Redbooks

ibm.com/redbooks

Leveraging LinuxONE to Maximize Your Data Serving Capabilities

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

Yoshimi Toyoshima

Cloud

LinuxONE

IBM

Redbooks

IBM Redbooks

Leveraging LinuxONE to Maximize Your Data Serving Capabilities

April 2022

SG24-8518-00

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

First Edition (April 2022)

This edition applies to:

- FUJITSU Software Enterprise Postgres Advanced Edition Annual Subscription License per IFL 13 for Linux on IBM Z
- FUJITSU Software Enterprise Postgres Advanced Edition Annual Subscription License per IFL 13 Operator Bundle for Kubernetes on IBM Z

Contents

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 Garage™	Redbooks (logo)  ®
IBM®	IBM Spectrum®	z/OS®
IBM Cloud®	IBM Z®	z/VM®
IBM Cloud Pak®	POWER®	z15™
IBM FlashSystem®	Redbooks®	

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.

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 enables organizations to access enterprise data and extend the business value of core data that has been collected over many years. Data modernization is any initiative or process that results in more relevant and accurate data or more efficient and timely data delivery and analyses that support both improved governance (management, science, and enforcement) and serve industry and public needs.

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 FUJITSU Enterprise Postgres on IBM LinuxONE and migrating your database to FUJITSU Enterprise Postgres on IBM LinuxONE. We present a database migration use case in 2.4, “Use cases: Migration from an Oracle Database system” on page 59.

We also describe using containers and the benefits of automation by using containers (see 3.2, “Benefits of automation when using containers” on page 99), and we describe how FUJITSU Enterprise Postgres on IBM LinuxONE can deploy databases quickly. We describe how database operations can be automated with FUJITSU Enterprise Postgres Operator. We also describe fluctuation in 3.5, “Fluctuation” on page 134.

In Chapter 4, “Application use cases: Geospatial processing” on page 155, we demonstrate geospatial processing when using PostGIS and FUJITSU Enterprise Postgres on IBM LinuxONE.

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

Chapter 5, “MongoDB as a service with Linux on IBM Z” on page 173 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 team of specialists from around the world working at IBM Redbooks, Poughkeepsie Center.

Kurt Acker is the Principal IT Architect for both Direct Systems Support (DSS) and Sine Nomine Associates (SNA). He shares design work with implementations done at events like SHARE, the VM Workshop, Hillgang, and other Linux events. This IBM® Redbooks® publication is the second on which he has worked. Kurt graduated from the Binghamton Universities Thomas J. Watson school of Engineering with a degree in Computer Engineering, which was followed by an Executive MBA from Binghamton’s School of Management.

Sam Amsavelu has worked for IBM for the past 27 years advising customers about how to build HA, reliable, and secure infrastructure architectures for their databases. He is an expert in implementing virtualization, server consolidation, and cloud technologies to help customers choose the correct platform and technologies for an optimized environment with the lowest total cost of ownership (TCO) possible. He is a subject matter expert (SME) in multiple operating systems (OSs) (IBM z/OS®, IBM z/VM®, UNIX, and multiple Linux distributions); relational databases (IBM Db2®, Oracle, PostgreSQL, and SQL Server); NoSQL databases (MongoDB); high availability and disaster recovery (HADR) solutions (Oracle RAC, Data Guard, GoldenGate, replication, and sharding); and Red Hat OpenShift solutions.

Elton de Souza has worked on the IBM Z® platform during his entire 11-year career at IBM. He leads cloud-native technology adoption on the IBM Z and LinuxONE platforms. The first half of his career was focused on Java, where he worked on using 200+ hardware instructions on IBM Z for mission-critical mobile and cloud workloads. He was one of the first technical experts for Docker and Kubernetes on IBM Z in early 2015, and since then has worked with significant IBM clients on successful adoption of cloud-native technology like Kubernetes, IBM Cloud® Private, Red Hat OpenShift and IBM Cloud Pak®, and IBM Cloud Hyper Protect Services. He has written over 30 publications; contributes to several open source projects; and leads the design and development of the CareKit SDK for Hyper Protect in partnership with Apple, which uses IBM LinuxONE based services in IBM Cloud.

Gary Evans is a Senior Manager at the Global Center of Excellence at Fujitsu Limited Software Business Unit. Gary has over 28 years of experience in software development and data technologies, and he has worked at numerous organizations, including IBM Global Services, cable and wireless (London), and the Inland Revenue Department of New Zealand before joining Fujitsu. He is an active contributor to the Australian PostgreSQL Community, and has presented at events such as FOSSASIA, PgDay Asia, OpenStack, and PgDay Down Under.

Neale Ferguson works on Linux and contributes to the Linux ecosystem. He has been involved in significant MongoDB-based projects on Linux. He holds degrees in Computer Science (B.Sc) and Cognitive Science (M.Cog.Sc). He has written and presented on many aspects of the Linux ecosystem, including HA, containers and Red Hat OpenShift, and porting and using .NET on IBM Z.

