconn',      value: '4096', state:
'present' }
[219]   - { name: 'net.ipv4.tcp_fin_timeout', value: '30',  state:
'present' }
[220]   - { name: 'net.ipv4.tcp_keepalive_intvl', value: '30',  state:
'present' }
[221]   - { name: 'net.ipv4.tcp_keepalive_time', value: '120', state:
'present' }
[222]   - { name: 'net.ipv4.tcp_max_syn_backlog', value: '4096', state:
'present' }
[223]
[224] #
[225] # Enable SNMP so we can monitor Mongo
[226] #
[227] - name: Copy SNMP server configuration
[228]   copy:
[229]     src: snmpd.conf
[230]     dest: /etc/snmp/snmpd.conf
[231]     owner: root
[232]     group: root
[233]     mode: 0644
[234]
[235] - name: Copy SNMP trap configuration
[236]   copy:
[237]     src: snmptrapd.conf
[238]     dest: /etc/snmp/snmptrapd.conf
[239]     owner: root
[240]     group: root
[241]     mode: 0644
[242]
[243] - name: Enforce SELinux
[244]   ansible.posix.selinux:
[245]     policy: targeted
[246]     state: enforcing
[247]
[248] - name: Ensure that services are enabled and running
[249]   ansible.builtin.systemd:
[250]     name:      "{{ item.name }}"
[251]     enabled:  "{{ item.enabled }}"
[252]     state:    "{{ item.state }}"
[253]   with_items:
[254]     - { name: 'snmpd',  enabled: 'yes', state: 'started' }
[255]     - { name: 'mongod', enabled: 'yes', state: 'started' }
[256]
[257] #
[258] # If we have >= 3 nodes, then we define a replica set
[259] #
[260] - name: Enable replica set operation [1] - Copy script
```



```

[261] copy:
[262]   src: rs.js
[263]   dest: /tmp
[264]   owner: root
[265]   group: root
[266]   mode: 0600
[267] when: inventory_hostname in groups['mongo_master'] and nodeCount >= "3"
[268]
[269] - name: Enable replica set operation [2] - Run shell
[270] command: mongo /tmp/rs.js
[271] when: inventory_hostname in groups['mongo_master'] and nodeCount >= "3"
[272]
[273] #
[274] # Define the admin user
[275] #
[276] - name: Add admin user to Mongo [1] - Copy script
[277] copy:
[278]   src: adminuser.js
[279]   dest: /tmp
[280]   owner: root
[281]   group: root
[282]   mode: 0600
[283] when: inventory_hostname in groups['mongo_master']
[284]
[285] - name: Add admin user to Mongo [2] - Run shell to add user
[286] command: mongo /tmp/adminuser.js
[287] when: inventory_hostname in groups['mongo_master']
[288]
[289] #
[290] # Get rid of the ephemera
[291] #
[292] - name: Cleanup
[293] file:
[294]   path: "{{ item }}"
[295]   state: absent
[296] with_items:
[297]   - /tmp/mongodb_cgroup_memory.pp
[298]   - /tmp/mongodb_proc_net.pp
[299]   - /tmp/rs.js
[300]   - /tmp/adminuser.js
[301]   - /tmp/findVol

```

Here are the line numbers from Example 8-20 on page 340 and their descriptions:

- ▶ [003 - 010]: Add the MongoDB repository so that YUM can install it.
- ▶ [012 - 021]: Install MongoDB and its supporting programs.
- ▶ [023 - 036]: Load and run the find volume process.
- ▶ [038 - 043]: Partition the data volume.
- ▶ [045 - 049]: Create pv and the volume group.
- ▶ [051 - 056]: Create a 768 MB logical volume.
- ▶ [058 - 062]: Make an XFS file system on the logical volume.
- ▶ [064 - 071]: Create a mount point for the volume.

- ▶ [073 - 083]: Add an entry in `/etc/fstab` for the volume.
- ▶ [085 - 087]: Mount the volume.
- ▶ [089 - 096]: Ensure that MongoDB owns the data volume.
- ▶ [098 - 145]: Create installation SELinux policies.
- ▶ [147 - 154]: Enable the Mongo firewall service.
- ▶ [156 - 164]: Copy MongoDB configuration files.
- ▶ [166 - 174]: Update the security limits configuration for Mongo.
- ▶ [176 - 189]: Update the `sysctl` settings.
- ▶ [191 - 205]: Install `net-snmp` and configuration files.
- ▶ [207 - 210]: Enable SELinux enforcing mode.
- ▶ [212-219]: Enable and start SNMP and MongoDB.
- ▶ [221 - 228]: On the master, copy the replica set script.
- ▶ [230 - 232]: On the master, run the replica set script.
- ▶ [234 - 241]: On the master, copy the add admin user script.
- ▶ [243 - 245]: On the master, run the add admin user script.
- ▶ [247 - 256]: Remove temporary files.

Quiescing

Use the `mongo` command to lock the database, as shown in Example 8-21.

Example 8-21 Quiescing and locking the database

```
[000] - name: Quiesce MongoDB
[001]   hosts: mongo_nodes
[002]   vars_prompt:
[003]     - name: password
[004]       prompt: Enter mongo admin password
[005]
[006]   tasks:
[007]     - name: Lock database
[008]       command: mongo --authenticationDatabase admin -u admin -p {{ password
}} --eval="try { db.fsyncLock() } catch { }"
```

Here are the line numbers from Example 8-21 and their descriptions:

- ▶ [001]: This playbook will run against all nodes because it does not know which node is primary.
- ▶ [008]: Run `db.fsyncLock()` to lock the database.

Resuming

Use the `mongo` command to unlock the database, as shown in Example 8-22.

Example 8-22 Unlocking the database and resuming

```
[000] - name: Resume MongoDB
[001]   hosts: mongo_nodes
[002]   gather_facts: no
[003]   vars_prompt:
[004]     - name: password
```

```
[005]     prompt: Enter mongo admin password
[006]
[007]   tasks:
[008]     - name: Unlock database
[009]       shell: mongo --authenticationDatabase admin -u admin -p {{ password }}
--eval="try { var rc = db.fsyncUnlock(); while (rc.lockCount > 0) { rc =
db.fsyncUnlock(); } } catch (err) { print(err); }"
```

Here are the line numbers from Example 8-22 and their descriptions:

- ▶ [001]: This playbook will run against all nodes because it does not which node is the primary.
- ▶ [009]: We run `db.fsyncUnlock` until the `lockCount` reaches 0.

8.2.5 Terminating

The termination of a replica set is performed by using a controller playbook (, “Controller” on page 347) and an embedded task.

Controller

The controller playbook that is shown in Example 8-23 includes the `terminate-host.yml` task that shuts down replica set VMs.

Example 8-23 Controller playbook

```
[000] ---
[001] - name: Shutdown replica set virtual machines
[002]   hosts:
[003]     - localhost
[004]   gather_facts: no
[005]   vars:
[006]     force: 1
[007]
[008]   tasks:
[009]     - include_tasks: terminate-host.yml
[010]       with_items: "{{ groups['mongo_nodes'] }}"
[011]       loop_control:
[012]         loop_var: node
```

Here are the line numbers from Example 8-23 on page 347 and their descriptions:

- ▶ [005] - [006]: Force the termination of the VMs.
- ▶ [008] - [012]: Invoke the terminate virtual machine task for each member of the replica set.

Embedded task

The playbook that is shown in Example 8-24 shuts down a VM if it is a shadow MongoDB server or if we force the shutdown of a *real* MongoDB server.

Example 8-24 Embedded task to shut down a virtual machine

```
[000] ---
[001] - name: Check force
[002]   set_fact:
[003]     force: 0
[004]   when: force is not defined
[005]
[006] - name: Shutdown Host
[007]   openstack.cloud.server_action:
[008]     action: stop
[009]     server: "{{ node }}"
[010]     timeout: 200
[011]   when: (hostvars[node].shadow == 1) or (force == 1)
```

Here are the line numbers from Example 8-24 and their descriptions:

- ▶ [002] - [004]: This playbook is also used when deploying shadows to shut them down if the playbook is configured to do so.
- ▶ [006] - [011]: Use the OpenStack API to terminate a VM.

8.2.6 Replica set deletion

As with replica set creation, there is a master playbook that invokes another playbook to delete the volume and image of each member of the replica set. This playbook uses the host inventory file that is created as part of the replica set creation. This section describes the playbook and tasks that are used to delete a replica set.

Master playbook

The `destroy-hosts.yml` playbook, which is shown in Example 8-25, controls the decommissioning process.

Example 8-25 Destroying a replica set

```
[000] ---
[001]
[002] - name: Destory a replica set
[003]   hosts: localhost
[004]   collections:
[005]     - ibm.spectrum_virtualize
[006]
[007]   tasks:
[008]     - include_tasks: destroy-image.yml
[009]       with_items: "{{ groups['mongo_nodes'] }}"
[010]     loop_control:
```

```
[011]     loop_var: node
```

Here are the line numbers from Example 8-25 on page 348 and their descriptions:

- ▶ [002 - 005]: The playbook runs on the localhost.
- ▶ [007 - 011]: Invoke the `destroy-image.yml` tasks for each node in the replica set. Use the data from the inventory file to guide the process.

Image and volume tasks

The `deploy-image.yml` file that is shown in Example 8-26 contains the tasks that are required to provision a single VM and data volume.

Example 8-26 The `deploy-image.yml` file

```
[000] ---
[001]     - name: Remove safeguarded policy
[002]       command: ssh -l {{ svc_user }} -o "StrictHostKeyChecking=no" -p {{
svc_port}} {{ svc_cluster}} chvolumegroup -nosafeguardedpolicy {{ node }}
[003]       register: result
[004]
[005]     - name: Remove volume from the volumegroup
[006]       command: ssh -l {{ svc_user }} -o "StrictHostKeyChecking=no" -p {{
svc_port}} {{ svc_cluster}} chvdisk -novolumegroup {{ hostvars[node].volName }}
[007]       register: result
[008]
[009]     - name: Remove volumegroup
[010]       command: ssh -l {{ svc_user }} -o "StrictHostKeyChecking=no" -p {{
svc_port}} {{ svc_cluster}} rmvolumegroup {{ node }}
[011]       register: result
[012]
[013]     - name: Destroy a compute instance
[014]       register: destroyed_vm
[015]       openstack.cloud.server:
[016]         state: absent
[017]         name: "{{ node }}"
[018]         availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[019]         timeout: 1800
[020]         wait: true
[021]
[022]     - name: Destroy the volume
[023]       register: deleted_volume
[024]       openstack.cloud.volume:
[025]         state: absent
[026]         name: "{{ hostvars[node].volId }}"
```

Here are the line numbers from Example 8-26 and their descriptions:

- ▶ [001 - 003]: Remove the safeguarded policy from the volume group.
- ▶ [005 - 007]: Remove the data volume from the volume group.
- ▶ [009 - 011]: Remove the volume group.
- ▶ [013 - 020]: Destroy the instance.
- ▶ [022 - 026]: Destroy the volume.

Deploying or destroying the variables file

The file that is shown in Example 8-27 provides the variables that are required by the `deploy-hosts.yml` playbook and `deploy-image.yml` tasks.

Example 8-27 Variables file

```
[000] {
[001]   "mongodb_instance_name" : "MONGO-RS1",
[002]   "cic_project" : "CDemo",
[003]   "cic_cacert" : "[cert_location]/[cert_name].crt",
[004]   "cic_flavor" : "tiny",
[005]   "cic_rhel_image" : "SNABASE_FBA-IMAGE",
[006]   "cic_vlan" : "VLAN710-MOP",
[007]   "cic_storage_pool" : "MOPFS9110",
[008]   "cic_key_name" : "[cic_key_name]",
[009]   "cic_availability_zone" : "Default Group",
[010]   "cic_host" : "VMHOST",
[011]   "cic_instances": [0, 1, 2],
[012]   "svc_cluster" : "",
[013]   "svc_user" : "[fs9200_user]",
[014]   "svc_policy" : "test-policy",
[015]   "svc_port" : 2222
[016] }
```

Here are the line numbers from Example 8-27 and their descriptions:

- ▶ [001]: Name of the nodes and hostnames.
- ▶ [002]: Flavor of image to be deployed.
- ▶ [003]: Name of the image to be deployed.
- ▶ [004]: Name of the LAN to be associated with the nodes.
- ▶ [005]: Storage pool for data volumes.
- ▶ [006]: Key that will be used for authenticating SSH connections.
- ▶ [007]: Zone that will be used for deployment.
- ▶ [008]: Host on which the nodes run.
- ▶ [009]: Instance numbers that will be used in the hostnames.
- ▶ [010]: IP name or address of the SAN Volume Controller.
- ▶ [011]: The SAN Volume Controller username and the public key that was specified when the user was created.
- ▶ [012]: Name of policy to apply to `volume` group.
- ▶ [013]: Name of the volume group.
- ▶ [014]: Port to SSH for SAN Volume Controller.

**B**

MongoDB as a service with IBM LinuxONE

Has continued growth for new and existing services pushed current infrastructures to their limits? Did the chip shortage slow growth potential? Is recovery from corruption easy? Is recovery from ransomware type events possible with existing systems? Can you maintain the cost of cloud services while protecting the data that is required to complete all transactions? Does it look like carbon-neutral targets can be achieved by 2025 while maintaining the demand that is created by your current server sprawl?

What if it is possible to resolve all these business issues by using today's mainframe technology?

This chapter describes how IBM LinuxONE, when combined with IBM Storage, provides superior availability, performance, and security than competing platforms by using a sample anonymized client environment. With the potential to reduce power requirements by approximately 70% while decreasing the footprint that is required to run equivalent services on x86 servers by up to 50% in a study that is associated with this type of consolidation effort, it is easy to see the start of your potential savings along with the reduced carbon footprint that this solution provides¹.

¹ The [IBM Economics Consulting and Research](#) team provided slightly better numbers for this specific consolidation effort. Refer to the link to find your savings based on your quantifiable metrics. You can save costs and gain other benefits by consolidating servers on IBM LinuxONE systems.

Figure B-1 shows the sample comparison that is used for this chapter.

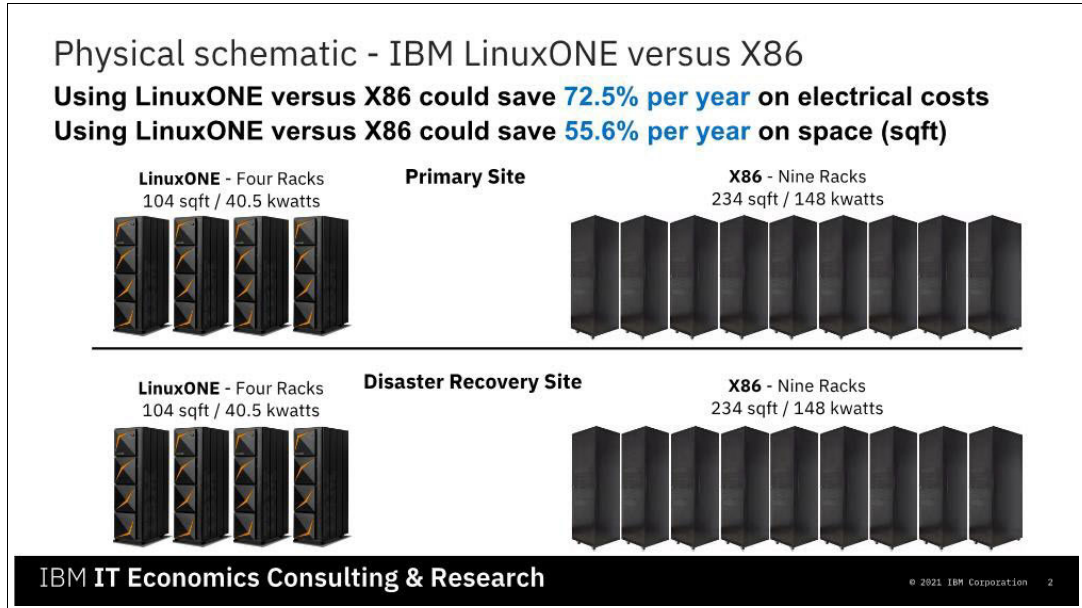


Figure B-1 IBM LinuxONE versus x86

IBM lab environment

IBM and Sine Nomine Associates (SNA) built an internal environment that mimicked a simplified version of a single-cluster client environment. This environment was hosted at the IBM Systems Client Engineering group (previously referred to as Garage for Systems). The environment is available as a demonstration outside the IBM lab. The environment was replicated on a much larger scaler at several client sites.

Figure B-2 shows a representation of the emulated data center.

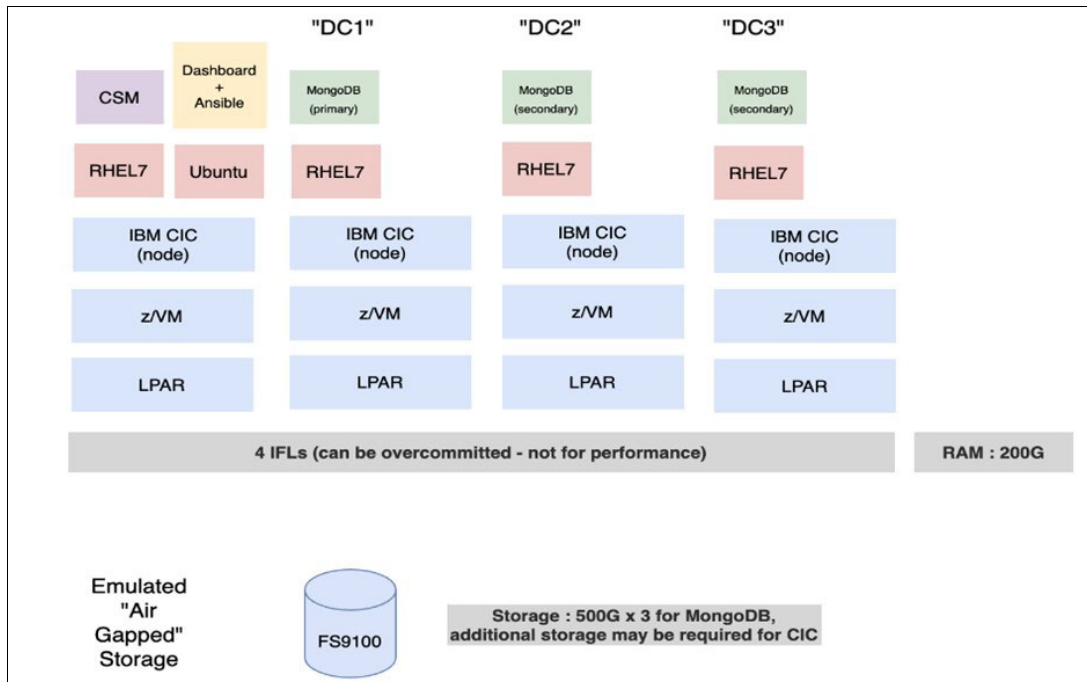


Figure B-2 Emulated data center

Figure B-3 shows how the environment was expanded to three active data centers for our use: Poughkeepsie, New York, US (POK), Montpellier France (MOP), and the Washington System Center (WSC) in Herndon, Virginia, US.

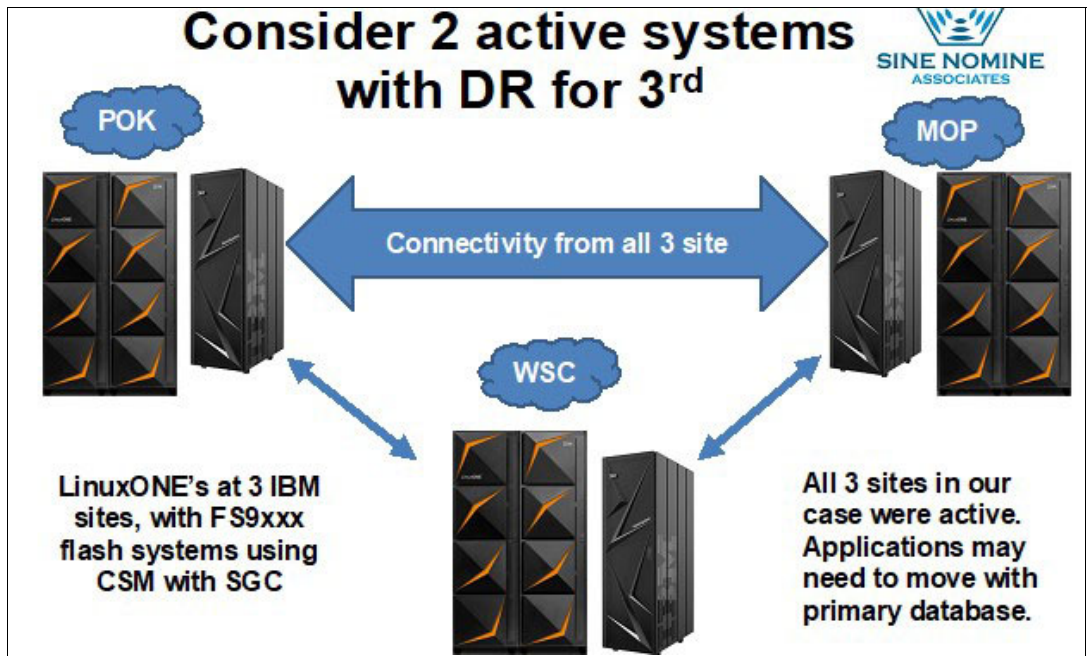


Figure B-3 Expanded lab environment

The three MongoDB database instances were deployed with IBM Cloud Infrastructure Center (IBM CIC) at each data center while keeping all the nodes in a cluster geographically dispersed. Connectivity was possible by using multiple virtual private networks (VPNs) for increased visibility and availability into the collective systems. A mix of IBM FlashSystem 9100 and IBM FlashSystem 9200 were used for storage, with all of them configured to levels that included IBM Safeguarded Copy (IBM SGC) V8.4.0 for the controllers. The IBM SGC volumes are managed and accessed with CSM for recovery, but creation was handled by automated policies on the storage devices.

IBM CIC does not contain a native mechanism for immutable snapshots because that capability is provided by the IBM FlashSystem 9x controller and Copy Services Manager.

Figure B-4 shows a sample GUI that you can use to complete the following tasks:

- ▶ Select the size of the MongoDB database that you are creating based on standard T-Shirt sizes.
- ▶ Select a checkbox to add Appendix J (App J) to the MongoDB databases that you are creating that require it.

Now, when a new MongoDB cluster is created, the App J Compliant checkbox can be selected and IBM SGC copies start based on the policy that are defined for them.

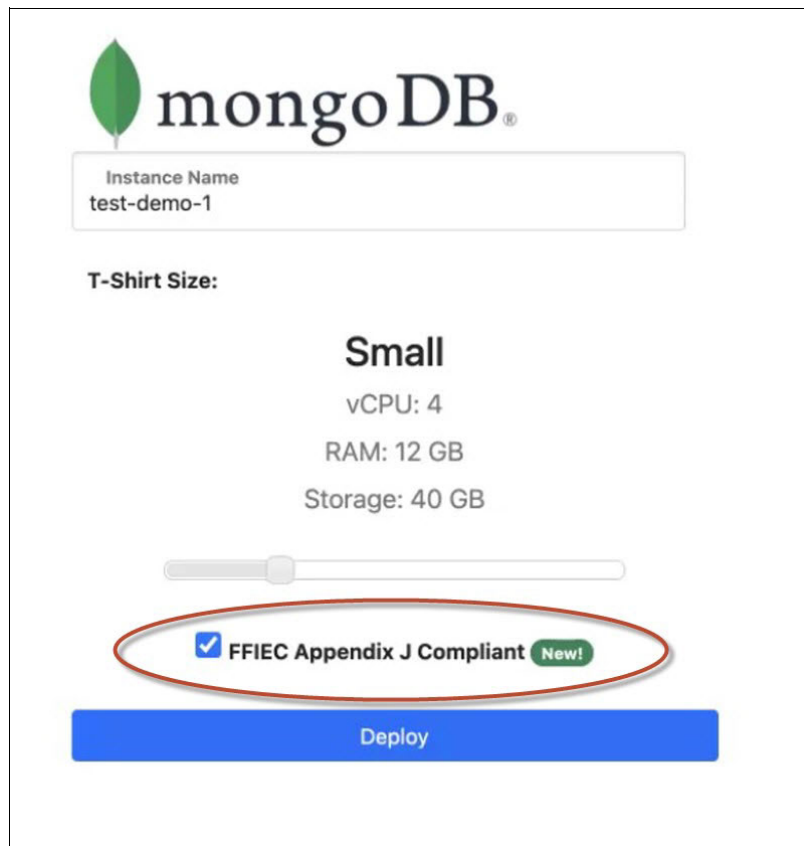


Figure B-4 Selecting the Appendix J Compliant checkbox

Federal Financial Institutions Examination Council Appendix J

The Federal Financial Institutions Examination Council (FFIEC) is a regulatory body for financial institutions in the US. As part of its regulations, App J was added as an additional appendix that sets standards for the resilience of outsourced technology. As with other

regulations, App J is open to interpretation, and the definition of being FFIEC App J compliant changes from organization to organization. Work with your organization's compliance group to determine what their technical and process checklist entails.

Appendix J, IBM Safeguarded Copy, and data serving

For our sample client, App J compliance meant building a cyberattack resilient environment. A key part of the implementation was the ability to generate immutable "air-gapped" copies of the live production data that was insulated from production access and protected from attack vectors such as malware and ransomware. IBM SGC provides a hardware-based solution that ensures the immutability of data. For our sample client use case, IBM SGC was sufficient to satisfy the air-gapped immutability aspect of the compliance document.

- ▶ Frequency and retention period

The lowest frequency for IBM SGC copies is 1 minute with a 1-day retention. However, using this frequency would deplete rapidly the storage that is needed for immutable snapshots based on write-rates, and so this frequency is impractical for real-world use cases. Most frequencies are set to a 30-min to 4-hour frequency with multi-day retention ranging from 3 days to several months (for less frequent snapshots), so a cyberattack might not be noticeable for months. As a best practice, use longer retention periods or complement on-device immutable copies with "cold" storage-immutable copies that are less performant and slower to recover, but are much more cost-effective.

In this case of MongoDB, the MongoDB live data (typically on `/var/lib/mongo` by default) required immutable air-gapping from its live production systems. In addition to the data being unmodifiable, the addition constraint of being un-erasable was also included under the definition of "immutable".

For the sample environment, we defined a 30-minute recovery time objective (RTO) and recovery point objective (RPO). By using a round-robin algorithm, each database was backed up 10 minutes apart to keep the backups within the defined range so that in the worst case, only 20 minutes of stale data would exist (within our 30-minute RPO).

- ▶ Application consistency versus crash consistency

Application consistency assumes that the application layer cleanly quiesces the application or DB before shutdown. In a real cyberattack, it is imprudent to expect or depend on application consistency. However, crash consistency treats failures as though power from the server was disconnected and caused an instantaneous or forced shutdown of the entire stack. MongoDB, with its journaling and checkpointing capabilities, offers both application consistency (with manual scripting and synchronization) and crash consistency (with a storage layer snapshot). With the default setting, the MongoDB built-in crash consistency (a loss of a few seconds at most) is sufficient to meet most RPO (for example, 30 minutes) where MongoDB is used. Zero RPO is possible, but at the expense of performance, and there are better databases that are suited for that use case.

Figure B-5 provides a quick outline of the two main types of attacks against data today, along with the recovery efforts that are needed. Although it is possible to use an IBM SGC copy of the data for normal database restoration or recovery, these efforts are part of normal backup and restore practices so that the application and Linux teams can recover from general inconsistencies that are discovered during normal operations.



Figure B-5 Main types of attacks with recovery efforts

The following automation scripts are available online from MongoDB to help with recovery:

- ▶ [Deploy Automatically with GitHub](#)
- ▶ [Backup and Restore with Filesystem Snapshots](#)

The [Configure LVM logical volumes](#) Ansible playbook was used to create the Logical Volume Management (LVM) and snapshot area.

We completed the following tasks:

1. Three-node cluster deployment automation.
2. Enablement of IBM SGC copies.
3. Validation of a “clean” (untainted) copy or snapshot.
4. Intentional corruption of live data.
5. Recovery.
6. Business continuity returned.

In the sample environment, we did not include automation to determine which copy was tainted because that task was beyond the scope of the project. It is not a hard prerequisite for App J compliance. In typical real-world scenarios, an organization has several options for this task:

- ▶ Validation before snapshot creation.
- ▶ Asynchronous validation.
- ▶ Validation on detection of a cyberattack in another workload (post-mortem).

IBM has solutions for all these options, but they are beyond the scope of this book.

Figure B-6 on page 358 shows a diagram of IBM Copy Services Manager with a 5-minute frequency and 1-day retention. 5 minutes was set up for demonstration purposes because waiting 30 minutes or multiple hours is not practical to showcase this technology.

The screenshot shows the IBM Copy Services Manager interface for a session named 'mongoDB_SGC_LBSSVC6'. The session is in a 'Normal' state, protected, and has a 'Safeguarded Copy' type. It is active on host 'H1' and is recoverable. The backup schedule is set to 'Every 5 mins', with the last recoverable backup occurring on '2021-09-13 20:06:39 MST' in the 'mongoDB_SGC' volume group. Below this, a table lists backup records with columns for Backup Time, Backup ID, Recoverability, Copy Sets, Last Result, and Expiration. All listed backups are successful and recoverable.

Backup Time	Backup ID	Recoverable	Copy Sets	Last Result	Expiration
2021-09-13 16:56:34 MST	1631577600	Yes	4	✓ IWNR2800I	2021-09-20 16:56:34 ...
2021-09-13 17:01:34 MST	1631577900	Yes	4	✓ IWNR2800I	2021-09-20 17:01:34 ...
2021-09-13 17:03:09 MST	1631577992	Yes	4	✓ IWNR2800I	2021-09-14 17:03:09 ...
2021-09-13 17:06:34 MST	1631578200	Yes	4	✓ IWNR2800I	2021-09-20 17:06:34 ...
2021-09-13 17:11:34 MST	1631578500	Yes	4	✓ IWNR2800I	2021-09-20 17:11:34 ...
2021-09-13 17:16:34 MST	1631578800	Yes	4	✓ IWNR2800I	2021-09-20 17:16:34 ...
2021-09-13 17:21:34 MST	1631579100	Yes	4	✓ IWNR2800I	2021-09-20 17:21:34 ...
2021-09-13 17:26:34 MST	1631579400	Yes	4	✓ IWNR2800I	2021-09-20 17:26:34 ...
2021-09-13 17:31:34 MST	1631579700	Yes	4	✓ IWNR2800I	2021-09-20 17:31:34 ...

Figure B-6 IBM Copy Services Manager with 5-minute frequency and 1-day retention

Deployment automation overview

The lifecycle of replica sets in MongoDB consists of several steps, which are briefly described here:

1. Create a base image that is used in all deployments (, “Base image deployment overview” on page 359).
2. Deploying a set of virtual machines (VMs) that make up a replica set (, “Replica set virtual machine instantiation overview” on page 359) or a shadow set of VMs (, “Shadow overview” on page 359).
3. Deleting a replica set or its shadows.

In addition, there are several supporting playbooks to perform tasks such as:

1. Quiescing and resuming the database so that backups may be taken.
2. Terminating the replica set gracefully.

In general, the process of enabling IBM SGC copies for a general environment is as follows:

1. Gather a list of volumes and respective storage devices.
2. Ensure that the storage devices have the Safeguarded Policy available.
3. Ensure that there is sufficient space for Safeguarded Pools.
4. Create a volume group (with Safeguarded Policy attached).
5. Attach volumes to a volume group.
6. (optional) Validate enablement in CSM (in about 2 - 3 minutes or by logging in to the storage device and inspecting the Safeguarded pool).

Base image deployment overview

This section describes the procedures for creating the base image for MongoDB deployment. Creating such an image provides a common base from which to work.

The process is controlled by a playbook with a parameter file that defines locations, credentials, and other important information.

- ▶ **Playbook:** The playbook performs the following tasks:
 - a. Updates the host's address from the deployment data.
 - b. Adds the name servers as specified in the parameter file.
 - c. Refreshes the Red Hat Enterprise Linux subscription manager data.
 - d. Updates the system and installs some extra packages.
 - e. Cleans the YUM cache.
 - f. Captures the VM as an image.
- ▶ **Parameter file:** This JSON file contains parameters that are used by the playbook to construct a base image.
- ▶ **Base inventory file:** This trivial file is used by the playbook so that you can create an IP address for the instantiated VM:
base_image

Replica set virtual machine instantiation overview

The provisioning of a replica set in MongoDB is a multistep process. The first step is provisioning the following items:

1. VMs

Ansible uses the OpenStack module to create three VMs for use as a Mongo replica set.

2. Network resources

The network parameters are used to associate a network with the provisioned VMs.

3. Data volume

Ansible uses the OpenStack module to define and attach a Mongo data volume to each of the provisioned VMs.

These tasks are performed by using Ansible playbooks. In addition, there are playbooks for creating shadow VMs, which can be used for recovery. The process is almost identical to the steps that are outlined in this section. The differences are described in , “Shadow instance deployment” on page 372.

The Ansible configuration consists of a playbook and a set of tasks that is repeated for each of the VMs that are being provisioned.

Shadow overview

When recovering a replica, here are the three options:

1. Use the existing VMs and replace the Mongo data volume.
2. Create VMs that mimic the size of the original.
3. Create shadows of the original (asynchronously during deployment).

Table B-1 provides a list of the pros and cons of each option.

Table B-1 Comparison table of the three options:

Mechanism	Pros	Cons
Reuse VMs.	<ul style="list-style-type: none"> ▶ Single set of VMs. ▶ No networking changes are required. 	<ul style="list-style-type: none"> ▶ Elegant malware is hard to detect, and a “clean state” might be impossible to determine. ▶ If the host is still compromised, the freshly recovered data might be compromised again. ▶ False recovery: Recovery and attachment might seem successful, but the malware might stay dormant and impact the data silently later (weeks or months).
Fresh VM deployment.	A clean state that is clear of malware.	<ul style="list-style-type: none"> ▶ Time-consuming (< 5 minutes) but reasonable given longer RTO windows.
Shadowing.	<ul style="list-style-type: none"> ▶ A clean state that is clear of malware. ▶ Quicker than spinning up a fresh VM because starting the instances is near instantaneous. 	<ul style="list-style-type: none"> ▶ Networking addresses or names need changing to match what the Mongo data expects.

MongoDB deployment overview

In this section, we describe several Ansible playbooks and supporting files that perform tasks that affect and effect the operation of MongoDB.

The mongo-operations playbooks were created to facilitate the deployment and operation of Mongo instances.

Required files

There are two base files that are used to define and control the Ansible environment:

1. `ansible.cfg`
2. `hosts`

The `hosts` file is created by running a script that is invoked by the playbook, which passes the IP names and addresses of the nodes.

Ansible configuration

Certain settings in Ansible are adjustable through a configuration file. By default, this file is `/etc/ansible/ansible.cfg`, and for this project, a simple file is in a local directory, which is shown in Example B-1.

Example B-1 Configuration settings in Ansible

```
[defaults]
inventory = hosts
host_key_checking = False
```

The important entry here is `inventory`, which tells Ansible where to find a definition of all the endpoints to be made known to Ansible.

Hosts

Hosts can be defined in Ansible in many different ways. In our simple implementation, we use a flat file in the `.ini` format that is created by a script that is invoked from a playbook. An example is show in , “Hosts file” on page 374. The file is named after the replica set. This file is used to create shadows, terminate the replica set, and destroy the replica set.

Deployment

Deployment consists of running a deployment playbook against the hosts that are defined in the `hosts` file.

Playbooks

The deployment playbook takes a base RHEL 7 system and performs the following tasks:

1. Install software, which includes MongoDB and supporting software.
2. Install configuration files.
3. Define an admin user for Mongo.
4. Enable operation in an SELinux enforcing environment.

Tasks

The deployment playbook (, “MongoDB deployment overview” on page 360) prepares the environment for a working MongoDB installation.

Supporting files

The following files are used to support the deployment process:

► SELinux files

Two policies must be created and installed:

- a. Enable Full Time Diagnostic Data Capture (FTDC).
- b. Allow access to `/sys/fs/cgroup`.

► Proc policy

The current SELinux policy does not allow the MongoDB process to open and read `/proc/net/netstat`, which is required for FTDC.

To create the policy that is used by the playbook, run the script that is shown in Example B-2 to create `mongodb_proc_net.te`.

Example B-2 Creating the policy

```
module mongodb_proc_net 1.0;
require {
    type proc_net_t;
    type mongod_t;
    class file { open read };
}
#===== mongod_t =====
allow mongod_t proc_net_t:file { open read };
```

Convert the policy into an SE module by running the following commands:

- **checkmodule -M -m -o mongodb_proc_net.mod mongodb_proc_net.te**
- **semodule_package -o mongodb_proc_net.pp -m mongodb_proc_net.mod**

► CGroup memory policy

The current SELinux Policy does not allow the MongoDB process to access `/sys/fs/cgroup`, which is required to determine the available memory on your system.

To create the policy that is used by the playbook, run the script that is shown in Example B-3.

Example B-3 Creating the CGroup memory policy

```
mongodb_cgroup_memory.te:
module mongodb_cgroup_memory 1.0;
require {
    type cgroup_t;
    type mongod_t;
    class dir search;
    class file { getattr open read };
}
#===== mongod_t =====
allow mongod_t cgroup_t:dir search;
allow mongod_t cgroup_t:file { getattr open read };
```

Convert the policy into an SE module by running the following commands:

- **checkmodule -M -m -o mongodb_cgroup_memory.mod mongodb_cgroup_memory.te**
- **semodule_package -o mongodb_cgroup_memory.pp -m mongodb_cgroup_memory.mod**

► JavaScript files

- rs.js

The JavaScript file that is shown in Example B-4 is generated by a script that is invoked from the playbook and used to initiate the replica set.

Example B-4 The rs.js file

```
try {
    if (rs.status().code != "") {
        rs.initiate( {
            _id: "rs0",
            members: [
                { _id: 0, host: "MONGO-RS1-1:27017" },
                { _id: 1, host: "MONGO-RS1-2:27017" },
                { _id: 2, host: "MONGO-RS1-3:27017" }
            ]
        })
    }
} catch {
}
```

– admin.js

The JavaScript file that is shown in Example B-5 adds the user admin to the admin database. The admin password is defined in this script and should be changed to meet your requirements.

Example B-5 admin.js

```
try {
  rs.secondaryOk()
  db = connect('admin')
  adminUser = db.system.users.find({user:'admin'}).count()
  if (adminUser == 0) {
    db.createUser( {
      user: "admin",
      pwd: "xxxxxxx",
      roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
    }
  )
}
} catch {
}
```

Other playbooks

Here are some other relevant playbooks:

► Quiesce

The quiesce playbook (, “Quiescing” on page 382) locks the database to prevent it from being updated. You are prompted for the admin password.

► Resume

The resume playbook (, “Resuming” on page 382) unlocks the database to allow updates to proceed. You are prompted for the admin password.

► Shutdown

The shutdown playbook (, “Terminating” on page 383) uses `systemd` to gracefully shut down a replica set.

Destroying replica sets

The decommissioning of a MongoDB replica set is a multi-step process, the first step of which is the provisioning of the following items:

- VMs
- Data volume

These tasks are performed by using Ansible playbooks.

Virtual machine deletion

Ansible uses the OpenStack module to delete the nodes that are defined in the inventory file that is specified.

Data volume

Ansible uses the OpenStack module to delete the MongoDB data volume of each provisioned VM.

Playbook and tasks

The Ansible configuration consists of a playbook and a set of tasks that is repeated for each of the VMs being provisioned. For more information, see , “Terminating” on page 383.

Playbooks

This section provides an annotated description of the playbooks that are used to create images and deploy MongoDB replica sets.

Base deployment

This playbook creates a base image for use by other deployment processes. Example B-6 shows our sample playbook. The line numbers are for this and the following examples only so that you can easily reference the explanations that follow.

Example B-6 Playbook for base deployment

```
[000] ---
[001]
[002] - name: Create a Base Image
[003]   hosts: all
[004]   gather_facts: no
[005]
[006]   tasks:
[007]   - name: Check for existing image
[008]     register: image
[009]     local_action:
[010]       module: openstack.cloud.image_info
[011]       image: "{{ cic_base_name }}-IMAGE"
[012]
[013]   - name: Set Image Data
[014]     local_action:
[015]       module: set_fact
[016]       id: "% if image.openstack_image %>{{ image.openstack_image.name }}{%
else %}{{ cic_rhel_image }}{% endif %}"
[017]
[018]   - name: Deploy a Starting Image
[019]     register: deployed_vm
[020]     local_action:
[021]       module: openstack.cloud.server
[022]       name: "{{ cic_base_name }}"
[023]       image: "{{ id }}"
[024]       key_name: "{{ cic_key_name }}"
[025]       availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[026]       flavor: "{{ cic_flavor }}"
[027]       security_groups: default
[028]       network: "{{ cic_vlan }}"
[029]       volume_size: 5
[030]       timeout: 1800
[031]       wait: true
[032]
[033]   - name: Update Inventory IP address
[034]     set_fact:
```

```
[035]         ansible_ssh_host: "{{ deployed_vm.openstack.public_v4 }}"
[036]
[037]     - name: Remove /etc/resolv.conf
[038]       file:
[039]         path: /etc/resolv.conf
[040]         state: absent
[041]
[042]     - name: Create new /etc/resolv.conf
[043]       file:
[044]         path: /etc/resolv.conf
[045]         state: touch
[046]         owner: root
[047]         group: root
[048]         mode: 0644
[049]
[050]     - name: Add content to /etc/resolv.conf
[051]       blockinfile:
[052]         path: /etc/resolv.conf
[053]         block: |
[054]             {% for dns in cic_dns %}
[055]                 nameserver {{ dns }}
[056]             {% endfor %}
[057]
[058]     - name: Clean Subscription Manager
[059]       command: subscription-manager clean
[060]
[061]     - name: Remove Old Subscription Manager
[062]       yum:
[063]         name: katello-ca-*
[064]         state: absent
[065]         update_cache: yes
[066]
[067]     - name: Add New Subscription Manager
[068]       yum:
[069]         name: "{{ sub_rpm }}"
[070]         state: present
[071]         update_cache: yes
[072]
[073]     - name: Register and auto-subscribe
[074]       community.general.redhat_subscription:
[075]         state: present
[076]         org_id: "{{ sub_org }}"
[077]         activationkey: "{{ sub_key }}"
[078]         ignore_errors: yes
[079]
[080]     - name: Upgrade System
[081]       yum:
[082]         name=*
[083]         state=latest
[084]
[085]     - name: Install Tools
[086]       yum:
[087]         name: "{{ item }}"
[088]         state: present
[089]       with_items:
```

```

[090]     - vim
[091]     - yum-utils
[092]     - net-tools
[093]     - lvm2
[094]
[095] - name: Update .bashrc
[096]   become: true
[097]   blockinfile:
[098]     path: .bashrc
[099]     block: |
[100]       alias vi=vim
[101]
[102] - name: Clean yum cache
[103]   command: yum clean all
[104]
[105] - name: Cleanup
[106]   file:
[107]     path: "{{ item }}"
[108]     state: absent
[109]   with_items:
[110]     - /var/cache/yum/s390x/7Server
[111]
[112] - name: Create Image Snapshot
[113]   local_action:
[114]     module: command
[115]     cmd: ./capture.py -v "{{ this_file }}" -i "{{ deployed_vm.openstack.id
}}}"
[116]   register: result

```

Here are the line numbers from Example B-6 on page 364 and their descriptions:

- ▶ [007] - [011]: Check whether the image that you are building exists.
- ▶ [013] - [016]: If the image exists, then use it or use the one from the parameter file.
- ▶ [018] - [031]: From the localhost, deploy a starting image.
- ▶ [033] - [035]: Update the host's address from the deployment data. All references to 'base' now resolve to this address.
- ▶ [037] - [040]: Erase the existing `/etc/resolv.conf` file.
- ▶ [042] - [048]: Create an empty `/etc/resolv.conf`.
- ▶ [050] - [056]: Add the name servers as specified in the parameter file.
- ▶ [058] - [059]: Clean any subscription manager configuration.
- ▶ [061] - [065]: Install the subscription manager parameter package.
- ▶ [067] - [071]: Install the subscription manager parameter package as specified in the parameter file.
- ▶ [073] - [078]: Activate the subscription.
- ▶ [080] - [083]: Update the system.
- ▶ [085] - [093]: Install some extra packages.
- ▶ [095] - [100]: Update `.bashrc` to include an alias for `vim`.
- ▶ [102] - [110]: Clean the YUM cache.
- ▶ [112] - [116]: Capture the VM as an image.

Parameters

The JSON file that is shown in Example B-7 contains parameters that are used by the playbook to construct a base image. The line numbers are to make it easier for reference in this example only and should not be included in your parameter file.

Example B-7 Parameters