Aya Hoshino is a Digital Content and Enablement Specialist in Australia. She has 7 years of experience in communication management and liaising. She holds a degree in Economics from The University of Tokyo. Her areas of expertise include multi-cultural project management assistance, communication facilitation, and interpretation between Japanese and English-speaking teams worldwide. She has contributed to the articles introducing how FUJITSU Enterprise Postgres uses container technology, and has supported the coordination of this project for the Fujitsu Global Team.

Yuki Ishimori is a Database Software Engineer for FUJITSU Enterprise Postgres in Japan. He has 10 years of experience developing various database features designing and deploying databases in the field. His areas of expertise include database operational design and performance tuning.

Niki Kennedy is a Senior Director in the Fujitsu Software Business unit with an IT career spanning over 30 years across technical and business strategy roles. Her area of focus is to enable start-ups and organizations to adopt new technologies. She has worked for the past 5 years with customers and IBM Business Partners to transform their data strategy by using FUJITSU Enterprise Postgres. As the Global Software Business Executive for the IBM Partnership and FUJITSU Enterprise Postgres Global Business, Niki leads a diverse team that covers expert services, technical pre-sales, commercials, and sales and marketing. Niki has global experience consulting across the full range of enterprise and consumer technology, with a focus on developing collaborative and strategic relationships with clients and IBM Business Partners.

Riho Minagi is a Database Software Engineer for FUJITSU Enterprise Postgres for Kubernetes in Japan. She has 6 years of experience developing database and Kubernetes operator functions. Her area of expertise is data analytics. She has written about deploying, operating, and autoscaling databases with FUJITSU Enterprise Postgres Operator.

Daiki Mukai is a Database Software Engineer for FUJITSU Enterprise Postgres in Japan. He has 15 years of experience developing software about databases and Business Intelligence (BI) and Business Analytics (BA), and he provides technical assistance in the field. His area of expertise is database migration. He has written extensively on experience-based migration technical knowledge and SQL and PL/SQL conversion.

Colin Page is an Enterprise Architect working for Cognition Foundry, an IBM Gold Business Partner, who focuses on delivering innovative projects for start-ups, and helping visionaries create meaningful change by using emerging technologies like blockchain, machine learning, AI, and geospatial analytics, running on enterprise-class infrastructures. He is based in the United Kingdom with over 30 years of experience in retail banking and IBM Z technologies. He worked at IBM for 25 years and before working at IBM, he worked for a UK Retail Bank. His areas of expertise include PostgreSQL, IBM Db2, Linux, systems architectures, and various core banking and payment solutions. He has written several white papers and IBM Redbooks publications on data analytics and banking solutions.

Anand Subramanian is a Principal Solutions Architect for FUJITSU Enterprise Postgres at Fujitsu. Previously, he was at IBM for 11 years, where his last role was as Technical Leader for IBM LinuxONE for Confidential Computing and Secure Data Serving in the Australia and New Zealand region. He is an author, speaker, and technology thought leader with 25 years of experience in the information technology field across multiple industries, including banking and finance; telecommunications; utilities; healthcare and life sciences; government; retail; and aviation. He holds a degree in Computing and Information Systems and various certifications, including IT Architecture and a certificate in Machine Learning from Caltech. This is his second IBM Redbooks collaboration.

Kaori Suyama is a Database Technical Services Specialist at Fujitsu Limited. She is one of the initial members of Fujitsu Database Migration Factory that was established in 2017, where she took the leadership role in standardizing methodology, knowledge, and processes for database migration to FUJITSU Enterprise Postgres. She has been working on multiple database migration projects internationally, and has published a migration guidebook at PostgreSQL Enterprise Consortium (PGEcons).

Kazuhisa Tanimoto is a Database Software Engineer for FUJITSU Enterprise Postgres for Kubernetes in Japan. He has two years of experience developing cloud databases, such as Kubernetes operator functions. He developed various middleware products, including databases.

Yoshimi Toyoshima is a Database Software Engineer for FUJITSU Enterprise Postgres in Japan. She has 10 years of experience in database migration and development field. Her area of expertise is database migration. She has written extensively on technical knowledge for database migration.