```
[000] {
[001]   "cic_base_name" : "[base_image_name]",
[002]   "cic_url" : "https://[ip]:5000",
[003]   "cic_user" : "XXXXXXXX",
[004]   "cic_password" : "*****",
[005]   "cic_project" : "CDemo",
[006]   "cic_cacert" : "[certlocation]/[certfile].crt",
[007]   "cic_flavor" : "tiny",
[008]   "cic_rhel_image" : "rhel_image_7.7",
[009]   "cic_vlan" : "Vlan133",
[010]   "cic_key_name" : "[key_name]",
[011]   "cic_availability_zone" : "Default Group",
[012]   "cic_host" : "[cic-host-name]",
[013]   "cic_dns" : [ "[ip]" ]
[014]   "sub_org" : "Default_Organization",
[015]   "sub_key" : "*****",
[016]   "sub_rpm" : "https://example.org/sub-manager-latest.noarch.rpm",
[017]   "this_file" : base_pok.json
[018] }
```

Here are the line numbers from Example B-7 and their descriptions:

- ▶ [001] - `cic_base_name`: The name of the image to be produced.
- ▶ [002] - `cic_url`: The URL of the IBM CIC management node.
- ▶ [003] - `cic_user`: The user ID that is used for connecting to the IBM CIC host.
- ▶ [004] - `cic_password`: The password that is associated with `cic_user`.
- ▶ [005] - `cic_project`: The project under which the IBM CIC work is performed.
- ▶ [006] - `cic_cert`: The certificate that is used if IBM CIC uses a self-signed certificate.
- ▶ [007] - `cic_flavor`: The flavor of the deployed VM.
- ▶ [008] - `cic_rhel_image`: The image on which ours will be based.
- ▶ [009] - `cic_vlan`: The LAN to be associated with the deployed VM.
- ▶ [010] - `cic_key_name`: The name of the key to be placed in `/root/.ssh/authorized_keys`. It must be uploaded before deployment.
- ▶ [011] - `cic_availability_zone`: A way to create logical groupings of hosts.
- ▶ [012] - `cic_host`: The name of the node on which to create this VM.
- ▶ [013] - `cic_dns`: A list of DNS addresses to be placed into `/etc/resolv.conf`.
- ▶ [014] - `sub_org`: The organization to use with subscription manager registration.
- ▶ [015] - `sub_key`: The subscription manager activation key.
- ▶ [016] - `sub_rpm`: The URL of the subscription manager satellite RPM.
- ▶ [017]: The parameter file name that is used by the playbook to invoke the capture process.

Replica set virtual machine instantiation

A replica set consists of three or more VMs. They are created by using a master playbook (Example B-8) that invokes an image and a task playbook (Example B-9 on page 369) for each member of the replica set. Variables that are required by the `deploy-hosts.yml` playbook and `deploy-image.yml` tasks are shown in Example B-10 on page 370.

Example B-8 shows the `deploy-hosts.yml` playbook, which controls the provisioning process. The line numbers in this example are for reference only for this book and would not appear in the playbook.

Example B-8 Master playbook

```
[000] ---
[001]
[002] - name: Launch a compute instance
[003]   hosts: localhost
[004]   collections:
[005]     - ibm.spectrum_virtualize
[006]   vars:
[007]     nodes: []
[008]     prep: ""
[009]
[010]   tasks:
[011]     - include_tasks: deploy-image.yml
[012]       with_items: "{{ cic_instances }}"
[013]       loop_control:
[014]         loop_var: replica_number
[015]
[016]     - name: Build Args
[017]       set_fact:
[018]         prep: "{{ prep }} -n {{ item.name }}:{{ item.vmName }}:{{ item.IP
[019] }}:{{ item.WWN }}:{{ item.volId }}:{{ item.volName }}"
[017]       with_items: "{{ nodes }}"
[019]
[020]     - name: Prepare Mongo
[021]       shell: ./prepareMongo {{ prep }} -p {{ mongodb_instance_name }}
```

Here are the line numbers from Example B-8 and their descriptions:

- ▶ [002 - 008]: The playbook runs on the localhost and captures data in variables.
- ▶ [019 - 025]: The playbook invokes the `deploy-image.yml` tasks for each node in the replica set.
- ▶ [027 - 030]: The playbook prepares the parameters to be passed to the `prepareMongo` script.
- ▶ [032 - 033]: The playbook invokes the `prepareMongo` script, which creates supporting files for the `deploy-mongo.yml` playbook.

The `deploy-image.yml` file that is called in Example B-9 contains the tasks that are required to provision a single VM and data volume.

Example B-9 Image and volume tasks playbook

```
[000] ---
[001]   - name: Launch a compute instance
[002]     register: deployed_vm
[003]     openstack.cloud.server:
[004]       state: present
[005]       name: "{{ mongodb_instance_name }}-{{ replica_number }}"
[006]       image: "{{ cic_rhel_image }}"
[007]       key_name: "{{ cic_key_name }}"
[008]       availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[009]       flavor: tiny # Fix t-shirt size
[010]       security_groups: default
[011]       network: "{{ cic_vlan }}"
[012]       timeout: 1800
[013]       wait: true
[014]
[015]   - name: Create a volume
[016]     register: new_volume
[017]     openstack.cloud.volume:
[018]       state: present
[019]       name: "{{ mongodb_instance_name }}-{{ replica_number }}-data"
[020]       #availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[021]       size: 1
[022]       metadata:
[023]         "capabilities:volume_backend_name": "{{ cic_host }}"
[024]         "drivers:storage_pool": "{{ cic_storage_pool }}"
[025]
[026]   - name: Attach the volume
[027]     os_server_volume:
[028]       state: present
[029]       server: "{{ deployed_vm.openstack.name }}"
[030]       volume: "{{ new_volume.id }}"
[031]       device: /dev/vdmdv
[032]
[033]   - name: Query volume
[034]     openstack.cloud.volume_info:
[035]       name: "{{ mongodb_instance_name }}-{{ replica_number }}-data"
[036]       details: yes
[037]       register: result
[038]
[039]   - name: Extract Volume Serial Number
[040]     shell: echo "{{ result.volumes[0].id }}" | cut -b 1-13
[041]     register: serial
[042]
[043]   - name: Form volume name
[044]     set_fact:
[045]       volname: "volume-{{ mongodb_instance_name }}-{{ replica_number }}-data-{{ serial.stdout }}"
[046]
[047]   - name: Create Volume Group
```

```

[048]     shell: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port }} {{ csm_cluster}} mkvolumegroup -name {{ mongodb_instance_name }}-{{
replica_number }} || true
[049]     register: result
[050]
[051]     - name: Associate Policy with Group
[052]     command: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port }} {{ csm_cluster}} chvolumegroup -safeguardedpolicy {{ svc_policy }} {{
mongodb_instance_name }}-{{ replica_number }}
[053]     register: result
[054]
[055]     - name: Add volume to the volumegroup
[056]     command: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port}} {{ csm_cluster}} chvdisk -volumegroup {{ csm_volgroup }} {{
mongodb_instance_name }}-{{ replica_number }}
[057]     register: volgroup
[058]
[059]     - name: Add deployment information
[060]     set_fact:
[061]         nodes: "{{ nodes }}" + [ { 'name': '{{deployed_vm.openstack.name
}}', 'IP': '{{deployed_vm.openstack.public_v4}}', 'WWN':
'{{new_volume.volume.metadata.volume_wwn}}', 'volId': '{{ result.volumes[0].id
}}', 'volName': '{{ volname }}', 'vmName': '{{ deployed_vm.openstack.instance_name
}}' } ]"

```

Here are the line numbers from Example B-9 on page 369 and their descriptions:

- ▶ [001 - 013]: Deploy a VM by using the parameters that are passed in `extra_args.json`.
- ▶ [015 - 024]: Create a data volume for Mongo.
- ▶ [026 - 031]: Attach the volume to the VM.
- ▶ [033 - 037]: Query the volume that was created in lines [015 - 024].
- ▶ [039 - 041]: Form the volume ID that the IBM SAN Volume Controller uses from the volume information.
- ▶ [043 - 045]: Define the volume name based on the volume data that you extracted.
- ▶ [047 - 059]: Send the `mkvolumegroup` command to the SAN Volume Controller and ignore any errors (`volumegroup` might already be defined).
- ▶ [051 - 053]: Send the `chvolumegroup` command to the SAN Volume Controller to associate the policy with `volumegroup`.
- ▶ [055 - 057]: Send the `chvdisk` command to the SAN Volume Controller to include it in the `volumegroup`.
- ▶ [059 - 061]: Set the facts that are used by the `prepareMongo` script.

The file that is shown in Example B-10 provides the variables that are required by the `deploy-hosts.yml` playbook and `deploy-image.yml` tasks.

Example B-10 Extra variable files

```

[000] {
[001]     "mongodb_instance_name" : "MONGO-RS0",
[002]     "cic_flavor" : "X-Small",
[003]     "cic_rhel_image" : "MOP_BASE_Image",
[004]     "cic_vlan" : "VLAN710-MOP",
[005]     "cic_storage_pool" : "MOPFS9110",

```

```
[006]     "cic_key_name" : "[cic_key_name]",
[007]     "cic_availability_zone" : "Default Group",
[008]     "cic_host" : "MYHOST",
[009]     "cic_instances": [1, 2, 3],
[010]     "svc_cluster" : "10.7.10.19",
[011]     "svc_user" : "[cic-user]",
[012]     "svc_policy" : "my-policy",
[013]     "svc_volgroup" : "mongo-volgroup",
[014]     "svc_port" : 22
[015] }
```

Here are the line numbers from Example B-10 on page 370 and their descriptions:

- ▶ [001]: Name of the nodes and hostnames.
- ▶ [002]: Flavor of image that will be deployed.
- ▶ [003]: Name of the image that will be deployed.
- ▶ [004]: Name of the LAN that will be associated with the nodes.
- ▶ [005]: Storage pool for data volumes.
- ▶ [006]: Key to use to authenticate SSH connections.
- ▶ [007]: Zone that will be used for deployment.
- ▶ [008]: Host on which nodes are run.
- ▶ [009]: Instance numbers that will be used in hostnames
- ▶ [010]: IP name or address of the SAN Volume Controller.
- ▶ [011]: SAN Volume Controller username, and the public key that was specified when user was created.
- ▶ [012]: Name of the policy to apply to volumegroup.
- ▶ [013]: Name of volumegroup.
- ▶ [014]: Port to SSH for SAN Volume Controller.

OpenStack support

The file that is shown in Example B-11 is used by the OpenStack module to authenticate against the IBM CIC host.

Example B-11 Authenticating against the IBM CIC host

```
[000] ---
[001] clouds:
[002]   openstack:
[003]     auth:
[004]       auth_url: 'https://[ip]:5000/v3/'
[005]       username: "cdemo"
[006]       password: "XXXXXXXX"
[007]       project_name: "CDemo"
[008]       cacert: [cert_location]/[cert_name].cert
[009]       interface: public
[010]       identity_api_version: 3
[011]       domain_name: default
```

Here are the line numbers from Example B-11 on page 371 and their descriptions:

- ▶ [004]: URL of the IBM CIC host.
- ▶ [005]: Username to authenticate.
- ▶ [006]: Password that will be used in authentication.
- ▶ [007]: Project that will be accessed.
- ▶ [008]: Certificate that is used in connection authentication.

Shadow instance deployment

If you choose to use a shadowing backup and recovery mechanism, the playbooks that are shown in Example B-12 and Example 8-28 on page 373 may be used to create “shadows” of a specified MongoDB replica set (Example B-12) by using the hosts file (Example 8-28 on page 373) that is created when the replica set was created.

Example B-12 Deploying a shadow replica set

```
[000] ---
[001]
[002] - name: Launch a compute instance
[003]   hosts: localhost
[004]   collections:
[005]     - ibm.spectrum_virtualize
[006]   vars:
[007]     nodes: []
[008]     prep: ""
[009]
[010]   tasks:
[019]     - include_tasks: deploy-umbra.yml
[020]       with_items: "{{ groups['mongo_nodes'] }}"
[021]       loop_control:
[022]         loop_var: shadow
[023]
[024]     - name: Build Args
[025]       set_fact:
[026]         prep: "{{ prep }} -n {{ item.name }}:{{ item.vmName }}:{{ item.IP
[027] }}:{{ item.WWN }}:{{ item.volId }}:{{ item.volName }}"
[027]       with_items: "{{ nodes }}"
[028]
[029]     - name: Prepare Mongo
[030]       shell: ./prepareMongo {{ prep }} -p {{ mongodb_instance_name }}-S -s
[031]
[032]     - name: Append Shadow to Inventory
[033]       blockinfile:
[034]         path: "{{ mongodb_instance_name }}-S-hosts"
[035]         block: |
[036]           {% for host in groups['mongo_nodes'] %}
[037]             {{ hostvars[host] }}
[038]           {% endfor %}
```

Example 8-28 Deploying a shadow host

```

[000] ---
[001]   - name: Launch a compute instance
[002]     register: deployed_vm
[003]     openstack.cloud.server:
[004]       state: present
[005]       name: "{{ shadow }}-S"
[006]       image: "{{ cic_rhel_image }}"
[007]       key_name: "{{ cic_key_name }}"
[008]       availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[009]       flavor: tiny # Fix t-shirt size
[010]       security_groups: default
[011]       network: "{{ cic_vlan }}"
[012]       timeout: 1800
[013]       wait: true
[014]
[015]   - name: Create a volume
[016]     register: new_volume
[017]     openstack.cloud.volume:
[018]       state: present
[019]       name: "{{ shadow }}-data-S"
[020]       #availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[021]       size: 1
[022]       metadata:
[023]         "capabilities:volume_backend_name": "{{ cic_host }}"
[024]         "drivers:storage_pool": "{{ cic_storage_pool }}"
[025]
[026]   - name: Attach the volume
[027]     os_server_volume:
[028]       state: present
[029]       server: "{{ deployed_vm.openstack.name }}"
[030]       volume: "{{ new_volume.id }}"
[031]       device: /dev/vdmdv
[032]
[033]   - name: Query volume
[034]     openstack.cloud.volume_info:
[035]       name: "{{ shadow }}-data-S"
[036]       details: yes
[037]     register: result
[038]
[039]   - name: Extract Volume Serial Number
[040]     shell: echo "{{ result.volumes[0].id }}" | cut -b 1-13
[041]     register: serial
[042]
[043]   - name: Form volume name
[044]     set_fact:
[045]       volname: "volume-{{ shadow }}-data-S-{{ serial.stdout }}"
[046]
[011]   - name: Create Volume Group
[012]     shell: ssh -l {{ svc_user }} -o "StrictHostKeyChecking=no" -p {{
svc_port }} {{ svc_cluster }} mkvolume group -name {{ shadow }} || true
[013]     register: result
[014]
[015]   - name: Associate Policy with Group

```

```

[016]     command: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port }} {{ csm_cluster}} chvolumegroup -safeguardedpolicy {{ svc_policy }} {{
shadow }}
[017]     register: result
[018]
[047]     - name: Add volume to the volumegroup
[048]     command: ssh -l {{ csm_user }} -o "StrictHostKeyChecking=no" -p {{
csm_port}} {{ csm_cluster}} chvdisk -volumegroup {{ shadow }} {{ volname }}
[049]     register: volgroup
[050]
[051]     - name: Add deployment information
[052]     set_fact:
[053]         nodes: "{{ nodes }}" + [ { 'name': '{{deployed_vm.openstack.name
}}', 'IP': '{{deployed_vm.openstack.public_v4}}', 'WWN':
'{{new_volume.volume.metadata.volume_wwn}}', 'volId': '{{ result.volumes[0].id
}}', 'volName': '{{ volname }}', 'vmName': '{{ deployed_vm.openstack.instance_name
}}' } ]"

```

Hosts file

The file that is shown in Example B-13 is created by the deployment playbooks and contains all the information that is needed for a successful deployment.

Example B-13 Hosts file

```

[all:vars]

[mongo_nodes]
MONGO-RS0-1 ansible_ssh_host="[ip-1]" shadow="0" vmName="cic003c1" wwn="[wwn]"
volId="2a40014a-f233-46a7-9dfa-07370bb604fe"
volName="volume-MONGO-RS0-1-data-2a40014a-f233" ansible_user=root
ansible_python_interpreter="/usr/bin/env python2"
MONGO-RS0-2 ansible_ssh_host="[ip-2]" shadow="0" vmName="cic003c2" wwn="[wwn]"
volId="01d40a84-1409-47e3-be49-429104d8c8fa"
volName="volume-MONGO-RS0-2-data-01d40a84-1409" ansible_user=root
ansible_python_interpreter="/usr/bin/env python2"
MONGO-RS0-3 ansible_ssh_host="[ip-3]" shadow="0" vmName="cic003c3" wwn="[wwn]"
volId="146d8c19-1073-41e6-91aa-937e26eb73d2"
volName="volume-MONGO-RS0-3-data-146d8c19-1073" ansible_user=root
ansible_python_interpreter="/usr/bin/env python2"

[mongo_master]
MONGO-RS0-1 ansible_ssh_host=[ip-1] ansible_user=root
ansible_python_interpreter="/usr/bin/env python2"

```

Under `[mongo_nodes]`, add the hosts that will participate in the replica set. Under `[mongo_master]`, specify one of the nodes to become the master.

In Example B-13 on page 374, our hosts file defines a category of hosts that is called `mongo_nodes` that consists of three hosts that are defined by the IP addresses, for example, `129.40.186.[215,218,220]`. These addresses identify servers that Ansible may manage. An IP name also can be used. The second parameter defines which user is used when contacting that host (any public keys for the Ansible player must be present on that host). The third parameter defines which Python interpreter to use. The YUM tasks that used by the deployment playbook require Python 2.7.

The `mongo_master` parameter defines one of the nodes as the master in the replica set.

MongoDB deployment

MongoDB deployment is performed by using a playbook (, “Controller” on page 375) that invokes another playbook for each member of the replica set. In this section, we describe each of those playbooks.

Controller

The controller playbook (shown in Example B-14) is a sort of master playbook that is used to invoke the Tasks playbook (, “Tasks” on page 376 playbook).

Example B-14 Controller playbook

```
[000] ---
[001] - name: install
[002]   hosts:
[003]     - mongo_nodes
[004]   roles:
[005]     - mongodb
[006]   vars:
[007]     - nodeCount: "{{ groups['mongo_nodes'] | length }}"
[008]     - force: 0
[009]
[010] - name: Terminate Shadows
[011]   hosts:
[012]     - localhost
[013]
[014]   tasks:
[015]     - include_tasks: terminate-host.yml
[016]       with_items: "{{ groups['mongo_nodes'] }}"
[017]       loop_control:
[018]         loop_var: node
```

Here are the line numbers from Example B-14 and their descriptions:

- ▶ [002] - [003]: Act on those hosts in the inventory file within the 'mongo_nodes' group.
- ▶ [004] - [005]: The Mongo deployment happens in the `mongodb/tasks/main.yml` file.
- ▶ [006] - [008]: Set the node count based on the number of hosts that is defined. Do not force termination of non-shadow hosts.
- ▶ [010] - [018]: When Mongo is installed and working, terminate the shadow virtual machines.

Tasks

Tasks let you configure and save frequently run jobs so you can later run them with one click.

Example B-15 Tasks file

```
[000] ---
[001] # tasks file for MongoDB setup
[002] #
[003] - name: Add mongo repo
[004]   yum_repository:
[005]     name: MongoDB
[006]     description: MongoDB 4.4 s390x Repository
[007]     baseurl:
https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.4/s390x/
[008]     gpgcheck: 1
[009]     enabled: 1
[010]     gpgkey: https://www.mongodb.org/static/pgp/server-4.4.asc
[011]
[012] - name: Refresh subscription manager
[013]   command: subscription-manager refresh
[014]
[015] - name: Install mongoDB and supporting programs
[016]   yum:
[017]     name:          "{{ packages }}"
[018]     state:         "present"
[019]     update_cache: yes
[020]   vars:
[021]     packages:
[022]       - 'mongodb-org'
[023]       - 'checkpolicy'
[024]       - 'policycoreutils-python'
[025]       - 'net-snmp'
[026]
[027] - name: Add node names to /etc/hosts (real)
[028]   become: true
[029]   blockinfile:
[030]     path: /etc/hosts
[031]     block: |
[032]         {% for host in groups['mongo_nodes'] %}
[033]         {{ hostvars[host].ansible_ssh_host }} {{ host }}
[034]         {% endfor %}
[035]   when: shadow == 0
[036]
[037] - name: Add node names to /etc/hosts (shadow)
[038]   become: true
[039]   blockinfile:
[040]     path: /etc/hosts
[041]     block: |
[042]         {% for host in groups['mongo_nodes'] %}
[043]         {{ hostvars[host].ansible_ssh_host }} {{ host }}
[044]         {% endfor %}
[045]         {% for host in groups['shadows'] %}
[046]         {{ hostvars[host].ansible_ssh_host }} {{ host }}
[047]         {% endfor %}
[048]   when: shadow == 1
```



```
[049]
[050] #
[051] # Check if we have a data volume and prepare it if found
[052] #
[053] - name: Get path
[054]   stat:
[055]     path: "/dev/vdmdv"
[056]     register: dev
[057]
[058] - name: partition data device
[059]   parted:
[060]     device: "/dev/vdmdv"
[061]     number: 1
[062]     state: present
[063]     when: dev.stat.exists
[064]
[065] - name: Get path to partition
[066]   find:
[067]     recurse: no
[068]     paths: "/dev/mapper"
[069]     file_type: link
[070]     follow: no
[071]     patterns: "*{{ wwn }}*1"
[072]     register: info
[073]     when: dev.stat.exists
[074]
[075] - name: Create Mongo data volume group
[076]   lvg:
[077]     vg: vg_mongo
[078]     pvs: "{{ info.files[0].path }}"
[079]     when: dev.stat.exists
[080]
[081] - name: Create LVM
[082]   lvol:
[083]     vg: vg_mongo
[084]     lv: data
[085]     size: 100%VG
[086]     when: dev.stat.exists
[087]
[088] - name: Create an xfs file system
[089]   filesystem:
[090]     fstype: xfs
[091]     dev: /dev/mapper/vg_mongo-data
[092]     when: dev.stat.exists
[093]
[094] - name: Make mount point
[095]   file:
[096]     path: /var/lib/mongo
[097]     state: directory
[098]     owner: mongod
[099]     group: mongod
[100]     mode: '0755'
[101]     when: dev.stat.exists
[102]
[103] - name: Add mount point for Mongo volume
```

```
[104] ansible.posix.mount:
[105]   backup: yes
[106]   path: /var/lib/mongo
[107]   src: /dev/mapper/vg_mongo-data
[108]   fstype: xfs
[109]   boot: yes
[110]   dump: '1'
[111]   passno: '2'
[112]   state: present
[113]   when: dev.stat.exists
[114]
[115] - name: Mount the volume
[116]   command: mount -a
[117]   when: dev.stat.exists
[118]
[119] - name: Change ownership of data volume
[120]   file:
[121]     path: /var/lib/mongo
[122]     state: directory
[123]     owner: mongod
[124]     group: mongod
[125]     recurse: yes
[126]     when: dev.stat.exists
[127]
[128] #
[129] # SELinux processing
[130] #
[131] - name: Copy SELinux proc policy file
[132]   copy:
[133]     src: mongodb_proc_net.pp
[134]     dest: /tmp/mongodb_proc_net.pp
[135]     owner: root
[136]     group: root
[137]     mode: 0644
[138]
[139] - name: Copy SELinux cgroup policy file
[140]   copy:
[141]     src: mongodb_cgroup_memory.pp
[142]     dest: /tmp/mongodb_cgroup_memory.pp
[143]     owner: root
[144]     group: root
[145]     mode: 0644
[146]
[147] - name: Update SELinux proc file policy
[148]   command: semodule -i /tmp/mongodb_proc_net.pp
[149]
[150] - name: Update SELinux cgroup memory policy
[151]   command: semodule -i /tmp/mongodb_cgroup_memory.pp
[152]
[153] - name: Apply SELinux policies
[154]   community.general.sefcontext:
[155]     target: "{{ item.target }}"
[156]     setype: "{{ item.set_type }}"
[157]     state: "{{ item.state }}"
[158]   with_items:
```

```
[159] - { state: 'present', set_type: 'mongod_var_lib_t', target:
'/var/lib/mongo.*' }
[160] - { state: 'present', set_type: 'mongod_log_t', target:
'/var/log/mongodb.*' }
[161] - { state: 'present', set_type: 'mongod_var_run_t', target:
'/var/run/mongodb.*' }
[162] register: filecontext
[163] notify:
[164] - Run restore context to reload selinux
[165]
[166] - name: Update user policy - data
[167] command: chcon -Rv -u system_u -t mongod_var_lib_t /var/lib/mongo
[168]
[169] - name: Update user policy - logs
[170] command: chcon -Rv -u system_u -t mongod_log_t /var/log/mongodb
[171]
[172] - name: Update user policy - run
[173] command: chcon -Rv -u system_u -t mongod_var_run_t /var/run/mongodb
[174]
[175] #
[176] # Open required firewall ports
[177] #
[178] - name: Open firewall to Mongo port
[179] firewallld:
[180] service: "{{ item.service }}"
[181] state: "{{ item.state }}"
[182] permanent: "{{ item.permanent }}"
[183] immediate: "{{ item.immediate }}"
[184] with_items:
[185] - { permanent: 'yes', immediate: 'yes', state: 'enabled', service:
'mongodb' }
[186]
[187] #
[188] # Configure Mongo
[189] #
[190] - name: Copy mongodb config file
[191] copy:
[192] src: mongod.conf
[193] dest: /etc/mongod.conf
[194] owner: root
[195] group: root
[196] mode: 0644
[197]
[198] - name: Create /etc/security/limits.d/mongodb.conf
[199] copy:
[200] src: security-mongodb.conf
[201] dest: /etc/security/limits.d/mongodb.conf
[202] owner: root
[203] group: root
[204] mode: 0644
[205]
[206] #
[207] # Update sysctl according to Mongo recommendations
[208] #
[209] - name: configure sysctl settings
```

```
[210] sysctl:
[211]   name: "{{ item.name }}"
[212]   value: "{{ item.value }}"
[213]   state: "{{ item.state }}"
[214] with_items:
[215]   - { name: 'vm.dirty_ratio',          value: '15',  state:
'present' }
[216]   - { name: 'vm.dirty_background_ratio', value: '5',   state:
'present' }
[217]   - { name: 'vm.swappiness',          value: '10',  state:
'present' }
[218]   - { name: 'net.core.somaxconn',      value: '4096', state:
'present' }
[219]   - { name: 'net.ipv4.tcp_fin_timeout', value: '30',  state:
'present' }
[220]   - { name: 'net.ipv4.tcp_keepalive_intvl', value: '30',  state:
'present' }
[221]   - { name: 'net.ipv4.tcp_keepalive_time', value: '120', state:
'present' }
[222]   - { name: 'net.ipv4.tcp_max_syn_backlog', value: '4096', state:
'present' }
[223]
[224] #
[225] # Enable SNMP so we can monitor Mongo
[226] #
[227] - name: Copy SNMP server configuration
[228]   copy:
[229]     src: snmpd.conf
[230]     dest: /etc/snmp/snmpd.conf
[231]     owner: root
[232]     group: root
[233]     mode: 0644
[234]
[235] - name: Copy SNMP trap configuration
[236]   copy:
[237]     src: snmptrapd.conf
[238]     dest: /etc/snmp/snmptrapd.conf
[239]     owner: root
[240]     group: root
[241]     mode: 0644
[242]
[243] - name: Enforce SELinux
[244]   ansible.posix.selinux:
[245]     policy: targeted
[246]     state: enforcing
[247]
[248] - name: Ensure that services are enabled and running
[249]   ansible.builtin.systemd:
[250]     name:      "{{ item.name }}"
[251]     enabled:  "{{ item.enabled }}"
[252]     state:    "{{ item.state }}"
[253] with_items:
[254]   - { name: 'snmpd',  enabled: 'yes', state: 'started' }
[255]   - { name: 'mongod', enabled: 'yes', state: 'started' }
[256]
```

```

[257] #
[258] # If we have >= 3 nodes, then we define a replica set
[259] #
[260] - name: Enable replica set operation [1] - Copy script
[261]   copy:
[262]     src: rs.js
[263]     dest: /tmp
[264]     owner: root
[265]     group: root
[266]     mode: 0600
[267]   when: inventory_hostname in groups['mongo_master'] and nodeCount >= "3"
[268]
[269] - name: Enable replica set operation [2] - Run shell
[270]   command: mongo /tmp/rs.js
[271]   when: inventory_hostname in groups['mongo_master'] and nodeCount >= "3"
[272]
[273] #
[274] # Define the admin user
[275] #
[276] - name: Add admin user to Mongo [1] - Copy script
[277]   copy:
[278]     src: adminuser.js
[279]     dest: /tmp
[280]     owner: root
[281]     group: root
[282]     mode: 0600
[283]   when: inventory_hostname in groups['mongo_master']
[284]
[285] - name: Add admin user to Mongo [2] - Run shell to add user
[286]   command: mongo /tmp/adminuser.js
[287]   when: inventory_hostname in groups['mongo_master']
[288]
[289] #
[290] # Get rid of the ephemera
[291] #
[292] - name: Cleanup
[293]   file:
[294]     path: "{{ item }}"
[295]     state: absent
[296]   with_items:
[297]     - /tmp/mongodb_cgroup_memory.pp
[298]     - /tmp/mongodb_proc_net.pp
[299]     - /tmp/rs.js
[300]     - /tmp/adminuser.js
[301]     - /tmp/findVol

```

Here are the line numbers from Example B-15 on page 376 and their descriptions:

- ▶ [003 - 010]: Add the MongoDB repository so that YUM can install it.
- ▶ [012 - 021]: Install MongoDB and its supporting programs.
- ▶ [023 - 036]: Load and run the find volume process.
- ▶ [038 - 043]: Partition the data volume.
- ▶ [045 - 049]: Create pv and the volume group.

- ▶ [051 - 056]: Create a 768 MB logical volume.
- ▶ [058 - 062]: Make an XFS file system on the logical volume.
- ▶ [064 - 071]: Create a mount point for the volume.
- ▶ [073 - 083]: Add an entry in `/etc/fstab` for the volume.
- ▶ [085 - 087]: Mount the volume.
- ▶ [089 - 096]: Ensure that MongoDB owns the data volume.
- ▶ [098 - 145]: Create installation SELinux policies.
- ▶ [147 - 154]: Enable the Mongo firewall service.
- ▶ [156 - 164]: Copy MongoDB configuration files.
- ▶ [166 - 174]: Update the security limits configuration for Mongo.
- ▶ [176 - 189]: Update the `sysctl` settings.
- ▶ [191 - 205]: Install `net-snmp` and configuration files.
- ▶ [207 - 210]: Enable SELinux enforcing mode.
- ▶ [212-219]: Enable and start SNMP and MongoDB.
- ▶ [221 - 228]: On the master, copy the replica set script.
- ▶ [230 - 232]: On the master, run the replica set script.
- ▶ [234 - 241]: On the master, copy the add admin user script.
- ▶ [243 - 245]: On the master, run the add admin user script.
- ▶ [247 - 256]: Remove temporary files.

Quiescing

Use the `mongo` command to lock the database, as shown in Example B-16.

Example B-16 Quiescing and locking the database

```
[000] - name: Quiesce MongoDB
[001]   hosts: mongo_nodes
[002]   vars_prompt:
[003]     - name: password
[004]       prompt: Enter mongo admin password
[005]
[006]   tasks:
[007]     - name: Lock database
[008]       command: mongo --authenticationDatabase admin -u admin -p {{ password
}} --eval="try { db.fsyncLock() } catch { }"
```

Here are the line numbers from Example B-16 and their descriptions:

- ▶ [001]: This playbook will run against all nodes because it does not know which node is primary.
- ▶ [008]: Run `db.fsyncLock()` to lock the database.

Resuming

Use the `mongo` command to unlock the database, as shown in Example B-17.

Example B-17 Unlocking the database and resuming

```
[000] - name: Resume MongoDB
[001]   hosts: mongo_nodes
[002]   gather_facts: no
[003]   vars_prompt:
[004]     - name: password
[005]       prompt: Enter mongo admin password
[006]
[007]   tasks:
[008]     - name: Unlock database
[009]       shell: mongo --authenticationDatabase admin -u admin -p {{ password }}
--eval="try { var rc = db.fsyncUnlock(); while (rc.lockCount > 0) { rc =
db.fsyncUnlock(); } } catch (err) { print(err); }"
```

Here are the line numbers from Example B-17 and their descriptions:

- ▶ [001]: This playbook will run against all nodes because it does not specify which node is the primary.
- ▶ [009]: We run `db.fsyncUnlock` until the `lockCount` reaches 0.

Terminating

The termination of a replica set is performed by using a controller playbook (, “Controller” on page 383) and an embedded task.

Controller

The controller playbook that is shown in Example B-18 includes the `terminate-host.yml` task that shuts down replica set VMs.

Example B-18 Controller playbook

```
[000] ---
[001] - name: Shutdown replica set virtual machines
[002]   hosts:
[003]     - localhost
[004]   gather_facts: no
[005]   vars:
[006]     force: 1
[007]
[008]   tasks:
[009]     - include_tasks: terminate-host.yml
[010]       with_items: "{{ groups['mongo_nodes'] }}"
[011]       loop_control:
[012]         loop_var: node
```

Here are the line numbers from Example B-18 on page 383 and their descriptions:

- ▶ [005] - [006]: Force the termination of the VMs.
- ▶ [008] - [012]: Invoke the terminate virtual machine task for each member of the replica set.

Embedded task

The playbook that is shown in Example B-19 shuts down a VM if it is a shadow MongoDB server or if we force the shutdown of a *real* MongoDB server.

Example B-19 Embedded task to shut down a virtual machine

```
[000] ---
[001] - name: Check force
[002]   set_fact:
[003]     force: 0
[004]   when: force is not defined
[005]
[006] - name: Shutdown Host
[007]   openstack.cloud.server_action:
[008]     action: stop
[009]     server: "{{ node }}"
[010]     timeout: 200
[011]   when: (hostvars[node].shadow == 1) or (force == 1)
```

Here are the line numbers from Example B-19 and their descriptions:

- ▶ [002] - [004]: This playbook is also used when deploying shadows to shut them down if the playbook is configured to do so.
- ▶ [006] - [011]: Use the OpenStack API to terminate a VM.

Replica set deletion

As with replica set creation, there is a master playbook that invokes another playbook to delete the volume and image of each member of the replica set. This playbook uses the host inventory file that is created as part of the replica set creation. This section describes the playbook and tasks that are used to delete a replica set.

Master playbook

The `destroy-hosts.yml` playbook, which is shown in Example B-20, controls the decommissioning process.

Example B-20 Destroying a replica set

```
[000] ---
[001]
[002] - name: Destory a replica set
[003]   hosts: localhost
[004]   collections:
[005]     - ibm.spectrum_virtualize
[006]
[007]   tasks:
```



```
[008] - include_tasks: destroy-image.yml
[009]   with_items: "{{ groups['mongo_nodes'] }}"
[010]   loop_control:
[011]     loop_var: node
```

Here are the line numbers from Example B-20 on page 384 and their descriptions:

- ▶ [002 - 005]: The playbook runs on the localhost.
- ▶ [007 - 011]: Invoke the `destroy-image.yml` tasks for each node in the replica set. Use the data from the inventory file to guide the process.

Image and volume tasks

The `deploy-image.yml` file that is shown in Example B-21 contains the tasks that are required to provision a single VM and data volume.

Example B-21 The `deploy-image.yml` file

```
[000] ---
[001] - name: Remove safeguarded policy
[002]   command: ssh -l {{ svc_user }} -o "StrictHostKeyChecking=no" -p {{
svc_port}} {{ svc_cluster}} chvolume group -nosafeguardedpolicy {{ node }}
[003]   register: result
[004]
[005] - name: Remove volume from the volume group
[006]   command: ssh -l {{ svc_user }} -o "StrictHostKeyChecking=no" -p {{
svc_port}} {{ svc_cluster}} chdisk -novolume group {{ hostvars[node].volName }}
[007]   register: result
[008]
[009] - name: Remove volume group
[010]   command: ssh -l {{ svc_user }} -o "StrictHostKeyChecking=no" -p {{
svc_port}} {{ svc_cluster}} rmvolume group {{ node }}
[011]   register: result
[012]
[013] - name: Destroy a compute instance
[014]   register: destroyed_vm
[015]   openstack.cloud.server:
[016]     state: absent
[017]     name: "{{ node }}"
[018]     availability_zone: "{{ cic_availability_zone }}:{{ cic_host }}"
[019]     timeout: 1800
[020]     wait: true
[021]
[022] - name: Destroy the volume
[023]   register: deleted_volume
[024]   openstack.cloud.volume:
[025]     state: absent
[026]     name: "{{ hostvars[node].volId }}"
```

Here are the line numbers from Example B-21 and their descriptions:

- ▶ [001 - 003]: Remove the safeguarded policy from the volume group.
- ▶ [005 - 007]: Remove the data volume from the volume group.
- ▶ [009 - 011]: Remove the volume group.

- ▶ [013 - 020]: Destroy the instance.
- ▶ [022 - 026]: Destroy the volume.

Deploying or destroying the variables file

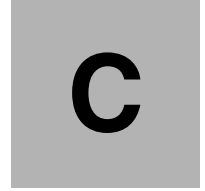
The file that is shown in Example B-22 provides the variables that are required by the `deploy-hosts.yml` playbook and `deploy-image.yml` tasks.

Example B-22 Variables file

```
[000] {
[001]   "mongodb_instance_name" : "MONGO-RS1",
[002]   "cic_project" : "CDemo",
[003]   "cic_cacert" : "[cert_location]/[cert_name].crt",
[004]   "cic_flavor" : "tiny",
[005]   "cic_rhel_image" : "SNABASE_FBA-IMAGE",
[006]   "cic_vlan" : "VLAN710-MOP",
[007]   "cic_storage_pool" : "MOPFS9110",
[008]   "cic_key_name" : "[cic_key_name]",
[009]   "cic_availability_zone" : "Default Group",
[010]   "cic_host" : "VMHOST",
[011]   "cic_instances": [0, 1, 2],
[012]   "svc_cluster" : "",
[013]   "svc_user" : "[fs9200_user]",
[014]   "svc_policy" : "test-policy",
[015]   "svc_port" : 2222
[016] }
```

Here are the line numbers from Example B-22 and their descriptions:

- ▶ [001]: Name of the nodes and hostnames.
- ▶ [002]: Flavor of image to be deployed.
- ▶ [003]: Name of the image to be deployed.
- ▶ [004]: Name of the LAN to be associated with the nodes.
- ▶ [005]: Storage pool for data volumes.
- ▶ [006]: Key that will be used for authenticating SSH connections.
- ▶ [007]: Zone that will be used for deployment.
- ▶ [008]: Host on which the nodes run.
- ▶ [009]: Instance numbers that will be used in the hostnames.
- ▶ [010]: IP name or address of the SAN Volume Controller.
- ▶ [011]: The SAN Volume Controller username and the public key that was specified when the user was created.
- ▶ [012]: Name of policy to apply to `volume` group.
- ▶ [013]: Name of the volume group.
- ▶ [014]: Port to SSH for SAN Volume Controller.



Converting SQL and PL/SQL to Fujitsu Enterprise Postgres SQL and PL/pgSQL

This appendix describes how to convert Oracle Database SQL and PL/SQL to FUJITSU Enterprise Postgres SQL and PL/pgSQL when migrating your database from Oracle Database to FUJITSU Enterprise Postgres. This appendix shows some examples of SQL and PL/SQL that are frequently used in Oracle applications. This chapter describes the difference in specifications and a concrete way to convert SQL and PL/SQL.

Challenges that are caused by the specification differences of SQL and PL/SQL

Some of the syntax and functions of SQL and PL/SQL on an Oracle Database is different from SQL and PL/pgSQL on FUJITSU Enterprise Postgres. Therefore, various problems can occur if the same SQL or PL/SQL runs in FUJITSU Enterprise Postgres. The result might be an error, or the results might be different from Oracle Database.

We provide two SQL examples that pose these challenges:

- ▶ Case of error
- ▶ Case with different execution results

Case of error

In this case, we use the SQL that is shown in Example C-1.

Example: C-1 SQL1

```
CREATE TABLE TBL(COL_1 CHAR(5));
INSERT INTO TBL VALUES('xxxxx');
INSERT INTO TBL VALUES('yyyyy');
DELETE TBL WHERE COL_1 = 'xxxxx';
```

In general, **DELETE** statements use the **FROM** clause to specify the database objects from which to delete rows. However, in SQL1, the **FROM** clause is missing from the **DELETE** statement.

When running SQL1 in an Oracle Database, one row is deleted from the table, as shown in Example C-2.

Example: C-2 Result of SQL1 in Oracle Database

1 row deleted.

When SQL1 is run in FUJITSU Enterprise Postgres, it results in an error, as shown in Example C-3. This error occurs because omitting the **FROM** clause is not allowed in FUJITSU Enterprise Postgres.

Example: C-3 Result of SQL1 in FUJITSU Enterprise Postgres

```
ERROR: syntax error (10474) at or near "TBL" (10620)
LINE 1: DELETE TBL WHERE COL_1 = 'xxxxx';
          ^
```

Use case with different runtime results

In this use case, we use the SQL that is shown in Example C-4. In SQL2, the second **INSERT** statement inserts a zero-length string (") into column COL_2 of table TBL.

Example: C-4 SQL2

```
CREATE TABLE TBL(COL_1 CHAR(5), COL_2 CHAR(5));
INSERT INTO TBL VALUES('xxxxx', 'XXXXX');
INSERT INTO TBL VALUES('yyyyy', '');
INSERT INTO TBL VALUES('zzzzz', NULL);
SELECT * FROM TBL WHERE COL_2 IS NULL;
```

Oracle treats zero-length strings as NULL. Therefore, the **SELECT** statement that extracts rows where a column (COL_2) is NULL returns rows, including the rows where a column (COL_1) is yyyyy, as shown in Example C-5.

Example: C-5 Result of SQL2 in Oracle Database

```
COL_1 COL_2
-----
yyyyy
zzzzz
```

When running SQL2 in FUJITSU Enterprise Postgres, the result does not include a row where column COL_1 is yyyyy, as shown in Example C-6 because FUJITSU Enterprise Postgres treats that zero-length string as a different value from NULL.

Example: C-6 Result of SQL2 in FUJITSU Enterprise Postgres

```
col_1 | col_2
-----+-----
zzzzz |
```

To avoid these problems after a database migration, engineers must modify the SQL and PL/SQL in applications to ensure that they receive the same results after migration as before.

Key to a successful SQL and PL/SQL conversion

Fujitsu has extensive database migration and conversion expertise in modifying SQL and PL/SQL on Oracle Database to run as SQL and PL/pgSQL on FUJITSU Enterprise Postgres. In this section, we select SQL and PL/SQL, which are frequently used in Oracle Database applications, and describe how to convert them. We classify them into several migration patterns, as shown in Table C-1 and Table C-2.

Table C-1 Migration patterns of SQL

Classification	Migration pattern
SELECT	MINUS operator.
	Hierarchical queries.
	Correlation name of subquery.
DELETE or TRUNCATE	DELETE statements.
	TRUNCATE statements for partitions.
ROWNUM pseudocolumn	ROWNUM specified in the SELECT list.
	ROWNUM specified in the WHERE clause.
Sequence	Sequence.
	Sequence pseudocolumns.
Conditions	Inequality operator.
	REGEXP_LIKE.
Function	SYSDATE.
	SYS_CONNECT_BY_PATH.
Others	Database object name.
	Implicit conversion.
	Zero-length string.
	Comparison of fixed-length character strings and variable-length character strings.

Table C-2 Migration pattern of PL/SQL

Classification	Migration pattern
DATABASE triggers	DATABASE triggers
Cursors	Cursor attribute
	Cursor variables
Error handling	Predefined exceptions
	SQLCODE
Stored functions	Stored functions
	Stored functions (performance improvement)

Classification	Migration pattern
Stored procedures	Stored procedures
Others	Cursor for FOR LOOP statements
	EXECUTE IMMEDIATE statement
	Exponentiation operator
	FORALL statement

One migration pattern contains multiple examples. The caption prefix identifies what is explained in each example. In the next sections, we demonstrate the differences between Oracle SQL and FUJITSU Enterprise Postgres SQL.

- ▶ When showing runtime examples in an Oracle Database, the caption prefixes are as follows:
 - [Oracle-SQL]: An example of SQL.
 - [Oracle-PL/SQL]: An example of PL/SQL.
 - [Oracle-Result]: A runtime result of SQL or PL/SQL. In some cases, the results that are not related to the migration pattern might not be shown in this example.
- ▶ When showing runtime examples in FUJITSU Enterprise Postgres, the caption prefixes are as follows:
 - [FUJITSU Enterprise Postgres: SQL]: An example of SQL.
 - [FUJITSU Enterprise Postgres-PL/pgSQL]: An example of PL/pgSQL.
 - [FUJITSU Enterprise Postgres-Result]: A runtime result of SQL or PL/pgSQL. In some cases, the results that are not related to the migration pattern might not be shown in this example.

Note: Some of the examples in this section use Oracle Compatible features. For more information about Oracle Compatible features, see 2.5.2, “Oracle compatible features” and 7.3 “Oracle Compatibility features” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

SQL

This section covers the following topics:

- ▶ SELECT statement
- ▶ DELETE or TRUNCATE statements
- ▶ ROWNUM pseudocolumn
- ▶ Sequence
- ▶ Conditions
- ▶ Function
- ▶ Others

SELECT statement

This section covers the following topics:

- ▶ Migration pattern: MINUS operator
- ▶ Migration pattern: Hierarchical queries
- ▶ Migration pattern: Correlation name of subquery

Migration pattern: MINUS operator

MINUS is one of a set of operators on Oracle Database. It is not supported on FUJITSU Enterprise Postgres, so replace it with the **EXCEPT** operator. Compare the commands and results that are shown in Example C-7 and Example C-8 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example C-9 and Example C-10.

Example: C-7 [Oracle-SQL] MINUS operator

```
CREATE TABLE TBL_1(COL_1 VARCHAR2(10));
CREATE TABLE TBL_2(COL_1 VARCHAR2(10));
INSERT INTO TBL_1 VALUES('AAA');
INSERT INTO TBL_1 VALUES('BBB');
INSERT INTO TBL_1 VALUES('CCC');
INSERT INTO TBL_1 VALUES('DDD');
INSERT INTO TBL_2 VALUES('BBB');
INSERT INTO TBL_2 VALUES('DDD');
SELECT COL_1 FROM TBL_1 MINUS SELECT COL_1 FROM TBL_2
ORDER BY COL_1;
```

Example: C-8 [Oracle-Result] MINUS operator

```
COL_1
-----
AAA
CCC
```

Example: C-9 FUJITSU Enterprise Postgres: SQL] MINUS operator

```
CREATE TABLE TBL_1(COL_1 VARCHAR(10));
CREATE TABLE TBL_2(COL_1 VARCHAR(10));
INSERT INTO TBL_1 VALUES('AAA');
INSERT INTO TBL_1 VALUES('BBB');
INSERT INTO TBL_1 VALUES('CCC');
INSERT INTO TBL_1 VALUES('DDD');
INSERT INTO TBL_2 VALUES('BBB');
INSERT INTO TBL_2 VALUES('DDD');
SELECT COL_1 FROM TBL_1 EXCEPT SELECT COL_1 FROM TBL_2
ORDER BY COL_1;
```

Example: C-10 [FUJITSU Enterprise Postgres-Result] MINUS operator