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

Lydia Parziale
IBM Redbooks, Poughkeepsie Center

Robert Haimowitz
IBM Garage™ for Systems, Poughkeepsie Center

Tom Ambrosio and Bill Lamastro
IBM CPO

Nikhil Kumar Bayawat, Ichiro Eguchi, Marcelo Hauschild, Junji Kawai, Takuma Maeda, Nozomi Minami, Marcin Jastrzab, Masaki Nagao, Zeus Ng, Masaki Nishigaki, Kaori Osaka, Yuho Shiinoki, Nobuyuki Takabe, Ryohei Takahashi, Takashi Tokuda, Kazuhiro Taniguchi, Naoki Umeda, Shinya Watanabe, Kiichi Yamada,
Fujitsu

Scott Courtney, Richard Scott Coyle, Kate Stringfield, Cheyenne Wills, Margarete Ziemer
SNA

Dennis Andrews
Velocity Software

Yvon-Marie Avril, Fred Bader, Randy Blea, Chuck Brazie, Raj Datta, Philippe Deverge, George Dillard, Joe Foti, Steve Gessner, Stev Glodowski, Marty Horan, Gerard Hosch, Chen Ji, Setareh Mehrabanzad, David Morley, Kenneth Morse, Paul Novak, Gene Ong, Yves Santos, Jackson Shea, Michael Snihur, Stuart Blae Tener, John Willis, Dong Yan Yang, Simon W. Yee
IBM

John Ryan
IT Economics and Research Team

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

1

Customer value

This IBM Redbooks publication focuses on the synergy between FUJITSU Enterprise Postgres and the enterprise class IBM LinuxONE server. It describes 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.

According to industry analysts such as Gartner, many organizations embarking on digital transformation (DX) initiatives adopt a continuous modernization strategy that is 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.

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.

In this book, we expand on a specific distribution of PostgresSQL that is provided by Fujitsu that augments the capabilities of the open source edition to provide a true enterprise-grade experience and support. This distribution, which is combined with the IBM LinuxONE server, delivers a robust solution that can replace alternative monolithic or n-tier commercial databases and appliances for any workload without compromise.

1.1 FUJITSU Enterprise Postgres on IBM LinuxONE

FUJITSU Enterprise Postgres extends all the benefits of open source Postgres with enterprise-grade support, security, and unique database features, and fully 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, as shown in Figure 1-1.

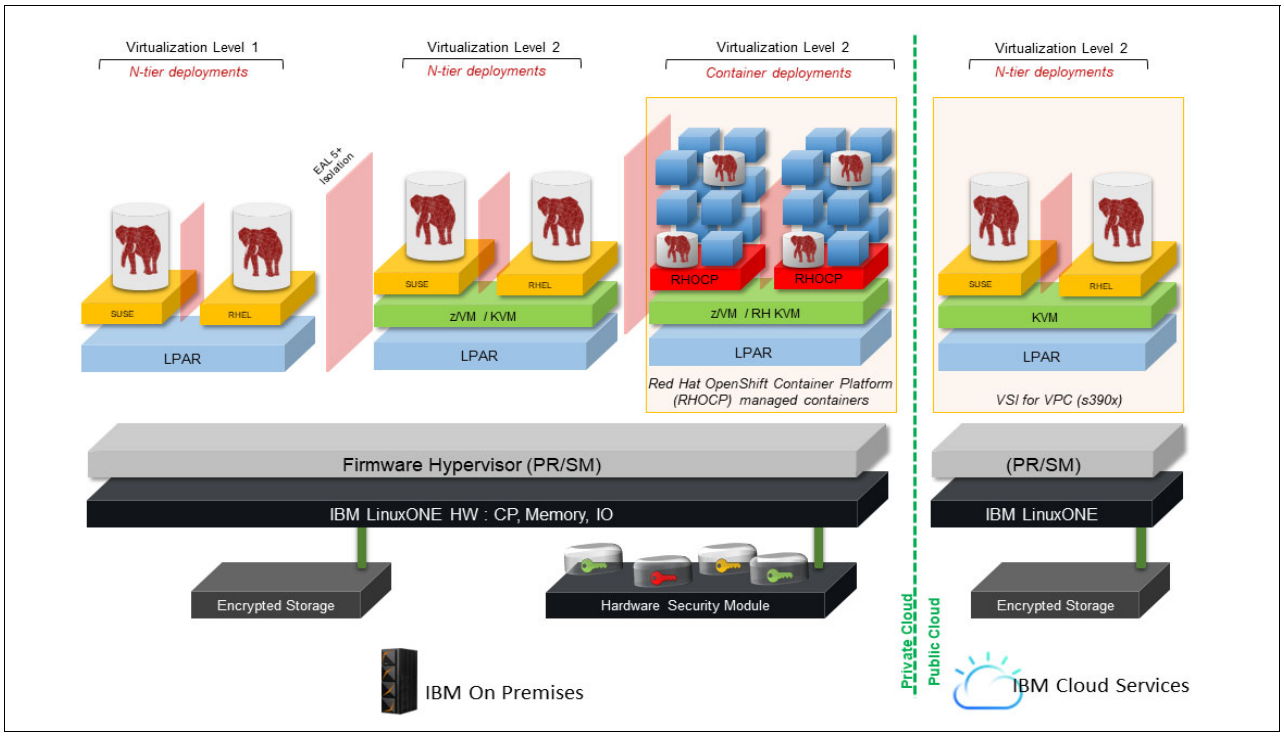

Figure 1-1 FUJITSU Enterprise Postgres on IBM LinuxONE supports multiple configurations

1.1.1 IBM LinuxONE: Virtualization Level 1

Designed for applications and databases where their environment characteristics mandate more direct access to infrastructure resources such as compute, memory, or I/O, this level of virtualization supports native deployments of Linux based databases, including Oracle, IBM Db2, and FUJITSU Enterprise Postgres.