```
col_1
-----
AAA
CCC
```

Migration pattern: Hierarchical queries

The **START WITH** clause and **CONNECT BY** clause are used in the hierarchical query clause of Oracle Database. They are not supported on FUJITSU Enterprise Postgres, so replace them by using **WITH RECURSIVE**. Compare the commands and results that are shown in Example C-11 and Example C-12 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example C-13 and Example C-14 on page 394.

Example: C-11 [Oracle-SQL] Hierarchical queries

```
CREATE TABLE TBL(
  ID NUMBER,
  NAME VARCHAR2(10),
  PARENTID NUMBER
);
INSERT INTO TBL VALUES (1, 'A', NULL);
INSERT INTO TBL VALUES (2, 'A1', 1);
INSERT INTO TBL VALUES (3, 'A2', 1);
INSERT INTO TBL VALUES (4, 'A3', 1);
INSERT INTO TBL VALUES (5, 'A11', 2);
INSERT INTO TBL VALUES (6, 'A21', 3);
INSERT INTO TBL VALUES (7, 'A22', 3);
INSERT INTO TBL VALUES (8, 'A221', 7);
SELECT ID, NAME, PARENTID, LEVEL FROM TBL
START WITH PARENTID IS NULL
CONNECT BY PRIOR ID = PARENTID
ORDER BY LEVEL, PARENTID, ID;
```

Example: C-12 [Oracle-Result] Hierarchical queries

ID	NAME	PARENTID	LEVEL
1	A		1
2	A1	1	2
3	A2	1	2
4	A3	1	2
5	A11	2	3
6	A21	3	3
7	A22	3	3
8	A221	7	4

Example: C-13 [FUJITSU Enterprise Postgres: SQL] Hierarchical queries

```
CREATE TABLE TBL(
  ID NUMERIC,
  NAME VARCHAR(10),
  PARENTID NUMERIC
);
INSERT INTO TBL VALUES (1, 'A', NULL);
INSERT INTO TBL VALUES (2, 'A1', 1);
INSERT INTO TBL VALUES (3, 'A2', 1);
INSERT INTO TBL VALUES (4, 'A3', 1);
INSERT INTO TBL VALUES (5, 'A11', 2);
INSERT INTO TBL VALUES (6, 'A21', 3);
INSERT INTO TBL VALUES (7, 'A22', 3);
INSERT INTO TBL VALUES (8, 'A221', 7);
WITH RECURSIVE W1 AS (
  SELECT TBL.*, 1 AS LEVEL
```

```

FROM TBL WHERE PARENTID IS NULL
UNION ALL
SELECT TBL.*, W1.LEVEL + 1
FROM TBL INNER JOIN W1 ON TBL.PARENTID = W1.ID
)
SELECT ID, NAME, PARENTID, LEVEL FROM W1
ORDER BY LEVEL, PARENTID, ID;

```

Example: C-14 [FUJITSU Enterprise Postgres-Result] Hierarchical queries

id	name	parentid	level
1	A		1
2	A1	1	2
3	A2	1	2
4	A3	1	2
5	A11	2	3
6	A21	3	3
7	A22	3	3
8	A221	7	4

Migration pattern: Correlation name of subquery

An Oracle Database subquery can omit a correlation name, but FUJITSU Enterprise Postgres cannot. If a subquery in an Oracle Database omits a correlation name, add a unique correlation name when migrating to FUJITSU Enterprise Postgres. Compare the commands and results that are shown in Example C-15 and Example C-16 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example C-17 and Example C-18.

Example: C-15 [Oracle-SQL] Correlation name of subquery

```

CREATE TABLE TBL(COL_1 CHAR(5), COL_2 CHAR(5));
INSERT INTO TBL VALUES('xxxxx', 'XXXXX');
INSERT INTO TBL VALUES('yyyyy', 'YYYYY');
SELECT * FROM (SELECT * FROM TBL);

```

Example: C-16 [Oracle-Result] Correlation name of subquery

```

COL_1 COL_2
-----
xxxxx XXXXX
yyyyy YYYYY

```

Example: C-17 [FUJITSU Enterprise Postgres: SQL] Correlation name of subquery

```

CREATE TABLE TBL(COL_1 CHAR(5), COL_2 CHAR(5));
INSERT INTO TBL VALUES('xxxxx', 'XXXXX');
INSERT INTO TBL VALUES('yyyyy', 'YYYYY');
SELECT * FROM (SELECT * FROM TBL) AS foo;

```

Example: C-18 [FUJITSU Enterprise Postgres-Result] Correlation name of subquery

col_1	col_2
xxxxx	XXXXX
yyyyy	YYYYY

DELETE or TRUNCATE statements

This section describes the following topics:

- ▶ Migration pattern: DELETE statements
- ▶ Migration pattern: TRUNCATE statements for partitions

Migration pattern: DELETE statements

Oracle Database can omit the keyword **FROM** in **DELETE** statements, but FUJITSU Enterprise Postgres cannot. If a **DELETE** statement that is used in an Oracle Database omits **FROM**, add it when migrating to FUJITSU Enterprise Postgres. Compare the commands and results that are shown in Example C-19 and Example C-20 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example C-21 and Example C-22.

Example: C-19 [Oracle-SQL] DELETE statements

```
CREATE TABLE TBL(COL_1 CHAR(5));
INSERT INTO TBL VALUES('xxxxx');
INSERT INTO TBL VALUES('yyyyy');
DELETE TBL WHERE COL_1 = 'xxxxx';
SELECT * FROM TBL;
```

Example: C-20 [Oracle-Result] DELETE statements

```
COL_1
-----
yyyyy
```

Example: C-21 [FUJITSU Enterprise Postgres: SQL] DELETE statements

```
CREATE TABLE TBL(COL_1 CHAR(5));
INSERT INTO TBL VALUES('xxxxx');
INSERT INTO TBL VALUES('yyyyy');
DELETE FROM TBL WHERE COL_1 = 'xxxxx';
SELECT * FROM TBL;
```

Example: C-22 [FUJITSU Enterprise Postgres-Result] DELETE statements

```
col_1
-----
yyyyy
```

Migration pattern: TRUNCATE statements for partitions

In Oracle Database, the **ALTER TABLE TRUNCATE PARTITION** statement that runs on any partition of a partition table locks only that partition. In FUJITSU Enterprise Postgres, the **TRUNCATE** statement that runs on any partition of a partition table locks all partitions.

How you convert the SQL depends on whether the application accesses other partitions while deleting data in the target partition:

- ▶ Case 1: Do not concurrently access any partition other than the one where the data is deleted.

Replace the **ALTER TABLE TRUNCATE PARTITION** statement in the Oracle Database (Example C-23 with the result in Example C-24) with the **TRUNCATE** statement in FUJITSU Enterprise Postgres (Example C-25).

- ▶ Case 2: Concurrently access partitions other than the one where the data is deleted.

Replace the **ALTER TABLE TRUNCATE PARTITION** statement in the Oracle Database (Example C-23 with the result in Example C-24) with the **DELETE** statement in FUJITSU Enterprise Postgres (Example C-26 on page 397 with the result in Example C-27 on page 397). The **DELETE** statement does not lock partitions other than the target partition. However, when migrating to FUJITSU Enterprise Postgres, consider the following items:

- The **DELETE** statement takes longer to run than the **TRUNCATE** statement.
- The **DELETE** statement works differently than the **TRUNCATE** statement. The **DELETE** statement sets flags to indicate that data is deleted in the area. This area is not freed until the **VACUUM** statement runs. As a result, performance might degrade when retrieving or updating after data is deleted and reinserted into the target partition.

Example: C-23 [Oracle-SQL] TRUNCATE statements for a partition

```
CREATE TABLE TBL(COL_1 CHAR(2), COL_2 CHAR(5))
PARTITION BY LIST(COL_1) (
  PARTITION TBL_A1 VALUES ('A1'),
  PARTITION TBL_A2 VALUES ('A2'),
  PARTITION TBL_B1 VALUES ('B1'),
  PARTITION TBL_B2 VALUES ('B2'));
INSERT INTO TBL VALUES ('A1', '11111');
INSERT INTO TBL VALUES ('A2', '22222');
INSERT INTO TBL VALUES ('B1', '33333');
INSERT INTO TBL VALUES ('B2', '44444');
ALTER TABLE TBL TRUNCATE PARTITION TBL_A1;
SELECT * FROM TBL;
```

Example: C-24 [Oracle-Result] TRUNCATE statements for a partition

```
CO COL_2
-- -----
A2 22222
B1 33333
B2 44444
```

Example: C-25 [FUJITSU Enterprise Postgres: SQL] TRUNCATE statements for a partition (Case 1: Do not access concurrently)

```
CREATE TABLE TBL(COL_1 CHAR(2), COL_2 CHAR(5))
PARTITION BY LIST(COL_1);
CREATE TABLE TBL_A1 PARTITION OF TBL FOR VALUES IN ('A1');
CREATE TABLE TBL_A2 PARTITION OF TBL FOR VALUES IN ('A2');
CREATE TABLE TBL_B1 PARTITION OF TBL FOR VALUES IN ('B1');
CREATE TABLE TBL_B2 PARTITION OF TBL FOR VALUES IN ('B2');
INSERT INTO TBL VALUES ('A1', '11111');
INSERT INTO TBL VALUES ('A2', '22222');
INSERT INTO TBL VALUES ('B1', '33333');
```

```
INSERT INTO TBL VALUES ('B2', '44444');
TRUNCATE TBL_A1;
SELECT * FROM TBL;
```

Example: C-26 [FUJITSU Enterprise Postgres: SQL] TRUNCATE statements for a partition (Case 2: Access concurrently)

```
CREATE TABLE TBL(COL_1 CHAR(2), COL_2 CHAR(5))
PARTITION BY LIST(COL_1);
CREATE TABLE TBL_A1 PARTITION OF TBL FOR VALUES IN ('A1');
CREATE TABLE TBL_A2 PARTITION OF TBL FOR VALUES IN ('A2');
CREATE TABLE TBL_B1 PARTITION OF TBL FOR VALUES IN ('B1');
CREATE TABLE TBL_B2 PARTITION OF TBL FOR VALUES IN ('B2');
INSERT INTO TBL VALUES ('A1', '11111');
INSERT INTO TBL VALUES ('A2', '22222');
INSERT INTO TBL VALUES ('B1', '33333');
INSERT INTO TBL VALUES ('B2', '44444');
DELETE FROM TBL_A1;
SELECT * FROM TBL;
```

Example: C-27 [FUJITSU Enterprise Postgres-Result] TRUNCATE statements for a partition

col_1	col_2
A2	22222
B1	33333
B2	44444

ROWNUM pseudocolumn

This section describes the following topics:

- ▶ Migration pattern: ROWNUM specified in the select list
- ▶ Migration pattern: ROWNUM specified in the WHERE clause

Migration pattern: ROWNUM specified in the select list

The ROWNUM pseudocolumn on Oracle Database (Example C-28 with its result in Example C-29 on page 398) is not supported on FUJITSU Enterprise Postgres. If ROWNUM is specified in the **SELECT** statement list, replace it with the ROW_NUMBER function (Example C-30 on page 398 with its result in Example C-31 on page 398) when migrating to FUJITSU Enterprise Postgres.

Example: C-28 [Oracle-SQL] ROWNUM specified in the SELECT list

```
CREATE TABLE TBL(COL_1 CHAR(3));
INSERT INTO TBL VALUES('AAA');
INSERT INTO TBL VALUES('BBB');
INSERT INTO TBL VALUES('CCC');
INSERT INTO TBL VALUES('DDD');
SELECT ROWNUM, COL_1
FROM (SELECT * FROM TBL ORDER BY COL_1 DESC) WTBL;
```

Example: C-29 [Oracle-Result] ROWNUM specified in the SELECT list

```

ROWNUM COL
----- ---
      1 DDD
      2 CCC
      3 BBB
      4 AAA

```

Example: C-30 [FUJITSU Enterprise Postgres: SQL] ROWNUM specified in the SELECT list

```

CREATE TABLE TBL(COL_1 CHAR(3));
INSERT INTO TBL VALUES('AAA');
INSERT INTO TBL VALUES('BBB');
INSERT INTO TBL VALUES('CCC');
INSERT INTO TBL VALUES('DDD');
SELECT ROW_NUMBER() OVER() AS ROWNUM, COL_1
FROM (SELECT * FROM TBL ORDER BY COL_1 DESC) WTBL;

```

Example: C-31 [FUJITSU Enterprise Postgres-Result] ROWNUM specified in the SELECT list

```

rownum | col_1
-----+-----
      1 | DDD
      2 | CCC
      3 | BBB
      4 | AAA

```

Migration pattern: ROWNUM specified in the WHERE clause

The ROWNUM pseudocolumn on an Oracle Database is not supported on FUJITSU Enterprise Postgres. If ROWNUM is specified in a **WHERE** clause, then replace it with the **LIMIT** clause when migrating to FUJITSU Enterprise Postgres. Compare the commands and results that are shown in Example C-32 and Example C-33 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example C-34 on page 399 and Example C-35 on page 399.

Example: C-32 [Oracle-SQL] ROWNUM specified in the WHERE clause

```

CREATE TABLE TBL(COL_1 CHAR(3));
INSERT INTO TBL VALUES('AAA');
INSERT INTO TBL VALUES('BBB');
INSERT INTO TBL VALUES('CCC');
INSERT INTO TBL VALUES('DDD');
SELECT COL_1
FROM (SELECT * FROM TBL ORDER BY COL_1 DESC) WTBL
WHERE ROWNUM < 3;

```

Example: C-33 [Oracle-Result] ROWNUM specified in the WHERE clause

```

COL
---
DDD
CCC

```

Example: C-34 [FUJITSU Enterprise Postgres: SQL] ROWNUM specified in the WHERE clause

```
CREATE TABLE TBL(COL_1 CHAR(3));
INSERT INTO TBL VALUES('AAA');
INSERT INTO TBL VALUES('BBB');
INSERT INTO TBL VALUES('CCC');
INSERT INTO TBL VALUES('DDD');
SELECT COL_1
FROM (SELECT * FROM TBL ORDER BY COL_1 DESC) WTBL
LIMIT 2;
```

Example: C-35 [FUJITSU Enterprise Postgres-Result] ROWNUM specified in the WHERE clause

```
col_1
-----
DDD
CCC
```

Sequence

This section describes the following topics:

- ▶ Migration pattern: Sequence
- ▶ Migration pattern: Sequence pseudocolumns

Migration pattern: Sequence

FUJITSU Enterprise Postgres supports the use of the database object “sequence” as does Oracle Database, but there are some differences in the syntax or functions.

Here are the following major differences and how to convert them:

▶ **CACHE**

To specify how many sequence numbers are to be pre-allocated and stored in memory, FUJITSU Enterprise Postgres supports the **CACHE** option, as does Oracle Database, but the function is different than Oracle Database. Oracle Database caches sequence numbers on a per instance basis, but FUJITSU Enterprise Postgres caches them on a per session basis.

The default value when the **CACHE** option is omitted is different. Oracle Database sets the cache size to 20, while FUJITSU Enterprise Postgres sets it to 1.

Because of these differences, when migrating to FUJITSU Enterprise Postgres, consider the intended use and performance impact of sequence numbers. For example, if performance when getting the sequence number is the highest priority and it is not important to skip the number on a per session basis, set the cache size as you would with Oracle Database. If the highest priority is to avoid skipping the number, set the cache size to 1.

▶ **NOCACHE**

To indicate that values of the sequence are not pre-allocated, Oracle Database supports the **NOCACHE** option, but FUJITSU Enterprise Postgres does not. Remove the **NOCACHE** keyword when migrating. If the **CACHE** option is not specified, FUJITSU Enterprise Postgres sets the cache size to 1, so the state is the same as when Oracle Database specifies the **NOCACHE** option.

► **NOMAXVALUE, NOMINVALUE, and NOCYCLE**

The **NOMAXVALUE**, **NOMINVALUE**, and **NOCYCLE** options are not supported on FUJITSU Enterprise Postgres. Replace them with the **NO MAXVALUE**, **NO MINVALUE**, and **NO CYCLE** options for the equivalent functions.

Example C-36 shows a sequence definition when using an Oracle Database with the results shown in Example C-37. Example C-38 shows a sequence definition when using FUJITSU Enterprise Postgres with the results shown in Example C-39.

Example: C-36 [Oracle-SQL] Sequence definition

```
CREATE SEQUENCE SEQ_1
  START WITH 1
  INCREMENT BY 1
  CACHE 1000;
CREATE SEQUENCE SEQ_2
  START WITH 1
  INCREMENT BY 1
  NOCACHE
  NOMAXVALUE
  NOMINVALUE
  NOCYCLE;
```

Example: C-37 [Oracle-Result] Sequence definition

```
Sequence created.
Sequence created.
```

Example: C-38 [FUJITSU Enterprise Postgres: SQL] Sequence definition

```
CREATE SEQUENCE SEQ_1
  START WITH 1
  INCREMENT BY 1
  CACHE 1;
CREATE SEQUENCE SEQ_2
  START WITH 1
  INCREMENT BY 1

  NO MAXVALUE
  NO MINVALUE
  NO CYCLE;
```

Example: C-39 [FUJITSU Enterprise Postgres-Result] Sequence definition

```
CREATE SEQUENCE
CREATE SEQUENCE
```

Migration pattern: Sequence pseudocolumns

Oracle Database supports sequence pseudocolumns such as **CURRVAL** and **NEXTVAL**, but FUJITSU Enterprise Postgres does not support them. Replace them with sequence functions such as **CURRVAL** and **NEXTVAL**.

Example C-40 on page 401 and Example C-41 on page 401 show the use of sequence pseudocolumns in the Oracle Database, and Example C-42 on page 401 and Example C-43 on page 401 show the use of sequence pseudocolumns in FUJITSU Enterprise Postgres.

Example: C-40 [Oracle-SQL] Sequence pseudocolumns

```
CREATE TABLE TBL(COL_1 NUMBER, COL_2 CHAR(5));
CREATE SEQUENCE SEQ_1
  START WITH 1
  INCREMENT BY 1
  CACHE 100;
INSERT INTO TBL VALUES(SEQ_1.NEXTVAL, 'AAAAA');
INSERT INTO TBL VALUES(SEQ_1.CURRVAL, 'aaaaa');
INSERT INTO TBL VALUES(SEQ_1.NEXTVAL, 'BBBBB');
INSERT INTO TBL VALUES(SEQ_1.CURRVAL, 'bbbbb');
SELECT * FROM TBL;
```

Example: C-41 [Oracle-Result] Sequence pseudocolumns

COL_1	COL_2
1	AAAAA
1	aaaaa
2	BBBBB
2	bbbbb

Example: C-42 [FUJITSU Enterprise Postgres: SQL] Sequence pseudocolumns

```
CREATE TABLE TBL(COL_1 NUMERIC, COL_2 CHAR(5));
CREATE SEQUENCE SEQ_1
  START WITH 1
  INCREMENT BY 1
  CACHE 1;
INSERT INTO TBL VALUES(NEXTVAL('SEQ_1'), 'AAAAA');
INSERT INTO TBL VALUES(CURRVAL('SEQ_1'), 'aaaaa');
INSERT INTO TBL VALUES(NEXTVAL('SEQ_1'), 'BBBBB');
INSERT INTO TBL VALUES(CURRVAL('SEQ_1'), 'bbbbb');
SELECT * FROM TBL;
```

Example: C-43 [FUJITSU Enterprise Postgres-Result] Sequence pseudocolumns

col_1	col_2
1	AAAAA
1	aaaaa
2	BBBBB
2	bbbbb

Conditions

This section describes the following topics:

- ▶ Migration pattern: Inequality operator
- ▶ Migration pattern: REGEXP_LIKE

Migration pattern: Inequality operator

To test inequality, Oracle Database supports three types of operators: "!=", "^=", and "<>". FUJITSU Enterprise Postgres supports only two types of operators: "!=" and "<>". Therefore, when migrating, if "^=" is specified, replace it with "!=" or "<>".

Compare the use of inequality operators as used in Oracle Database (Example C-44 and Example C-45) with their use in FUJITSU Enterprise Postgres (Example C-46 and Example C-47).

Example: C-44 [Oracle-SQL] Inequality operator

```
CREATE TABLE TBL(COL_1 CHAR(5), COL_2 CHAR(5));
INSERT INTO TBL VALUES('AAAAA', 'aaaaa');
INSERT INTO TBL VALUES('BBBBB', 'bbbbbb');
INSERT INTO TBL VALUES('CCCCC', 'ccccc');
SELECT * FROM TBL WHERE COL_2 ^= 'bbbbbb'
ORDER BY COL_1;
```

Example: C-45 [Oracle-Result] Inequality operator

```
COL_1 COL_2
-----
AAAAA aaaaa
CCCCC ccccc
```

Example: C-46 [FUJITSU Enterprise Postgres: SQL] Inequality operator

```
CREATE TABLE TBL(COL_1 CHAR(5), COL_2 CHAR(5));
INSERT INTO TBL VALUES('AAAAA', 'aaaaa');
INSERT INTO TBL VALUES('BBBBB', 'bbbbbb');
INSERT INTO TBL VALUES('CCCCC', 'ccccc');
SELECT * FROM TBL WHERE COL_2 != 'bbbbbb'
ORDER BY COL_1;
```

Example: C-47 [FUJITSU Enterprise Postgres-Result] Inequality operator

```
col_1 | col_2
-----+-----
AAAAA | aaaaa
CCCCC | ccccc
```

Migration pattern: REGEXP_LIKE

Oracle Database supports the **REGEXP_LIKE** condition as one of the pattern-matching conditions to compare character data. However, FUJITSU Enterprise Postgres does not support it, so replace it with the "~" operator.

Compare the use of **REGEXP_LIKE** by Oracle Database in Example C-48 and Example C-49 on page 403 to the use of **REGEXP_LIKE** in FUJITSU Enterprise Postgres in Example C-50 on page 403 and Example C-51 on page 403.

Example: C-48 [Oracle-SQL] REGEXP_LIKE

```
CREATE TABLE TBL(COL_1 NUMBER, COL_2 VARCHAR2(10));
INSERT INTO TBL VALUES(1, 'ABCDE');
INSERT INTO TBL VALUES(2, 'Abcde');
INSERT INTO TBL VALUES(3, 'abcde');
INSERT INTO TBL VALUES(4, 'abcdefg');
SELECT * FROM TBL
WHERE REGEXP_LIKE(COL_2, '^A|a|bcde$')
ORDER BY COL_1;
```

Example: C-49 [Oracle-Result] REGEXP_LIKE

COL_1	COL_2
2	Abcde
3	abcde

Example: C-50 [FUJITSU Enterprise Postgres: SQL] REGEXP_LIKE

```
CREATE TABLE TBL(COL_1 NUMERIC, COL_2 VARCHAR(10));
INSERT INTO TBL VALUES(1, 'ABCDE');
INSERT INTO TBL VALUES(2, 'Abcde');
INSERT INTO TBL VALUES(3, 'abcde');
INSERT INTO TBL VALUES(4, 'abcdefg');
SELECT * FROM TBL
WHERE COL_2 ~ '^(A|a)bcde$'
ORDER BY COL_1;
```

Example: C-51 [FUJITSU Enterprise Postgres-Result] REGEXP_LIKE

col_1	col_2
2	Abcde
3	abcde

Function

This section describes the following topics:

- ▶ Migration pattern: SYSDATE
- ▶ Migration pattern: SYS_CONNECT_BY_PATH

Migration pattern: SYSDATE

The **SYSDATE** function that is used on Oracle Database is not supported on FUJITSU Enterprise Postgres, so replace it with the **STATEMENT_TIMESTAMP** function when migrating.

The data type of the **STATEMENT_TIMESTAMP** result is different from the **SYSDATE** result. The runtime result of **STATEMENT_TIMESTAMP** is a **TIMESTAMP WITH TIME ZONE** data type. So, you must cast the result to the appropriate data type, such as the **DATE** data type, depending on the requirements of the application.

Compare the use of the function and results in Oracle Database (Example C-52 and Example C-53 on page 404) with FUJITSU Enterprise Postgres (Example C-54 on page 404 and Example C-55 on page 404).

Example: C-52 [Oracle-SQL] SYSDATE

```
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD';
CREATE TABLE TBL(COL_1 DATE, COL_2 CHAR(5));
INSERT INTO TBL VALUES(SYSDATE, 'XXXXX');
SELECT * FROM TBL;
```

Example: C-53 [Oracle-Result] SYSDATE

```
COL_1      COL_2
-----  -----
2021-10-12 XXXXX
```

Example: C-54 [FUJITSU Enterprise Postgres: SQL] SYSDATE

```
CREATE TABLE TBL(COL_1 DATE, COL_2 CHAR(5));
INSERT INTO TBL VALUES(CAST(STATEMENT_TIMESTAMP() AS DATE), 'XXXXX');
SELECT * FROM TBL;
```

Example: C-55 [FUJITSU Enterprise Postgres-Result] SYSDATE

```
col_1      | col_2
-----+-----
2021-10-12 | XXXXX
```

Migration pattern: SYS_CONNECT_BY_PATH

SYS_CONNECT_BY_PATH is supported on Oracle Database to retrieve the path of column values from root to node in a hierarchical query. However, it is not supported on FUJITSU Enterprise Postgres. Therefore, when migrating a hierarchical query from Oracle Database to FUJITSU Enterprise Postgres, the following instructions achieve the equivalent functions of **SYS_CONNECT_BY_PATH**.

Replace **START WITH** and **CONNECT BY** clauses in Oracle Database hierarchical queries with the clause **WITH RECURSIVE** when migrating to FUJITSU Enterprise Postgres. Use the string concatenation operator (||) and add the following two processing methods in the recursive query:

1. Add processing to the root row in a recursive query.

The root row of a recursive query is specified in the **SELECT** list before **UNION ALL** in the **SELECT** statement. Add to this **SELECT** list the equivalent value of the root row of the **SYS_CONNECT_BY_PATH** function. The value to add is the string concatenation of the second argument delimiter and the first argument of this function.

2. Add processing to the repeated row in a recursive query.

The repeated row of a recursive query is specified in the **SELECT** list after **UNION ALL** in the **SELECT** statement. Add to this **SELECT** list the equivalent value of the repeated row of the **SYS_CONNECT_BY_PATH** function. The value to add is the string concatenation of the parent row, the second argument delimiter, and the first argument of this function.

Example C-56 and Example C-57 on page 405 demonstrate how Oracle Database handles the **SYS_CONNECT_BY_PATH** function. Example C-58 on page 405 and Example C-59 on page 405 demonstrate how FUJITSU Enterprise Postgres handles the **SYS_CONNECT_BY_PATH** function.

Example: C-56 [Oracle-SQL] SYS_CONNECT_BY_PATH

```
CREATE TABLE TBL(
  ID NUMBER,
  NAME VARCHAR2(10),
  PARENTID NUMBER
);
INSERT INTO TBL VALUES (1, 'A', NULL);
INSERT INTO TBL VALUES (2, 'A1', 1);
INSERT INTO TBL VALUES (3, 'A2', 1);
```

```

INSERT INTO TBL VALUES (4, 'A3', 1);
INSERT INTO TBL VALUES (5, 'A11', 2);
INSERT INTO TBL VALUES (6, 'A21', 3);
INSERT INTO TBL VALUES (7, 'A22', 3);
INSERT INTO TBL VALUES (8, 'A221', 7);
SELECT SYS_CONNECT_BY_PATH(NAME, '/') AS PATH FROM TBL
START WITH PARENTID IS NULL
CONNECT BY PRIOR ID = PARENTID
ORDER BY PATH;

```

Example: C-57 [Oracle-Result] SYS_CONNECT_BY_PATH

PATH

```

-----
/A
/A/A1
/A/A1/A11
/A/A2
/A/A2/A21
/A/A2/A22
/A/A2/A22/A221
/A/A3

```

Example: C-58 [FUJITSU Enterprise Postgres: SQL] SYS_CONNECT_BY_PATH

```

CREATE TABLE TBL(
  ID NUMERIC,
  NAME VARCHAR(10),
  PARENTID NUMERIC
);
INSERT INTO TBL VALUES (1, 'A', NULL);
INSERT INTO TBL VALUES (2, 'A1', 1);
INSERT INTO TBL VALUES (3, 'A2', 1);
INSERT INTO TBL VALUES (4, 'A3', 1);
INSERT INTO TBL VALUES (5, 'A11', 2);
INSERT INTO TBL VALUES (6, 'A21', 3);
INSERT INTO TBL VALUES (7, 'A22', 3);
INSERT INTO TBL VALUES (8, 'A221', 7);
WITH RECURSIVE W1 AS (
  SELECT TBL.*, '/' || NAME AS PATH
  FROM TBL WHERE PARENTID IS NULL
  UNION ALL
  SELECT TBL.*, W1.PATH || '/' || TBL.NAME
  FROM TBL INNER JOIN W1 ON TBL.PARENTID = W1.ID
)
SELECT PATH FROM W1
ORDER BY PATH;

```

Example: C-59 [FUJITSU Enterprise Postgres-Result] SYS_CONNECT_BY_PATH

path

```

-----
/A
/A/A1
/A/A1/A11
/A/A2

```

/A/A2/A21
 /A/A2/A22
 /A/A2/A22/A221
 /A/A3

Others

This section describes the following topics:

- ▶ Migration pattern: Database object name
- ▶ Migration pattern: Implicit conversion
- ▶ Migration pattern: Zero-length strings
- ▶ Migration pattern: Comparing fixed-length character strings and variable-length character strings

Migration pattern: Database object name

If a database object name is specified as an unquoted identifier, Oracle Database parses it as uppercase, but FUJITSU Enterprise Postgres parses it as lowercase. Therefore, in some cases, Oracle Database might parse an object name as the same name, but FUJITSU Enterprise Postgres might parse it as a different name when running the same code. For example, a database object whose name is defined with a quoted identifier might be defined by an unquoted identifier, or a database object whose name is defined with an unquoted identifier might be defined by a quoted identifier. Thus, when using quoted identifiers in FUJITSU Enterprise Postgres, consider that the object name is identified as either uppercase or lowercase.

As a best practice, define database object names with an unquoted identifier and refer to them by using an unquoted identifier when migrating to FUJITSU Enterprise Postgres because the quoted identifier is supported on FUJITSU Enterprise Postgres but not accepted by some tools that manage database objects.

Example C-60 and Example C-61 provide examples when using Oracle Database, and Example C-62 on page 407 and Example C-63 on page 407 provide examples when using FUJITSU Enterprise Postgres.

Example: C-60 [Oracle-SQL] Database object name

```
CREATE TABLE "TBL_1" ("COL_1" NUMBER, "COL_2" CHAR(5));
CREATE TABLE TBL_2 ( COL_1 NUMBER, COL_2 CHAR(5));
INSERT INTO TBL_1 ( COL_1 , COL_2 ) VALUES(1, 'AAAAA');
INSERT INTO "TBL_2" ("COL_1", "COL_2") VALUES(2, 'BBBBB');
SELECT COL_1, COL_2 FROM TBL_1 UNION
SELECT "COL_1", "COL_2" FROM "TBL_2"
ORDER BY COL_1;
```

Example: C-61 [Oracle-Result] Database object name

```
COL_1 COL_2
-----
1 AAAAA
2 BBBBB
```

Example: C-62 [FUJITSU Enterprise Postgres: SQL] Database object name

```
CREATE TABLE TBL_1 ( COL_1 NUMERIC, COL_2 CHAR(5));
CREATE TABLE TBL_2 ( COL_1 NUMERIC, COL_2 CHAR(5));
INSERT INTO TBL_1 ( COL_1 , COL_2 ) VALUES(1, 'AAAAA');
INSERT INTO TBL_2 ( COL_1 , COL_2 ) VALUES(2, 'BBBBB');
SELECT COL_1, COL_2 FROM TBL_1 UNION
SELECT COL_1, COL_2 FROM TBL_2
ORDER BY COL_1;
```

Example: C-63 [FUJITSU Enterprise Postgres-Result] Database Object name

```
col_1 | col_2
-----+-----
    1 | AAAAA
    2 | BBBBB
```

Migration pattern: Implicit conversion

Oracle Database and FUJITSU Enterprise Postgres have different conditions when implicit data conversion is enabled. For example, when comparing string data and numeric data, Oracle Database implicitly converts the string type to the numeric type and succeeds when comparing them, but FUJITSU Enterprise Postgres does not convert implicitly and a comparison results in an error. Therefore, code with implicit conversion enabled must be modified to perform explicit data conversion when migrating.

Example C-64 and Example C-65 provide examples when using an Oracle Database, and Example C-66 and Example C-67 on page 408 provide examples when using FUJITSU Enterprise Postgres.

Example: C-64 [Oracle-SQL] Implicit conversion

```
CREATE TABLE TBL_1(COL_1 CHAR(5), COL_2 CHAR(5));
CREATE TABLE TBL_2(COL_1 CHAR(5), COL_2 NUMBER);
INSERT INTO TBL_1 VALUES('AAAAA', '11111');
INSERT INTO TBL_1 VALUES('BBBBB', '22222');
INSERT INTO TBL_2 VALUES('AAAAA', 11111);
INSERT INTO TBL_2 VALUES('BBBBB', 22222);
SELECT TBL_1.COL_1 FROM TBL_1, TBL_2
WHERE TBL_1.COL_2 = TBL_2.COL_2;
```

Example: C-65 [Oracle-Result] Implicit conversion

```
COL_1
-----
AAAAA
BBBBB
```

Example: C-66 [FUJITSU Enterprise Postgres: SQL] Implicit conversion

```
CREATE TABLE TBL_1(COL_1 CHAR(5), COL_2 CHAR(5));
CREATE TABLE TBL_2(COL_1 CHAR(5), COL_2 NUMERIC);
INSERT INTO TBL_1 VALUES('AAAAA', '11111');
INSERT INTO TBL_1 VALUES('BBBBB', '22222');
INSERT INTO TBL_2 VALUES('AAAAA', 11111);
INSERT INTO TBL_2 VALUES('BBBBB', 22222);
SELECT TBL_1.COL_1 FROM TBL_1, TBL_2
WHERE CAST(TBL_1.COL_2 AS NUMERIC) = TBL_2.COL_2;
```

Example: C-67 [FUJITSU Enterprise Postgres-Result] Implicit conversion

```
col_1
-----
AAAAA
BBBBB
```

Migration pattern: Zero-length strings

Oracle Database treats zero-length strings as NULL, but FUJITSU Enterprise Postgres treats zero-length strings with a different value than NULL, so replace the value with NULL when migrating.

Example C-68 and Example C-69 provide examples when using an Oracle Database, and Example C-70 and Example C-71 provide examples when using FUJITSU Enterprise Postgres.

Example: C-68 [Oracle-SQL] Zero-length string

```
CREATE TABLE TBL(COL_1 CHAR(5), COL_2 CHAR(5));
INSERT INTO TBL VALUES('xxxxx', 'XXXXX');
INSERT INTO TBL VALUES('yyyyy', '');
INSERT INTO TBL VALUES('zzzzz', NULL);
SELECT * FROM TBL WHERE COL_2 IS NULL;
```

Example: C-69 [Oracle-Result] Zero-length string

```
COL_1 COL_2
-----
yyyyy
zzzzz
```

Example: C-70 [FUJITSU Enterprise Postgres: SQL] Zero-length string

```
CREATE TABLE TBL(COL_1 CHAR(5), COL_2 CHAR(5));
INSERT INTO TBL VALUES('xxxxx', 'XXXXX');
INSERT INTO TBL VALUES('yyyyy', NULL);
INSERT INTO TBL VALUES('zzzzz', NULL);
SELECT * FROM TBL WHERE COL_2 IS NULL;
```

Example: C-71 [FUJITSU Enterprise Postgres-Result] Zero-length string

```
col_1 | col_2
-----+-----
yyyyy |
zzzzz |
```

Migration pattern: Comparing fixed-length character strings and variable-length character strings

Even if the value that is stored in a variable is the same, Oracle Database determines that the value that is specified as a fixed-length character string data type, such as CHAR, does not match the value that is specified as a variable-length character string data type, such as VARCHAR2 because Oracle Database treats trailing spaces in fixed-length strings and variable-length strings as valid values for comparison.

FUJITSU Enterprise Postgres behaves differently than Oracle Database when comparing fixed-length string data of different lengths with variable-length string data. FUJITSU Enterprise Postgres removes or adds trailing spaces in fixed-length strings to match the length of value of the fixed-length string to be that of the length of the variable-length string before comparing. If trailing spaces are removed or added but the length of data does not match, these values are determined not to match.

Because of these differences in the data comparison processing, Oracle Database determines the strings to have different values, but FUJITSU Enterprise Postgres might determine that they have the same value. For example, you might be performing processing in an application that inserts the same data into columns of different tables where one column is defined as a fixed-length string data type and the other column is defined as a variable-length string data type. If we accidentally insert a value that is the same string as the fixed-length column data but has a different length because of trailing spaces into a variable-length column, Oracle Database determines that the two columns have different values. However, FUJITSU Enterprise Postgres determines that the two columns have the same value.

When migrating the SQL for a comparison between fixed-length character string data and variable-length character string data, use the **RPAD** function for the fixed-length character string data on FUJITSU Enterprise Postgres. Specify fixed-length character string data in the first argument of **RPAD**, and the length of the data in the second argument. The result of this **RPAD** function is the same value as a variable-length character string and the same length as a fixed-length character string with trailing spaces. So, FUJITSU Enterprise Postgres can compare the values, including trailing spaces, and the comparison result is the same as thought it were done in an Oracle Database.

Example C-72 and Example C-73 provide examples when using an Oracle Database, and Example C-74 on page 410 and Example C-75 on page 410 provide examples when using FUJITSU Enterprise Postgres.

Example: C-72 [Oracle-SQL] Comparing a fixed-length character string and variable-length character string

```
CREATE TABLE TBL_1(COL_1 CHAR(5));
CREATE TABLE TBL_2(COL_1 VARCHAR2(5));
INSERT INTO TBL_1 VALUES('AAA');
INSERT INTO TBL_1 VALUES('BBB');
INSERT INTO TBL_1 VALUES('CCC');
INSERT INTO TBL_2 VALUES('AAA ');
INSERT INTO TBL_2 VALUES('BBB');
INSERT INTO TBL_2 VALUES('CCC ');
SELECT TBL_1.COL_1 FROM TBL_1, TBL_2
WHERE TBL_1.COL_1 = TBL_2.COL_1
ORDER BY COL_1;
```

Example: C-73 [Oracle-Result] Comparing a fixed-length character string and variable-length character string

```
COL_1
-----
AAA
CCC
```

Example: C-74 [FUJITSU Enterprise Postgres: SQL] Comparing a fixed-length character string and variable-length character string

```
CREATE TABLE TBL_1(COL_1 CHAR(5));
CREATE TABLE TBL_2(COL_1 VARCHAR(5));
INSERT INTO TBL_1 VALUES('AAA');
INSERT INTO TBL_1 VALUES('BBB');
INSERT INTO TBL_1 VALUES('CCC');
INSERT INTO TBL_2 VALUES('AAA ');
INSERT INTO TBL_2 VALUES('BBB');
INSERT INTO TBL_2 VALUES('CCC ');
SELECT TBL_1.COL_1 FROM TBL_1, TBL_2
WHERE RPAD(TBL_1.COL_1, 5) = TBL_2.COL_1
ORDER BY COL_1;
```

Example: C-75 [FUJITSU Enterprise Postgres-Result] Comparing a fixed-length character string and variable-length character string

```
col_1
-----
AAA
CCC
```

PL/SQL

This section describes the following topics:

- ▶ Database trigger migration pattern
- ▶ Cursors
- ▶ Error handling
- ▶ Stored functions
- ▶ Stored procedures migration pattern
- ▶ Other migration patterns

Database trigger migration pattern

The trigger function is supported on FUJITSU Enterprise Postgres and Oracle Database. However, there are differences between FUJITSU Enterprise Postgres and Oracle Database when defining triggers. Three major differences are as follows:

- ▶ How to describe trigger processing.

In Oracle Database, trigger processing is defined in the **CREATE TRIGGER** statement, and in FUJITSU Enterprise Postgres, trigger processing is defined as a user-defined function, and the **CREATE TRIGGER** statement defines the function as a trigger.

In addition to the difference of defining trigger processing, due to other syntax differences, consider using removing options that are not supported by FUJITSU Enterprise Postgres.

- ▶ How to determine the event trigger.

When an event fires a trigger, Oracle Database uses a conditional predicate to determine which event fired. The conditional predicates, such as **INSERTING**, **UPDATING**, and **DELETING**, are specified in statements such as conditional selection statements. FUJITSU Enterprise Postgres stores the information of the triggering event in the *TG_OP* variable. So, FUJITSU Enterprise Postgres can determine from which operation the trigger was fired by using the value of *TG_OP* variable. The value is a string of **INSERT**, **UPDATE**, **DELETE**, or **TRUNCATE** statements.

- ▶ How to read OLD and NEW values of pseudorecords.

To read OLD or NEW values of pseudorecords in an Oracle Database trigger, use `:OLD` or `:NEW`. In FUJITSU Enterprise Postgres, use a RECORD data type, such as OLD or NEW to read their values. To use these data type in FUJITSU Enterprise Postgres, specify OLD or NEW without including a colon (:).

Example C-76 and Example C-77 on page 412 provide examples when using Oracle Database, and Example C-78 on page 412 and Example C-79 on page 413 provide examples when using FUJITSU Enterprise Postgres.

Example: C-76 [Oracle -PL/SQL] Database triggers

```
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD';
CREATE TABLE department (
    employee_id NUMBER,
    department VARCHAR2(10)
);
CREATE TABLE department_history (
    employee_id NUMBER,
    old_department VARCHAR2(10),
    new_department VARCHAR2(10),
    change_date DATE
);
CREATE OR REPLACE TRIGGER save_history
BEFORE INSERT OR UPDATE OR DELETE ON department
FOR EACH ROW
BEGIN
    IF INSERTING THEN
        INSERT INTO department_history VALUES (
            :NEW.employee_id, NULL,
            :NEW.department, SYSDATE
        );
    ELSIF UPDATING THEN
        INSERT INTO department_history VALUES (
            :OLD.employee_id, :OLD.department,
            :NEW.department, SYSDATE
        );
    ELSIF DELETING THEN
        INSERT INTO department_history VALUES (
            :OLD.employee_id, :OLD.department,
            NULL, SYSDATE
        );
    END IF;
END;
/
INSERT INTO department VALUES(1000, 'ABC Dept. ');
INSERT INTO department VALUES(2000, 'DEF Dept. ');
```

```

UPDATE department SET department = 'XYZ Dept.'
  WHERE employee_id = 1000;
DELETE FROM department WHERE employee_id = 2000;
SELECT * FROM department;
SELECT * FROM department_history;

```

Example: C-77 [Oracle-Result] Database triggers

```

EMPLOYEE_ID DEPARTMENT
-----
          1000 XYZ Dept.
EMPLOYEE_ID OLD_DEPART NEW_DEPART CHANGE_DAT
-----
          1000          ABC Dept.  2021-10-21
          2000          DEF Dept.  2021-10-21
          1000 ABC Dept.  XYZ Dept.  2021-10-21
          2000 DEF Dept.          2021-10-21

```

Example: C-78 [FUJITSU Enterprise Postgres-PL/pgSQL] Database triggers

```

CREATE TABLE department (
  employee_id INT,
  department VARCHAR(10)
);
CREATE TABLE department_history (
  employee_id INT,
  old_department VARCHAR(10),
  new_department VARCHAR(10),
  change_date DATE
);
CREATE OR REPLACE FUNCTION save_history_function()
RETURNS TRIGGER
AS $$
BEGIN
  IF (TG_OP = 'INSERT') THEN
    INSERT INTO department_history VALUES (
      NEW.employee_id, NULL,
      NEW.department, statement_timestamp()
    );
    RETURN NEW;
  ELSIF (TG_OP = 'UPDATE') THEN
    INSERT INTO department_history VALUES (
      OLD.employee_id, OLD.department,
      NEW.department, statement_timestamp()
    );
    RETURN NEW;
  ELSIF (TG_OP = 'DELETE') THEN
    INSERT INTO department_history VALUES (
      OLD.employee_id, OLD.department,
      NULL, statement_timestamp()
    );
    RETURN OLD;
  END IF;
END;
$$ LANGUAGE plpgsql;
CREATE OR REPLACE TRIGGER save_history

```

```

BEFORE INSERT OR UPDATE OR DELETE ON department
FOR EACH ROW
EXECUTE FUNCTION save_history_function();
INSERT INTO department VALUES(1000, 'ABC Dept. ');
INSERT INTO department VALUES(2000, 'DEF Dept. ');
UPDATE department SET department = 'XYZ Dept.' WHERE employee_id = 1000;
DELETE FROM department WHERE employee_id = 2000;
SELECT * FROM department;
SELECT * FROM department_history;

```

Example: C-79 [FUJITSU Enterprise Postgres - Result] Database triggers

employee_id	department
1000	XYZ Dept.

employee_id	old_department	new_department	change_date
1000		ABC Dept.	2021-10-21
2000		DEF Dept.	2021-10-21
1000	ABC Dept.	XYZ Dept.	2021-10-21
2000	DEF Dept.		2021-10-21

Cursors

Cursors are supported on FUJITSU Enterprise Postgres and on Oracle Database. However, some functions might need to be modified when migrating because there are some incompatibilities.

In this section, two major differences are described: One is cursor attributes, and the other is cursor variables. These examples of FUJITSU Enterprise Postgres enable Oracle compatibility features.

Migration pattern: Cursor attributes

FUJITSU Enterprise Postgres does not support cursor attributes the same way on Oracle Database. However, there is an alternative way to retrieve information that is equivalent to the Oracle Database cursor attribute, except for **%ISOPEN**.

Each cursor attribute is converted in the following ways:

► **%ISOPEN**

FUJITSU Enterprise Postgres does not have the equivalent function of **%ISOPEN**.

When migrating to FUJITSU Enterprise Postgres, define a variable to store information about whether a cursor is open, and use this variable to manage the cursor state as part of PL/pgSQL processing.

In the exception-handling part of the block in which a cursor was opened, **%ISOPEN** might be used to determine whether to close the cursor. If **%ISOPEN** is used for that purpose only, remove the decision processing and cursor close process when migrating to FUJITSU Enterprise Postgres because it automatically closes the cursor.

► **%NOTFOUND**

Use the *FOUND* variable in FUJITSU Enterprise Postgres to get the same information as **%NOTFOUND**.

► **%FOUND**

Use the *FOUND* variable in FUJITSU Enterprise Postgres to get the same information as **%FOUND**.

► **%ROWCOUNT**

GET DIAGNOSTICS enables you to get the number of rows that is processed by the last SQL in FUJITSU Enterprise Postgres. Use **GET DIAGNOSTICS** to retrieve the information that is retrieved by **%ROWCOUNT**.

Example C-80 and Example C-81 on page 415 provide examples when using Oracle Database, and Example C-82 on page 415 and Example C-83 on page 416 provide examples when using FUJITSU Enterprise Postgres.

Example: C-80 [Oracle -PL/SQL] Cursor attribute