This level of virtualization is best suited for organizations that continue to rely on monolithic (n-tier) architectures for their applications or databases and can be deployed only on on-premises LinuxONE servers. Organizations that are looking to migrate from n-tier architectures to more agile containerized and microservices deployments benefit from first migrating their n-tier environments to the LinuxONE environment.

This virtualization level enables the client to take advantage of key features such as Evaluation Assurance Level 5+ (EAL5+) partitioning and granular resource allocation capabilities (for more information, see *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499) to run modernization projects on a single platform that are close to the source systems but with full separation of environments, and to realize an acceleration toward an agile application and database architecture.

1.1.2 IBM LinuxONE: Virtualization Level 2-non-containerized workloads

Designed for applications and databases where their environment characteristics support virtualized, granular access to and sharing of infrastructure resources such as compute, memory, or I/O, this level of virtualization supports native deployments of Linux based databases, including Oracle, IBM Db2, and FUJITSU Enterprise Postgres.

Note: IBM z/VM must be used for Oracle deployments that use Virtualization Level 2 non-containerized workloads. For more information, contact your IBM customer service representative.

This level of virtualization is suited for organizations that continue to rely on monolithic (n-tier) architectures for their applications and databases or are looking to migrate from them to a server platform that is suited to run different types of architectures in parallel without resource contention or security concerns.

With IBM Cloud Infrastructure Center, an Infrastructure-as-a-Service offering, you can manage the lifecycle management for the IBM z/VM® and Red Hat® KVM-based virtual infrastructure. It delivers an industry-standard user experience for the IaaS management of containerized and non-containerized workloads.

At the time of writing, Virtualization Level 2, which is based on KVM virtualization on IBM LinuxONE, is available for consumption through the IBM public cloud under an infrastructure as a service (IaaS) subscription service in select IBM Data Centers globally.

1.1.3 IBM LinuxONE: Virtualization Level 2-containerized workloads

Virtualization Level 2 containerized workloads on IBM LinuxONE addresses application modernization requirements to host applications and databases that support containerization.

This level of virtualization provides highly granular access to shared infrastructure resources that are allocated to nominated worker nodes that can be provisioned and scaled on-demand to host fully self-contained, pre-packaged application or database environments.

In addition, these container images can be published in a reference catalog for simplified consumption and automatically deployed on IBM LinuxONE through the Red Hat OpenShift orchestration software. For more information about this topic, see *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

With IBM Cloud Infrastructure Center, an Infrastructure-as-a-Service offering, you can manage the lifecycle management for the IBM z/VM® and Red Hat® KVM-based virtual infrastructure. It delivers an industry-standard user experience for the IaaS management of containerized and non-containerized workloads.

At the time of writing, only the FUJITSU Enterprise Postgres database, which is built on multi-arch Kubernetes containers, is available as a fully supported Red Hat OpenShift Operator (level 5) for the IBM LinuxONE platform for on-premises deployments.

Because these containers support multiple server CPU architectures, these containers can be deployed on IBM LinuxONE, on IBM POWER®, on any x86 platform, or in any cloud, which makes the databases and applications in these containers truly portable and allows for a true hybrid multi-cloud architecture that uses the hardware or cloud that best meets the organizational requirements.

1.2 Application modernization with FUJITSU Enterprise Postgres

There are many reasons that an organization might choose to modernize their applications, such as:

- *Seeking ways to rationalize business operating expenditures* by migrating to alternative enterprise solutions with simplified and lower-cost software licensing models without compromising on vendor support capabilities.

  Most database vendors require customers to procure their software on an initial base software license metric and then require independent licenses for enterprise features such as encryption, compression, and high availability (HA). License requirements are calculated by multiple metrics that depend on the platform, including cores processors and variable processor value units.

  In addition to the initial software license purchases, customers must subscribe to annual support for each the individual software components, which typically increases each year, to retain vendor support beyond the first year of purchase. However, FUJITSU Enterprise Postgres is licensed based on a simple, linear, and annual support subscription license metric that includes all the enterprise capabilities of FUJITSU Enterprise Postgres. On the IBM LinuxONE server, support is calculated based on the number of IBM Integrated Facility for Linux (IFL) cores that FUJITSU Enterprise Postgres is using. If FUJITSU Enterprise Postgres is running in a containerized or virtualized logical partition (LPAR), then the support subscription is calculated based on the number of IFLs that are assigned to that LPAR.

  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.

  An overview of the licensing metrics that apply depending on the configuration, including HA and varying levels of virtualization, can be found in Chapter 1, “Customer value”, in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499. For more information about hardware and software licensing, including help with a comparative total cost of ownership (TCO) analysis for an existing deployment, contact your IBM or Fujitsu customer service representative.

- *Moving from a traditional monolithic or n-tier application architecture through containerization* for portability, scalability, manageability, and flexibility of deployment, which are essential when adopting agile development methodologies and CI/CD to accelerate application modernization.

  Because the containerization capabilities of FUJITSU Enterprise Postgres on the IBM LinuxONE server that are based on the Red Hat OpenShift Operator model were covered in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499, the following chapters of this document focus on the benefits of leveraging containerized FUJITSU Enterprise Postgres for business transformation initiatives.

- *Meeting enterprise multi-cloud and multi-architecture requirements* with ease. To reap the benefits of a cloud consumption model, customers must have a solution that has the inherent flexibility to be deployed in any architecture that best serves the enterprise needs. Also, different cloud vendors have different services to meet specific customer needs and can provide value when adopted in a fit-for-purpose approach to hybrid cloud.

As a truly multi-cloud database offering, FUJITSU Enterprise Postgres delivers the necessary flexibility of tapping in to the value propositions of different clouds while allowing customers to implement a “single” enterprise database standard across their hybrid multi-cloud environment.

1.3 FUJITSU Enterprise Postgres benefits

FUJITSU Enterprise Postgres 13 with the PostGIS extension running on IBM LinuxONE provides benefits to running geospatial processing.

Key performance features

Here is a list of some of the key performance features:

- In-column scalar capability accelerates, through parallel processing, the complex geospatial SQL statements that use inner joins on multi-million record reference tables.
- High-performance data load for updating frequently changing environmental data, such as weather (for example, Climatology Lab).
- IBM LinuxONE Enterprise Data Compression (IBM zEDC) is available on the IBM z15™ server. This feature is good for large data ingestion processes. For example, the GBIF Bird species data alone has 782 million records and is 1.7 TB.
- Low or near-zero latency that is driven by internal connectivity allows true multi-chip processing within the chip module.

High availability

Some of the HA features include the following items:

- Clustered application instances support many self-serve clients. The database is supported by the Fujitsu PostgreSQL Connection Manager function for a seamless transfer of transactions across different database instances (dynamic routing).
- With Fujitsu database log mirroring for remote database synchronization, a duplicate instance can store the same data asynchronously that is fed through data loading procedures.
- Security: Using the Fujitsu PostgreSQL Transparent Data Encryption (TDE) capability and 256-bit Payment Card Industry Data Security Standard (PCI-DSS)-compliant encryption.
- Hardware accelerated (hardware security module (HSM)) encryption that uses the IBM LinuxONE CryptoExpress adapters.
- Key management storage and execution by using firmware-based processing.
- The data masking feature of FUJITSU Enterprise Postgres obfuscates commercially sensitive data (important in both our projects) when sharing access to third parties.
- LinuxONE EAL4+ and FIPS104-2 Level 4 certification ensures workload isolation across workload instances in a shared service platform.

Operations

The performance of operations is enhanced by using the following features:

- IBM Hyper Protect Virtual Servers support secure enclave processing and protect data and access control from unauthorized users. The system admin retains access and control of the underlying operational platform for maintenance tasks, but they cannot access a client's data.
- IBM z/VM DirMaint and Performance Toolkit provide management services for Linux guests.

Within the scope of this book, we focus on those multi-cloud architectures that are based on the IBM LinuxONE server on-premises and the IBM Cloud. For more information about all supported multi-cloud reference architectures and various internal and externally published insights[1] about how FUJITSU Enterprise Postgres accelerates hybrid cloud and application modernization journeys, contact your Fujitsu customer service representative.

[1] https://futurumresearch.com/research-notes/FUJITSU-enterprise-postgres-provides-the-security-and-containerization-keys-to-hybrid-cloud-success/

2

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.

2.1 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 2-1 shows examples of open source software that is used for data handling in enterprise systems.

Table 2-1 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.

2.2 Key considerations for database migration

As mentioned in 2.1, “Increasing demand for database migration” on page 8, 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 2.2.1, “Business continuity” on page 11

    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 2.2.2, “Mitigating security threats” on page 13

    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 2-1 shows one of the HA, highly secure architecture implementations of FUJITSU Enterprise Postgres on LinuxONE.