```

CREATE TABLE cur_tbl(
  id NUMBER,
  val VARCHAR2(10)
);
INSERT INTO cur_tbl(id, val) (
  SELECT LEVEL, 'data' || LEVEL FROM DUAL
  CONNECT BY LEVEL <= 10
);
DECLARE
  TYPE cur_type IS REF CURSOR;
  cur1 cur_type;
  cur2 cur_type;
  var_id PLS_INTEGER;
  var_val VARCHAR2(10);
  var_count PLS_INTEGER;
BEGIN
  DBMS_OUTPUT.PUT_LINE('**** using cur1 ****');
  IF NOT cur1%ISOPEN THEN
    OPEN cur1 FOR
      SELECT id, val FROM cur_tbl WHERE id < 3;
  END IF;
  LOOP
    FETCH cur1 INTO var_id, var_val;
    IF cur1%FOUND THEN
      DBMS_OUTPUT.PUT_LINE(
        'id=' || var_id || ', val=' || var_val);
    ELSE
      EXIT;
    END IF;
  END LOOP;
  CLOSE cur1;
  DBMS_OUTPUT.PUT_LINE('**** using cur2 ****');
  IF NOT cur2%ISOPEN THEN
    OPEN cur2 FOR
      SELECT id, val FROM cur_tbl WHERE id > 100;
  END IF;
  LOOP
    FETCH cur2 INTO var_id, var_val;
    IF cur2%ROWCOUNT = 0 THEN
      DBMS_OUTPUT.PUT_LINE('No data found');
      EXIT;
    
```

```

        END IF;
        EXIT WHEN cur2%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(
            'id=' || var_id || ', row_number=' || cur2%ROWCOUNT);
    END LOOP;
    CLOSE cur2;
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('exception block');
        IF cur1%ISOPEN THEN
            CLOSE cur1;
        END IF;
        IF cur2%ISOPEN THEN
            CLOSE cur2;
        END IF;
END;
/

```

Example: C-81 [Oracle-Result] Cursor attribute

```

**** using cur1 ****
id=1, val=data1
id=2, val=data2
**** using cur2 ****
No data found
PL/SQL procedure successfully completed.

```

Example: C-82 [FUJITSU Enterprise Postgres-PL/pgSQL] Cursor attribute

```

CREATE TABLE cur_tbl(
    id INT,
    val VARCHAR(10)
);
INSERT INTO cur_tbl(id, val) (
    SELECT
        generate_series, 'data' || generate_series
    FROM generate_series(1, 10)
);
DO $$
DECLARE
    cur1 REFCURSOR;
    cur2 REFCURSOR;
    var_id INT;
    var_val VARCHAR(10);
    var_count INT;
    row_count INT;
    cur1_isopen boolean := FALSE;
    cur2_isopen boolean := FALSE;
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    PERFORM DBMS_OUTPUT.PUT_LINE('**** using cur1 ****');
    IF NOT cur1_isopen THEN
        OPEN cur1 FOR
            SELECT id, val FROM cur_tbl WHERE id < 3;
        cur1_isopen := TRUE;
    END IF;

```

```

LOOP
  FETCH cur1 INTO var_id, var_val;
  IF FOUND THEN
    PERFORM DBMS_OUTPUT.PUT_LINE(
      'id=' || var_id || ', val=' || var_val);
  ELSE
    EXIT;
  END IF;
END LOOP;
CLOSE cur1;
cur1_isopen := FALSE;
PERFORM DBMS_OUTPUT.PUT_LINE('**** using cur2 ****');
IF NOT cur2_isopen THEN
  OPEN cur2 FOR
    SELECT id, val FROM cur_tbl WHERE id > 100;
  cur2_isopen := TRUE;
  var_count := 0;
END IF;
LOOP
  FETCH cur2 INTO var_id, var_val;
  GET DIAGNOSTICS row_count = ROW_COUNT;
  var_count := var_count + row_count;
  IF var_count = 0 THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('No data found');
    EXIT;
  END IF;
  EXIT WHEN NOT FOUND;
  PERFORM DBMS_OUTPUT.PUT_LINE(
    'id=' || var_id || ', row_number=' || var_count);
END LOOP;
CLOSE cur2;
cur2_isopen := FALSE;
EXCEPTION
  WHEN OTHERS THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('exception block');
END;
$$ LANGUAGE plpgsql;

```

Example: C-83 [FUJITSU Enterprise Postgres- Result] Cursor attribute

```

**** using cur1 ****
id=1, val=data1
id=2, val=data2
**** using cur2 ****
No data found
DO

```

Migration pattern: Cursor variables

To create a cursor variable, use the REF CURSOR type on Oracle Database. FUJITSU Enterprise Postgres does not support it, so you should use the refcursor data type instead.

When migrating to FUJITSU Enterprise Postgres, remove the REF CURSOR definition, and define the variable that was defined as REF CURSOR in Oracle Database as the refcursor data type.

Example C-84 and Example C-85 provide examples when using Oracle Database, and Example C-86 and Example C-87 on page 418 provide examples when using FUJITSU Enterprise Postgres.

Example: C-84 [Oracle-PL/SQL] Cursor variables

```
CREATE TABLE cur_tbl(
  id NUMBER,
  val VARCHAR2(10)
);
INSERT INTO cur_tbl(id, val) (
  SELECT LEVEL, 'data' || LEVEL FROM DUAL
  CONNECT BY LEVEL <= 10
);
DECLARE
  TYPE cur_type IS REF CURSOR;
  cur cur_type;
  var_id PLS_INTEGER;
  var_val VARCHAR2(10);
BEGIN
  IF NOT cur%ISOPEN THEN
    OPEN cur FOR
      SELECT id, val FROM cur_tbl WHERE id < 3;
  END IF;
  LOOP
    FETCH cur INTO var_id, var_val;
    EXIT WHEN cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(
      'id=' || var_id || ', val=' || var_val);
  END LOOP;
  CLOSE cur;
END;
/
```

Example: C-85 [Oracle-Result] Cursor variables

```
id=1, val=data1
id=2, val=data2
PL/SQL procedure successfully completed.
```

Example: C-86 [FUJITSU Enterprise Postgres-PL/pgSQL] Cursor variables

```
CREATE TABLE cur_tbl(
  id INT,
  val VARCHAR(10)
);
INSERT INTO cur_tbl(id, val) (
  SELECT
    generate_series, 'data' || generate_series
  FROM generate_series(1, 10)
);
DO $$
DECLARE
  cur REFCURSOR;
  var_id INT;
  var_val VARCHAR(10);
```

```

cur_isopen boolean := FALSE;
BEGIN
PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
IF NOT cur_isopen THEN
OPEN cur FOR
SELECT id, val FROM cur_tbl WHERE id < 3;
cur_isopen := TRUE;
END IF;
LOOP
FETCH cur INTO var_id, var_val;
EXIT WHEN NOT FOUND;
PERFORM DBMS_OUTPUT.PUT_LINE(
'id=' || var_id || ', val=' || var_val);
END LOOP;
CLOSE cur;
cur_isopen := FALSE;
END;
$$ LANGUAGE plpgsql;

```

Example: C-87 [FUJITSU Enterprise Postgres- Result] Cursor variables

```

id=1, val=data1
id=2, val=data2
DO

```

Error handling

This section describes the following topics:

- ▶ Migration pattern: Predefined exceptions
- ▶ Migration pattern: SQLCODE

Migration pattern: Predefined exceptions

Oracle Database and FUJITSU Enterprise Postgres support predefined exceptions. However, the exception name and the meaning of the error exception are different in FUJITSU Enterprise Postgres compared to Oracle Database predefined exceptions.

Table C-3 provides the major predefined exceptions of Oracle Database, and examples of what predefined exceptions should be used in FUJITSU Enterprise Postgres when migrating.

Table C-3 Example of converting predefined exceptions

Predefined exceptions of Oracle Database	Brief description of the predefined exceptions	Corresponding predefined exceptions on FUJITSU Enterprise Postgres
CURSOR_ALREADY_OPEN	Attempt to open an already open cursor.	DUPLICATE_CURSOR
DUP_VAL_ON_INDEX	Attempt to store duplicate values in a column with the UNIQUE constraint.	UNIQUE_VIOLATION
INVALID_CURSOR	Attempt an not allowed cursor operation such as fetch by using an unopened cursor.	INVALID_CURSOR_STATE INVALID_CURSOR_NAME

Predefined exceptions of Oracle Database	Brief description of the predefined exceptions	Corresponding predefined exceptions on FUJITSU Enterprise Postgres
INVALID_NUMBER	An invalid number was specified.	INVALID_TEXT_REPRESENTATION INVALID_CHARACTER_VALUE_FOR_CAST NUMERIC_VALUE_OUT_OF_RANGE DATATYPE_MISMATCH
ZERO_DIVIDE	Attempt to divide a number by zero.	DIVISION_BY_ZERO

In this section, we show concrete examples of how to convert the five predefined exceptions that are listed in Table C-3 on page 418 when migrating to FUJITSU Enterprise Postgres. These examples enable Oracle compatibility features.

- ▶ `CURSOR_ALREADY_OPEN`: Shown in Example C-88 through Example C-91 on page 421.
- ▶ `DUP_VAL_ON_INDEX`: Shown in Example C-92 on page 421 through Example C-95 on page 422.
- ▶ `INVALID_CURSOR`: Shown in Example C-96 on page 422 through Example C-99 on page 423.
- ▶ `INVALID_NUMBER`: Shown in Example C-100 on page 423 through Example C-103 on page 424.
- ▶ `ZERO_DIVIDE`: Shown in Example C-104 on page 424 through Example C-107 on page 425.

CURSOR_ALREADY_OPEN

Example: C-88 [Oracle -PL/SQL] Predefined exceptions: CURSOR_ALREADY_OPEN

```
CREATE TABLE cur_tbl(
  id NUMBER,
  val VARCHAR2(10)
);
INSERT INTO cur_tbl(id, val) (
  SELECT LEVEL, 'data' || LEVEL FROM DUAL
  CONNECT BY LEVEL <= 10
);
DECLARE
  CURSOR cur IS SELECT id, val from cur_tbl WHERE id < 3;
  var_id PLS_INTEGER;
  var_val VARCHAR2(10);
BEGIN
  DBMS_OUTPUT.PUT_LINE('Anonymous block: BEGIN');
  OPEN cur;
  LOOP
    FETCH cur INTO var_id, var_val;
    EXIT WHEN cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(
      'id=' || var_id || ', val=' || var_val);
  END LOOP;
  -- various processing
  -- Exception occurs
  OPEN cur;
  -- various processing
  CLOSE cur;
```

```

    DBMS_OUTPUT.PUT_LINE('Anonymous block: END');
EXCEPTION
  WHEN CURSOR_ALREADY_OPEN THEN
    DBMS_OUTPUT.PUT_LINE('CURSOR_ALREADY_OPEN');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('OTHERS');
  IF cur%ISOPEN THEN
    CLOSE cur;
  END IF;
END;
/

```

Example: C-89 [Oracle-Result] Predefined exceptions: CURSOR_ALREADY_OPEN

```

Anonymous block: BEGIN
id=1, val=data1
id=2, val=data2
CURSOR_ALREADY_OPEN
PL/SQL procedure successfully completed.

```

Example: C-90 [FUJITSU Enterprise Postgres-PL/pgSQL] Predefined exceptions: CURSOR_ALREADY_OPEN

```

CREATE TABLE cur_tbl(
  id INT,
  val VARCHAR(10)
);
INSERT INTO cur_tbl(id, val) (
  SELECT
    generate_series, 'data' || generate_series
  FROM generate_series(1, 10)
);
DO $$
DECLARE
  cur CURSOR FOR SELECT id, val from cur_tbl WHERE id < 3;
  var_id INT;
  var_val VARCHAR(10);
BEGIN
  PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
  PERFORM DBMS_OUTPUT.PUT_LINE('Anonymous block: BEGIN');
  OPEN cur;
  LOOP
    FETCH cur INTO var_id, var_val;
    EXIT WHEN NOT FOUND;
    PERFORM DBMS_OUTPUT.PUT_LINE(
      'id=' || var_id || ', val=' || var_val);
  END LOOP;
  -- various processing
  -- Exception occurs
  OPEN cur;
  -- various processing
  CLOSE cur;
  PERFORM DBMS_OUTPUT.PUT_LINE('Anonymous block: END');
EXCEPTION
  WHEN DUPLICATE_CURSOR THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('CURSOR_ALREADY_OPEN');

```

```

    WHEN OTHERS THEN
        PERFORM DBMS_OUTPUT.PUT_LINE('OTHERS');
END;
$$ LANGUAGE plpgsql;

```

Example: C-91 [FUJITSU Enterprise Postgres- Result] Predefined exceptions: CURSOR_ALREADY_OPEN

```

Anonymous block: BEGIN
id=1, val=data1
id=2, val=data2
CURSOR_ALREADY_OPEN
DO

```

DUP_VAL_ON_INDEX

Example: C-92 [Oracle -PL/SQL] Predefined exceptions: DUP_VAL_ON_INDEX

```

CREATE TABLE sample_tbl(
    id NUMBER UNIQUE,
    val VARCHAR2(10)
);
BEGIN
    INSERT INTO sample_tbl VALUES(1, 'data1');
    DBMS_OUTPUT.PUT_LINE('INSERT: data1');
    -- Exception occurs
    INSERT INTO sample_tbl VALUES(1, 'data2');
    DBMS_OUTPUT.PUT_LINE('INSERT: data2');
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('DUP_VAL_ON_INDEX');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('OTHERS');
END;
/

```

Example: C-93 [Oracle-Result] Predefined exceptions: DUP_VAL_ON_INDEX

```

INSERT: data1
DUP_VAL_ON_INDEX
PL/SQL procedure successfully completed.

```

Example: C-94 [FUJITSU Enterprise Postgres-PL/pgSQL] Predefined exceptions: DUP_VAL_ON_INDEX

```

CREATE TABLE sample_tbl(
    id INT UNIQUE,
    val VARCHAR(10)
);
DO $$
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    INSERT INTO sample_tbl VALUES(1, 'data1');
    PERFORM DBMS_OUTPUT.PUT_LINE('INSERT: data1');
    -- Exception occurs
    INSERT INTO sample_tbl VALUES(1, 'data2');
    PERFORM DBMS_OUTPUT.PUT_LINE('INSERT: data2');

```

```

EXCEPTION
  WHEN UNIQUE_VIOLATION THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('DUP_VAL_ON_INDEX');
  WHEN OTHERS THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('OTHERS');
END;
$$ LANGUAGE plpgsql;

```

Example: C-95 [FUJITSU Enterprise Postgres- Result] Predefined exceptions: DUP_VAL_ON_INDEX

```

INSERT: data1
DUP_VAL_ON_INDEX
DO

```

INVALID_CURSOR

Example: C-96 [Oracle -PL/SQL] Predefined exceptions: INVALID_CURSOR

```

CREATE TABLE cur_tbl(
  id NUMBER,
  val VARCHAR2(10)
);
INSERT INTO cur_tbl(id, val) (
  SELECT LEVEL, 'data' || LEVEL FROM DUAL
  CONNECT BY LEVEL <= 10
);
DECLARE
  CURSOR cur IS SELECT id, val FROM cur_tbl WHERE id = 1;
  var_id PLS_INTEGER;
  var_val VARCHAR2(10);
BEGIN
  DBMS_OUTPUT.PUT_LINE('Anonymous block: BEGIN');
  LOOP
    -- Exception occurs
    FETCH cur INTO var_id, var_val;
    EXIT WHEN cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('id:' || var_id);
    DBMS_OUTPUT.PUT_LINE('val:' || var_val);
  END LOOP;

  CLOSE cur;
  DBMS_OUTPUT.PUT_LINE('Anonymous block: END');
EXCEPTION
  WHEN INVALID_CURSOR THEN
    DBMS_OUTPUT.PUT_LINE('INVALID_CURSOR');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('OTHERS');
END;
/

```

Example: C-97 [Oracle-Result] Predefined exceptions: INVALID_CURSOR

```

Anonymous block: BEGIN
INVALID_CURSOR
PL/SQL procedure successfully completed.

```

Example: C-98 [FUJITSU Enterprise Postgres-PL/pgSQL] Predefined exceptions: INVALID_CURSOR

```

CREATE TABLE cur_tbl(
  id INT,
  val VARCHAR(10)
);
INSERT INTO cur_tbl(id, val) (
  SELECT
    generate_series, 'data' || generate_series
  FROM generate_series(1, 10)
);
DO $$
DECLARE
  cur CURSOR FOR SELECT id, val FROM cur_tbl WHERE id = 1;
  var_id INT;
  var_val VARCHAR(10);
BEGIN
  PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
  PERFORM DBMS_OUTPUT.PUT_LINE('Anonymous block: BEGIN');
  LOOP
    -- Exception occurs
    FETCH cur INTO var_id, var_val;
    EXIT WHEN NOT FOUND;
    PERFORM DBMS_OUTPUT.PUT_LINE('id:' || var_id);
    PERFORM DBMS_OUTPUT.PUT_LINE('val:' || var_val);
  END LOOP;

  CLOSE cur;
  PERFORM DBMS_OUTPUT.PUT_LINE('Anonymous block: END');
EXCEPTION
  WHEN INVALID_CURSOR_STATE OR INVALID_CURSOR_NAME THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('INVALID_CURSOR');
  WHEN OTHERS THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('OTHERS');
END;
$$ LANGUAGE plpgsql;

```

Example: C-99 [FUJITSU Enterprise Postgres- Result] Predefined exceptions: INVALID_CURSOR

```

Anonymous block: BEGIN
INVALID_CURSOR
DO

```

INVALID_NUMBER

Example: C-100 [Oracle -PL/SQL] Predefined exceptions: INVALID_NUMBER

```

CREATE TABLE sample_tbl(
  id NUMBER,
  val VARCHAR2(10)
);
DECLARE
  id CHAR(5) := 'a001';
  val VARCHAR2(10) := 'data';
BEGIN
  -- Exception occurs
  INSERT INTO sample_tbl VALUES(CAST(id AS NUMBER), val);

```

```

EXCEPTION
  WHEN INVALID_NUMBER THEN
    DBMS_OUTPUT.PUT_LINE('INVALID_NUMBER');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('OTHERS');
END;
/

```

Example: C-101 [Oracle-Result] Predefined exceptions: INVALID_NUMBER

```

INVALID_NUMBER
PL/SQL procedure successfully completed.

```

Example: C-102 [FUJITSU Enterprise Postgres-PL/pgSQL] Predefined exceptions: INVALID_NUMBER

```

CREATE TABLE sample_tbl(
  id INT,
  val VARCHAR(10)
);
DO $$
DECLARE
  id CHAR(5) := 'a001';
  val VARCHAR(10) := 'data';
BEGIN
  PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
  -- Exception occurs
  INSERT INTO sample_tbl VALUES(CAST(id AS INT), val);
EXCEPTION
  WHEN invalid_text_representation OR
  invalid_character_value_for_cast OR
  numeric_value_out_of_range OR
  datatype_mismatch THEN

  PERFORM DBMS_OUTPUT.PUT_LINE('INVALID_NUMBER');
  WHEN OTHERS THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('OTHERS');
END;
$$ LANGUAGE plpgsql;

```

Example: C-103 [FUJITSU Enterprise Postgres- Result] Predefined exceptions: INVALID_NUMBER

```

INVALID_NUMBER
DO

```

ZERO_DIVIDE

Example: C-104 [Oracle -PL/SQL] Predefined exceptions: ZERO_DIVIDE

```

DECLARE
  day_num NUMBER := 0;
  total_sales NUMBER := 0;
  sales_avg NUMBER := 0;
BEGIN
  -- Exception occurs
  sales_avg := total_sales / day_num;
  DBMS_OUTPUT.PUT_LINE(

```



```

        'day_num=' || day_num || ', total_sales=' || total_sales
        || ', sales_average=' || sales_avg);
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('ZERO_DIVIDE');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('OTHERS');
END;
/

```

Example: C-105 [Oracle-Result] Predefined exceptions: ZERO_DIVIDE

```

ZERO_DIVIDE
PL/SQL procedure successfully completed.

```

Example: C-106 [FUJITSU Enterprise Postgres-PL/pgSQL] Predefined exceptions: ZERO_DIVIDE

```

DO $$
DECLARE
  day_num NUMERIC := 0;
  total_sales NUMERIC := 0;
  sales_avg NUMERIC := 0;
BEGIN
  PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
  -- Exception occurs
  sales_avg := total_sales / day_num;
  PERFORM DBMS_OUTPUT.PUT_LINE(
    'day_num=' || day_num || ', total_sales=' || total_sales
    || ', sales_average=' || sales_avg);
EXCEPTION
  WHEN DIVISION_BY_ZERO THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('ZERO_DIVIDE');
  WHEN OTHERS THEN
    PERFORM DBMS_OUTPUT.PUT_LINE('OTHERS');
END;
$$ LANGUAGE plpgsql;

```

Example: C-107 [FUJITSU Enterprise Postgres- Result] Predefined exceptions: ZERO_DIVIDE

```

ZERO_DIVIDE
DO

```

Migration pattern: SQLCODE

Oracle Database uses the **SQLCODE** function to get error codes, and FUJITSU Enterprise Postgres uses the **SQLSTATE** variable to retrieve the error code instead of **SQLCODE**. However, the value of **SQLCODE** and **SQLSTATE** is different.

In addition, there is a difference in data type of the returned value. **SQLCODE** returns a value of numeric data type, and **SQLSTATE** returns an alphanumeric value of a string data type. Therefore, if the process is created with the expectation of returning a numeric value, you must modify the process to use the string data type.

These examples of FUJITSU Enterprise Postgres enable Oracle compatibility features.

Compare the handling of the `SQLCODE` function in Example C-108 and Example C-109 from Oracle Database with the handling of `SQLSTATE` in Example C-110 and Example C-111 on page 427 with FUJITSU Enterprise Postgres.

Example: C-108 [Oracle -PL/SQL] SQLCODE