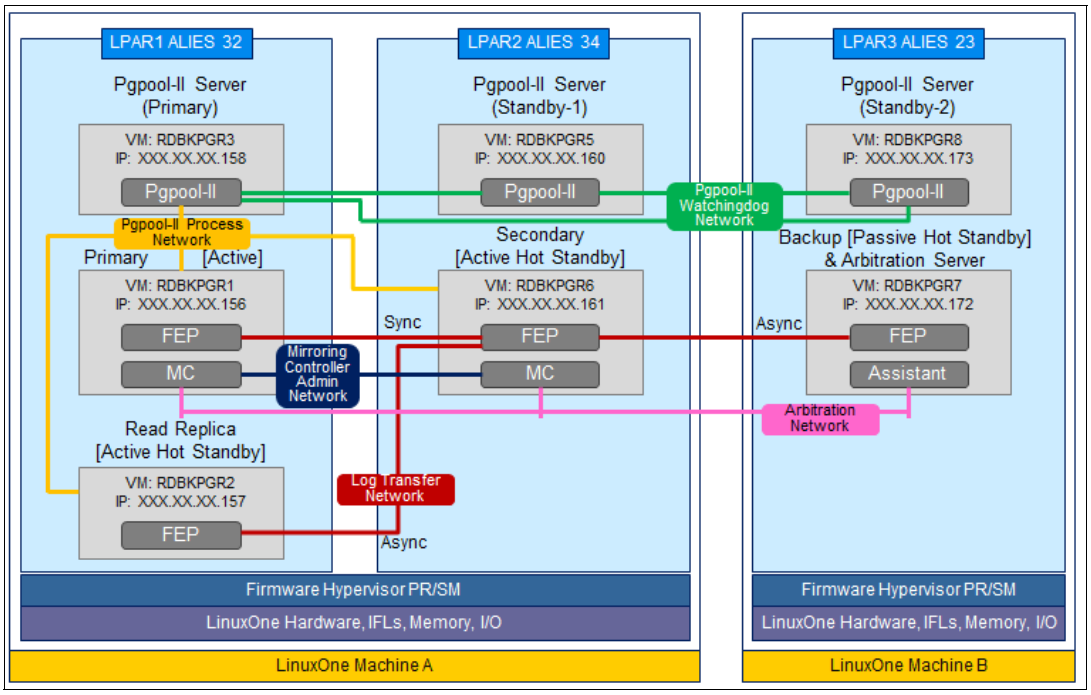

Figure 2-1 Database configuration for an enterprise system with IBM LinuxONE and FUJITSU Enterprise Postgres

Note: For more information about implementing the architecture that is shown in Figure 2-1, 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 2.2.3, “SQL performance tuning” on page 17

    This section provides an introduction to SQL tuning to resolve PostgreSQL performance challenges that organizations often face during enterprise system migration.

  – Section 2.2.4, “Optimizing database migration costs” on page 25

    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.

2.2.1 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 2-2 shows the HA architectures for Oracle RAC and FUJITSU Enterprise Postgres.

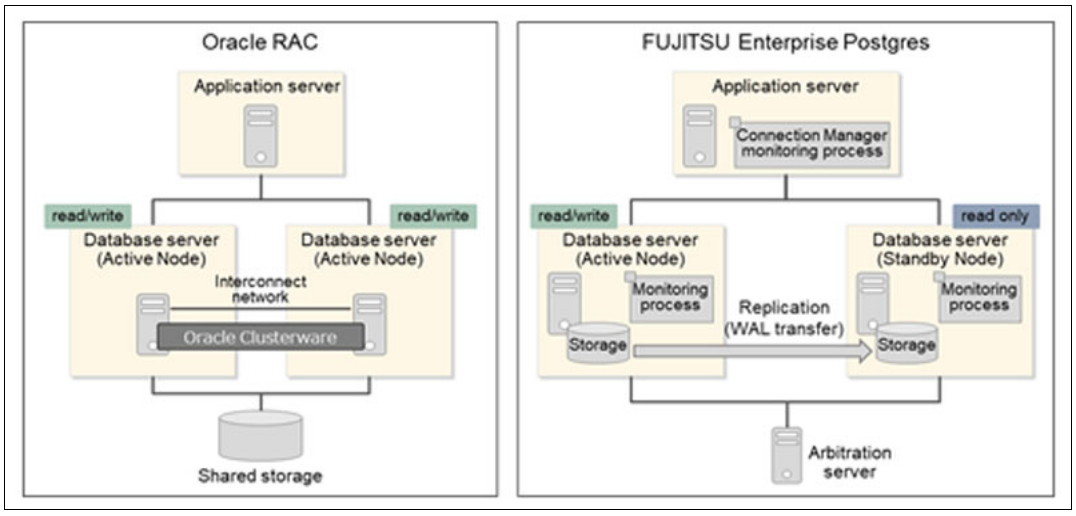

Figure 2-2 HA architectures for Oracle RAC and FUJITSU Enterprise Postgres

Despite the differences in storage and synchronization, similarities can be seen in Figure 2-2 on page 11. 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 2-3 shows how Database Multiplexing works.

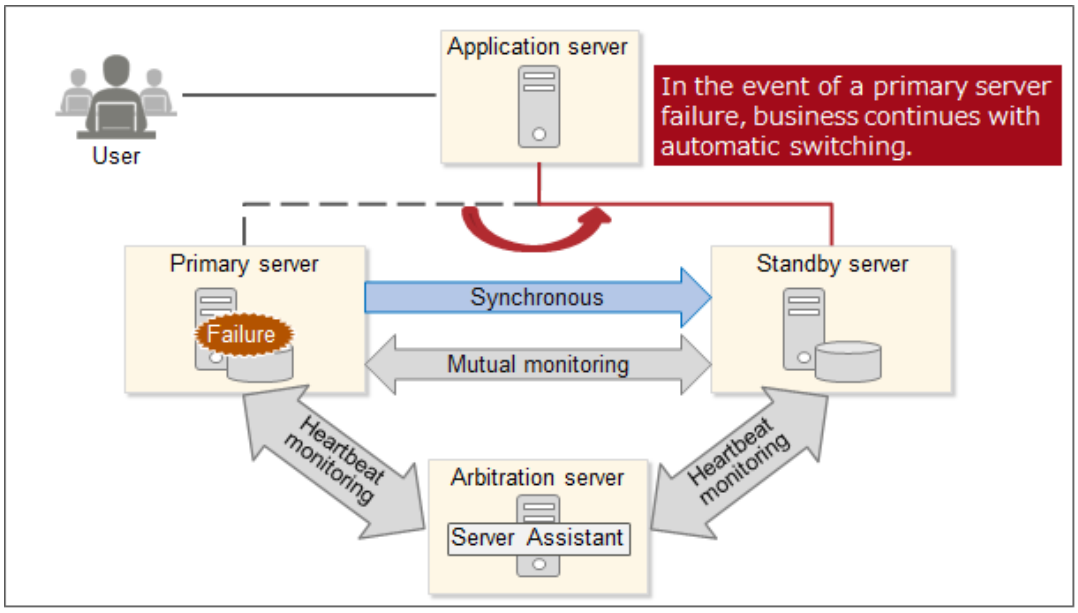

Figure 2-3 FUJITSU Enterprise Postgres Database Multiplexing

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 2-4 shows the Connection Manager processes.

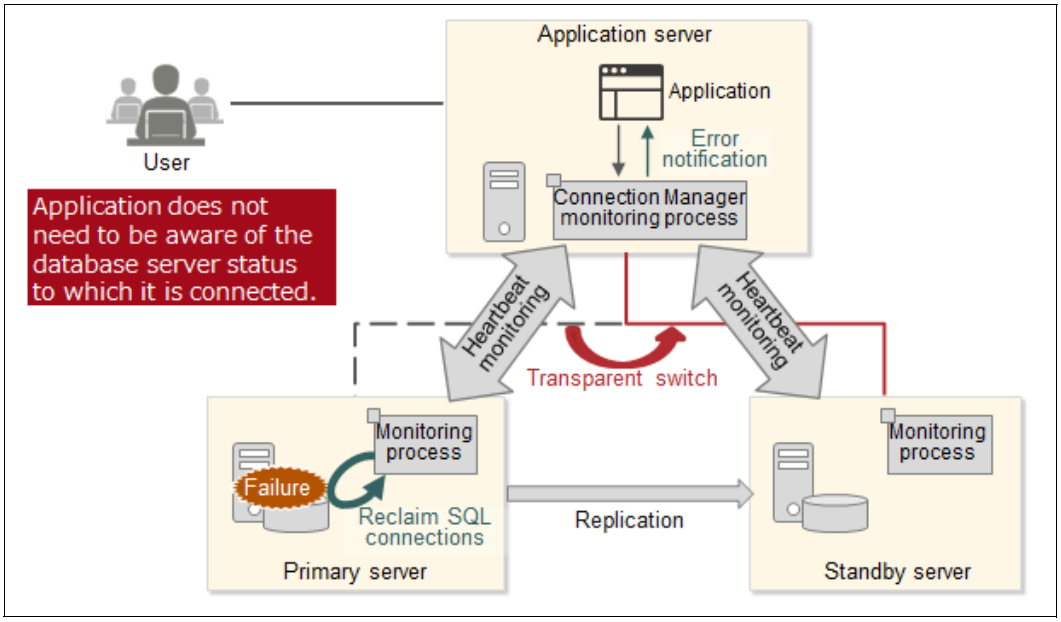

Figure 2-4 Connection Manager

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”

2.2.2 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 2-5 on page 15 shows the FUJITSU Enterprise Postgres Transparent Data Encryption processes.