```
CREATE TABLE tbl(
  id NUMBER PRIMARY KEY,
  val VARCHAR2(10)
);
INSERT INTO tbl VALUES(1, 'data');
DECLARE
  var_sqlcode NUMBER;
  var_errormsg VARCHAR2(100);
BEGIN
  INSERT INTO tbl VALUES(1, 'abcde');
  DBMS_OUTPUT.PUT_LINE('Insert completed');
EXCEPTION
  WHEN OTHERS THEN
    var_sqlcode := SQLCODE;
    var_errormsg := SUBSTR(SQLERRM, 1, 100);
    DBMS_OUTPUT.PUT_LINE(
      'Error: code=' || var_sqlcode
      || ', message=' || var_errormsg
    );
END;
/
```

Example: C-109 [Oracle-Result] SQLCODE

```
Error: code=-1, message=ORA-00001: unique constraint (TESTUSER01.SYS_C007646)
violated
PL/SQL procedure successfully completed.
```

Example: C-110 [FUJITSU Enterprise Postgres-PL/pgSQL] SQLCODE

```
CREATE TABLE tbl(
  id NUMERIC PRIMARY KEY,
  val VARCHAR(10)
);
INSERT INTO tbl VALUES(1, 'data');
DO $$
DECLARE
  var_sqlcode CHAR(5);
  var_errormsg VARCHAR(100);
BEGIN
  PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
  INSERT INTO tbl VALUES(1, 'abcde');
  PERFORM DBMS_OUTPUT.PUT_LINE('Insert completed');
EXCEPTION
  WHEN OTHERS THEN
    var_sqlcode := SQLSTATE;
    var_errormsg := SUBSTR(SQLERRM, 1, 100);
    PERFORM DBMS_OUTPUT.PUT_LINE(
      'Error: code=' || var_sqlcode
      || ', message=' || var_errormsg
    );
END;
```

```
);
END;
$$ LANGUAGE plpgsql;
```

Example: C-111 [FUJITSU Enterprise Postgres- Result] SQLCODE

```
Error: code=23505, message=duplicate key value violates unique constraint
"tbl_pkey" (10232)
DO
```

Stored functions

This section describes the following topics:

- ▶ Migration pattern: Stored functions
- ▶ Migration pattern: Stored functions (performance improvements)

Migration pattern: Stored functions

Oracle Database and FUJITSU Enterprise Postgres support stored functions. However, there are differences between them, such as syntax.

Here are the major differences.

- ▶ The equivalent property of **AUTHID** in Oracle Database is **SECURITY** in FUJITSU Enterprise Postgres. However, the behavior is different when the property is omitted. Therefore, if **AUTHID** is omitted in Oracle Database, specify **SECURITY DEFINER** when migrating to FUJITSU Enterprise Postgres.
- ▶ The **IN OUT** parameter mode that is used in the parameter declaration in Oracle Database must be converted to **INOUT** in FUJITSU Enterprise Postgres.
- ▶ **IN OUT** or **OUT** may be specified in the parameter declaration of Oracle Database, but FUJITSU Enterprise Postgres does not allow the **RETURNS** clause, which is equivalent to the **RETURN** clause in Oracle Database. Therefore, convert as follows when migrating to FUJITSU Enterprise Postgres:
 - a. Add the return value **OUT** as an argument to the parameter declaration.
 - b. Set the return value to the added argument in the stored function processing.
 - c. Receive the return value by using the **SELECT INTO** statement when calling the stored function.

The following examples show the basic conversion method of the stored function. These examples of FUJITSU Enterprise Postgres enable Oracle compatibility features.

Example: C-112 [Oracle -PL/SQL] Stored functions

```
CREATE OR REPLACE FUNCTION create_message(
  user_name VARCHAR2,
  msg IN OUT VARCHAR2
)
  RETURN VARCHAR2
AS
  ret_val VARCHAR2(50);
BEGIN
  ret_val := user_name || ' execute create_message';
  msg := 'NOTICE: ' || msg;
```

```

    RETURN ret_val;
END create_message;
/
DECLARE
    msg1 VARCHAR2(50);
    msg2 VARCHAR2(100);
BEGIN
    msg2 := 'sample message';
    msg1 := create_message('Tom', msg2);
    DBMS_OUTPUT.PUT_LINE(msg1);
    DBMS_OUTPUT.PUT_LINE(msg2);
END;
/

```

Example: C-113 [Oracle-Result] Stored functions

```

Tom execute create_message
NOTICE: sample message
PL/SQL procedure successfully completed.

```

Example: C-114 [FUJITSU Enterprise Postgres-PL/pgSQL] Stored functions

```

CREATE OR REPLACE FUNCTION create_message(
    user_name VARCHAR,
    msg INOUT VARCHAR,
    ret_val OUT VARCHAR
)
AS $$
BEGIN
    ret_val := user_name || ' execute create_message';
    msg := 'NOTICE: ' || msg;
    RETURN;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
DO $$
DECLARE
    msg1 VARCHAR(50);
    msg2 VARCHAR(100);
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    msg2 := 'sample message';
    SELECT * INTO msg2, msg1 FROM create_message('Tom', msg2);
    PERFORM DBMS_OUTPUT.PUT_LINE(msg1);
    PERFORM DBMS_OUTPUT.PUT_LINE(msg2);
END;
$$ LANGUAGE plpgsql;

```

Example: C-115 [FUJITSU Enterprise Postgres- Result] Stored functions

```

Tom execute create_message
NOTICE: sample message
DO

```

Migration pattern: Stored functions (performance improvements)

FUJITSU Enterprise Postgres supports three types of volatility categories: IMMUTABLE, STABLE, and VOLATILE. The volatility category informs the optimizer about the behavior of the function.

► IMMUTABLE

Specify IMMUTABLE if the function does not modify the database and returns the same results with the same arguments forever. That is, the result is determined by the argument value.

► STABLE

Specify STABLE if the function does not modify the database and returns the same results with the same arguments for all rows within a single statement but its result might change across SQL statements. For example, the case is a reference a database and returning results.

► VOLATILE

Specify VOLATILE if the function performs any processing, including the database modification.

By specifying IMMUTABLE or STABLE to stored functions, FUJITSU Enterprise Postgres can optimize processing, which enables improved processing speed. If there are any performance issues after migration, use IMMUTABLE or STABLE.

We use a stored function that returns a message in the following examples:

- Oracle Database stored functions, as shown in Example C-116, with the results shown in Example C-117 on page 430.
- FUJITSU Enterprise Postgres stored functions that are converted from the Oracle Database stored functions, as shown in Example C-118 on page 430, with the results shown in Example C-119 on page 430 without specifying IMMUTABLE.
- FUJITSU Enterprise Postgres stored functions that specify IMMUTABLE for Example C-118 on page 430 are shown in Example C-120 on page 430, with the results in Example C-121 on page 431.

These examples of FUJITSU Enterprise Postgres use the **EXPLAIN** statement and output the query plan to confirm the run time. These examples show that the run time of a stored function with IMMUTABLE is shorter than the one of a stored function without IMMUTABLE, although the difference is small because it is a simple example.

These examples of FUJITSU Enterprise Postgres enable Oracle compatibility features.

► Example of Oracle Database

Example: C-116 [Oracle -PL/SQL] Stored functions

```
CREATE OR REPLACE FUNCTION create_message(msg VARCHAR2)
  RETURN VARCHAR2
AS
  ret_val VARCHAR2(256);
BEGIN
  ret_val := 'NOTICE: ' || msg;
  RETURN ret_val;
END create_message;
/
SELECT create_message('sample message') as message
FROM DUAL;
```

Example: C-117 [Oracle-Result] Stored functions

MESSAGE

 NOTICE: sample message

► Example of FUJITSU Enterprise Postgres simply converted

Example: C-118 [FUJITSU Enterprise Postgres-PL/pgSQL] Stored functions without specifying IMMUTABLE

```
CREATE OR REPLACE FUNCTION create_message(msg VARCHAR)
  RETURNS VARCHAR
AS $$
DECLARE
  ret_val VARCHAR(256);
BEGIN
  ret_val := 'NOTICE: ' || msg;
  RETURN ret_val;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
SELECT create_message('sample message') as message
FROM DUAL;
EXPLAIN(ANALYZE, BUFFERS, VERBOSE)
SELECT create_message('sample message') as message
FROM DUAL;
```

Example: C-119 [FUJITSU Enterprise Postgres- Result] Stored functions without specifying IMMUTABLE

message

 NOTICE: sample message

QUERY PLAN

 Result (cost=0.00..0.26 rows=1 width=32) (actual time=0.006..0.008 rows=1 loops=1)
 Output: create_message('sample message'::character varying)
 Planning Time: 0.012 ms
Execution Time: 0.021 ms

Example: C-120 [FUJITSU Enterprise Postgres-PL/pgSQL] Stored functions with specifying IMMUTABLE

```
CREATE OR REPLACE FUNCTION create_message(msg VARCHAR)
  RETURNS VARCHAR
  IMMUTABLE
AS $$
DECLARE
  ret_val VARCHAR(256);
BEGIN
  ret_val := 'NOTICE: ' || msg;
  RETURN ret_val;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
SELECT create_message('sample message') as message
FROM DUAL;
```

```
EXPLAIN(ANALYZE, BUFFERS, VERBOSE)
SELECT create_message('sample message') as message
FROM DUAL;
```

- ▶ Example of FUJITSU Enterprise Postgres specifying IMMUTABLE

Example: C-121 [FUJITSU Enterprise Postgres-PL/pgSQL] Stored functions with specifying IMMUTABLE

```
message
```

```
-----
NOTICE: sample message
```

```
QUERY PLAN
```

```
-----
Result (cost=0.00..0.01 rows=1 width=32) (actual time=0.001..0.002 rows=1
loops=1)
```

```
Output: 'NOTICE: sample message'::character varying
```

```
Planning Time: 0.009 ms
```

```
Execution Time: 0.009 ms
```

Stored procedures migration pattern

Oracle Database and FUJITSU Enterprise Postgres support stored procedures. However, there are differences between them, such as in syntax.

Here are the major differences.

- ▶ The equivalent property of **AUTHID** in an Oracle Database is **SECURITY** in FUJITSU Enterprise Postgres. However, the behavior is different when the property is omitted. Therefore, if **AUTHID** is omitted in an Oracle Database, specify **SECURITY DEFINER** when migrating to FUJITSU Enterprise Postgres.
- ▶ The **IN OUT** parameter mode that is used in the parameter declaration in an Oracle Database must be converted to **INOUT** in FUJITSU Enterprise Postgres.
- ▶ Specify **INOUT** instead of the **OUT** argument in the parameter declaration because **OUT** is not supported in FUJITSU Enterprise Postgres.
- ▶ Oracle Database runs stored procedures from a PL/SQL block by specifying the stored procedure name directly. When migrating to FUJITSU Enterprise Postgres, use the **CALL** statement instead.
- ▶ With Oracle Database, you can omit brackets for parameters when defining procedures without parameters and when running procedures from a PL/SQL block without specifying parameters. You cannot omit the brackets in FUJITSU Enterprise Postgres.

Example C-122 - Example C-125 on page 433 show the basic conversion method of stored procedures. These examples of FUJITSU Enterprise Postgres enable Oracle compatibility features.

Example: C-122 [Oracle -PL/SQL] Stored procedures

```
ALTER SESSION SET NLS_TIMESTAMP_FORMAT = 'YYYY-MM-DD HH24:MI:SS.FF';
CREATE TABLE user_tb1(
  user_id NUMBER UNIQUE,
  name VARCHAR2(10)
);
INSERT INTO user_tb1 VALUES(1, 'Tom');
CREATE OR REPLACE PROCEDURE login_message
```

```

IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('---- Login message ----');
END;
/
CREATE OR REPLACE PROCEDURE sample_proc(
  user_id NUMBER,
  message IN OUT VARCHAR2,
  login_time OUT TIMESTAMP
)
IS
  user_name VARCHAR2(10);
  query VARCHAR2(100) := 'SELECT name FROM user_tbl WHERE user_id = :user_id';
BEGIN
  EXECUTE IMMEDIATE query INTO user_name USING user_id;
  message := message || ' ' || user_name;
  login_time := SYSTIMESTAMP;
END;
/
DECLARE
  message VARCHAR2(100) := 'Hello';
  login_time TIMESTAMP;
BEGIN
  login_message;
  sample_proc(1, message, login_time);
  DBMS_OUTPUT.PUT_LINE(message);
  DBMS_OUTPUT.PUT_LINE('Login Time: ' || login_time);
END;
/

```

Example: C-123 [Oracle-Result] Stored procedures

```

---- Login message ----
Hello Tom
Login Time: 2021-10-21 04:17:05.207654
PL/SQL procedure successfully completed.

```

Example: C-124 [FUJITSU Enterprise Postgres-PL/pgSQL] Stored procedures

```

CREATE TABLE user_tbl(
  user_id INT UNIQUE,
  name VARCHAR(10)
);
INSERT INTO user_tbl VALUES(1, 'Tom');
CREATE OR REPLACE PROCEDURE login_message()
AS $$
BEGIN
  PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
  PERFORM DBMS_OUTPUT.PUT_LINE('---- Login message ----');
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
CREATE OR REPLACE PROCEDURE sample_proc(
  user_id INT,
  message INOUT VARCHAR,
  login_time INOUT TIMESTAMP
)

```



```

AS $$
DECLARE
    user_name VARCHAR(10);
    query VARCHAR(100) := 'SELECT name FROM user_tbl WHERE user_id = $1';
BEGIN
    EXECUTE query INTO user_name USING user_id;
    message := message || ' ' || user_name;
    login_time := statement_timestamp();
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
DO $$
DECLARE
    message VARCHAR(100) := 'Hello';
    login_time TIMESTAMP;
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    CALL login_message();
    CALL sample_proc(1, message, login_time);
    PERFORM DBMS_OUTPUT.PUT_LINE(message);
    PERFORM DBMS_OUTPUT.PUT_LINE('Login Time: ' || login_time);
END;
$$ LANGUAGE plpgsql;

```

Example: C-125 [FUJITSU Enterprise Postgres- Result] Stored procedures

```

---- Login message ----
Hello Tom
Login Time: 2021-10-21 04:17:36.280428
DO

```

Other migration patterns

This section describes the following topics:

- ▶ Migration pattern: Cursor FOR LOOP statements
- ▶ Migration pattern: EXECUTE IMMEDIATE statement
- ▶ Migration pattern: Exponentiation operator
- ▶ Migration pattern: FORALL statement

Migration pattern: Cursor FOR LOOP statements

In Oracle Database, the cursor **FOR LOOP** statement can use an implicit cursor. FUJITSU Enterprise Postgres does not support the use of implicit cursors, so you must define explicitly a variable of the RECORD type.

Example C-126 - Example C-129 on page 434 of FUJITSU Enterprise Postgres enable Oracle compatibility features.

Example: C-126 [Oracle -PL/SQL] Cursor FOR LOOP statements

```

CREATE TABLE cur_tbl(
    id NUMBER,
    val VARCHAR2(10)
);
INSERT INTO cur_tbl(id, val) (

```

```

SELECT LEVEL, 'data' || LEVEL FROM DUAL
CONNECT BY LEVEL <= 10
);
BEGIN
  FOR rec_cur IN (
    SELECT id, val FROM cur_tbl WHERE id < 3
  )
  LOOP
    DBMS_OUTPUT.PUT_LINE(
      'id=' || rec_cur.id || ', val=' || rec_cur.val);
  END LOOP;
END;
/

```

Example: C-127 [Oracle-Result] Cursor FOR LOOP statements

```

id=1, val=data1
id=2, val=data2
PL/SQL procedure successfully completed.

```

Example: C-128 [FUJITSU Enterprise Postgres-PL/pgSQL] Cursor FOR LOOP statements

```

CREATE TABLE cur_tbl(
  id INT,
  val VARCHAR(10)
);
INSERT INTO cur_tbl(id, val) (
  SELECT
    generate_series, 'data' || generate_series
  FROM generate_series(1, 10)
);
DO $$
DECLARE
  rec_cur RECORD;
BEGIN
  PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
  FOR rec_cur IN (
    SELECT id, val FROM cur_tbl WHERE id < 3
  )
  LOOP
    PERFORM DBMS_OUTPUT.PUT_LINE(
      'id=' || rec_cur.id || ', val=' || rec_cur.val);
  END LOOP;
END;
$$ LANGUAGE plpgsql;

```

Example: C-129 [FUJITSU Enterprise Postgres- Result] Cursor FOR LOOP statements

```

id=1, val=data1
id=2, val=data2
DO

```

Migration pattern: EXECUTE IMMEDIATE statement

To run dynamic SQL, Oracle Database uses the **EXECUTE IMMEDIATE** statement, and FUJITSU Enterprise Postgres uses the **EXECUTE** statement.

The basic function is the same, but there are differences, such as in syntax or other functions. The major differences are as follows.

- ▶ FUJITSU Enterprise Postgres does not support the **IMMEDIATE** keyword. When migrating from an Oracle Database, remove the keyword **IMMEDIATE** (see Example C-130 - Example C-133 on page 436).
- ▶ The way to write placeholders differs between Oracle Database and FUJITSU Enterprise Postgres. Specify \$1, \$2, and so on, when migrating to FUJITSU Enterprise Postgres.

Example: C-130 [Oracle -PL/SQL] EXECUTE IMMEDIATE statement

```

CREATE SEQUENCE user_id_seq
  START WITH 1
  INCREMENT BY 1
  NOCACHE NOCYCLE;
CREATE TABLE user_tbl(
  user_id NUMBER UNIQUE,
  name VARCHAR2(10)
);
INSERT INTO user_tbl VALUES(user_id_seq.NEXTVAL, 'Tom');
CREATE OR REPLACE PROCEDURE user_registration(
  user_name VARCHAR2
)
IS
  query VARCHAR2(256)
  := 'INSERT INTO user_tbl VALUES(user_id_seq.NEXTVAL, :user_name)';
BEGIN
  EXECUTE IMMEDIATE query USING user_name;
END;
/
CALL user_registration('Mary');
SELECT * FROM user_tbl;

```

Example: C-131 [Oracle-Result] EXECUTE IMMEDIATE statement

```

USER_ID NAME
-----
      1 Tom
      2 Mary

```

Example: C-132 [FUJITSU Enterprise Postgres-PL/pgSQL] EXECUTE IMMEDIATE statement

```

CREATE SEQUENCE user_id_seq
  START WITH 1
  INCREMENT BY 1
  NO CYCLE;
CREATE TABLE user_tbl(
  user_id BIGINT UNIQUE,
  name VARCHAR(10)
);
INSERT INTO user_tbl VALUES(nextval('user_id_seq'), 'Tom');
CREATE OR REPLACE PROCEDURE user_registration(

```

```

    user_name VARCHAR
)
AS $$
DECLARE
    query VARCHAR(256)
        := 'INSERT INTO user_tbl VALUES(nextval(''user_id_seq''), $1)';
BEGIN
    EXECUTE query USING user_name;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
CALL user_registration('Mary');
SELECT * FROM user_tbl;

```

Example: C-133 [FUJITSU Enterprise Postgres- Result] EXECUTE IMMEDIATE statement

user_id	name
1	Tom
2	Mary

Migration pattern: Exponentiation operator

To calculate exponentiation, Oracle Database uses the Exponentiation operator "**", and FUJITSU Enterprise Postgres uses the Mathematical operator "^".

Example C-134 - Example C-137 on page 437 of FUJITSU Enterprise Postgres enables Oracle compatibility features.

Example: C-134 [Oracle -PL/SQL] Exponentiation operator

```

DECLARE
    a PLS_INTEGER := 2;
    b PLS_INTEGER := 10;
    result PLS_INTEGER;
BEGIN
    result := a ** b;
    DBMS_OUTPUT.PUT_LINE(a || ' ** ' || b || ' = ' || result);
END;
/

```

Example: C-135 [Oracle-Result] Exponentiation operator

```

2 ** 10 = 1024
PL/SQL procedure successfully completed.

```

Example: C-136 [FUJITSU Enterprise Postgres-PL/pgSQL] Exponentiation operator

```

DO $$
DECLARE
    a INT := 2;
    b INT := 10;
    result INT;
BEGIN
    PERFORM DBMS_OUTPUT.SERVEROUTPUT(TRUE);
    result := a ^ b;
    PERFORM DBMS_OUTPUT.PUT_LINE(a || ' ** ' || b || ' = ' || result);
END;
$$ LANGUAGE plpgsql;

```

Example: C-137 [FUJITSU Enterprise Postgres- Result] Exponentiation operator

```
2 ** 10 = 1024
DO
```

Migration pattern: FORALL statement

The **FORALL** statement runs bulk SQL statements in Oracle Database. Alternatively, use the **FOR LOOP** statement in FUJITSU Enterprise Postgres because **FORALL** is not supported. Examples of an Oracle Database **FORALL** statement and its results are shown in Example C-138 and Example C-139. Examples of a FUJITSU Enterprise Postgres **FOR LOOP** statement are in Example C-140 and Example C-141 on page 438.

Example: C-138 [Oracle -PL/SQL] FORALL statement

```
CREATE TABLE sample_tbl(name VARCHAR2(10));
DECLARE
  TYPE name_list IS VARRAY(5) OF VARCHAR2(10);
  persons name_list := name_list('Tom', 'Mary', 'John', 'Jane', 'Alex');
BEGIN
  FORALL i IN 1..5
    INSERT INTO sample_tbl VALUES(persons(i));
END;
/
SELECT * FROM sample_tbl;
```

Example: C-139 [Oracle-Result] FORALL statement

```
PL/SQL procedure successfully completed.
NAME
-----
Tom
Mary
John
Jane
Alex
```

Example: C-140 [FUJITSU Enterprise Postgres-PL/pgSQL] FOR LOOP statement

```
CREATE TABLE sample_tbl(name VARCHAR(10));
DO $$
DECLARE
  persons VARCHAR[5]
    := '{"Tom", "Mary", "John", "Jane", "Alex"}';
BEGIN
  FOR i IN 1..5 LOOP
    INSERT INTO sample_tbl VALUES(persons[i]);
  END LOOP;
END;
$$ LANGUAGE plpgsql;
SELECT * FROM sample_tbl;
```

Example: C-141 [FUJITSU Enterprise Postgres- Result] FOR LOOP statement

DO

 name

 Tom

 Mary

 John

 Jane

 Alex

Related publications

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

IBM Redbooks

The following IBM Redbooks publications provides more information about the topics in this document. The publication that is referenced in this list might be available in softcopy only.

- ▶ *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499
- ▶ *Leveraging LinuxONE to Maximize Your Data Serving Capabilities*, SG24-8518

You can search for, view, download, or order this document and other Redbooks, Redpapers, web docs, drafts, and additional materials, at the following website:

ibm.com/redbooks

Online resources

The following websites are also relevant as further information sources:

- ▶ FUJITSU Enterprise Postgres on IBM LinuxONE
https://www.postgresql.fastware.com/fujitsu-enterprise-postgres-on-ibm-linuxone?utm_referrer=https%3A%2F%2Ffastware.com%2F

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(->Hide)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine:fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

Draft Document for Review November 21, 2024 7:17 am

8518spine.fm 441

Redbooks

Leveraging LINUXONE to Maximize Your Data Serving Capabilities

SG24-8518-01

ISBN



(0.5" spine)

0.475" <-> 0.873"

250 <-> 459 pages

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(->Hide)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine:fm still open and **File>Import>Formats the Conditional Text Settings (ONL Y1)** to the book files.



SG24-8518-01

ISBN

Printed in U.S.A.

Get connected