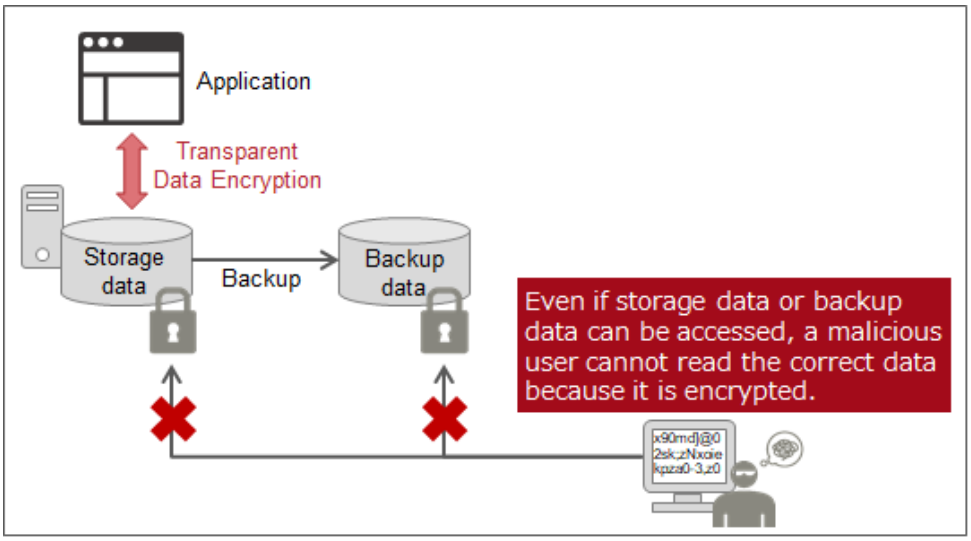

Figure 2-5 FUJITSU Enterprise Postgres Transparent Data Encryption

– 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 2-6.

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.

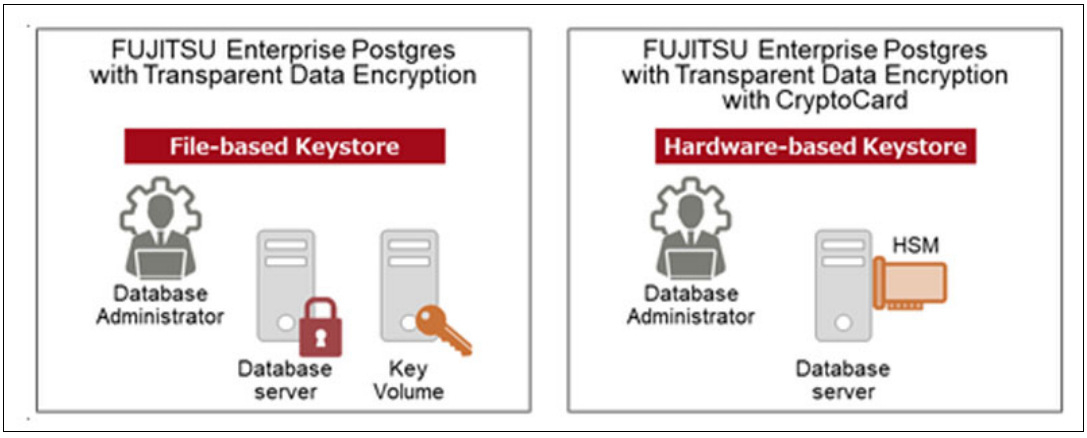

Figure 2-6 File-based and hardware-based keystore management options in Fujitsu TDE

- 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 2-7 shows a Data Masking use case for test data management.

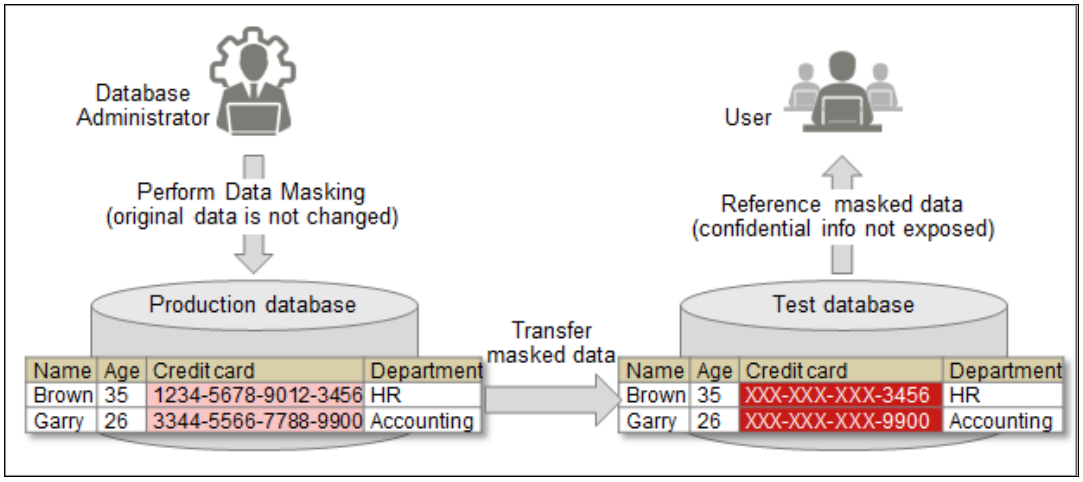

Figure 2-7 Test data management use case for data masking

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 2-8 on page 17 shows an example of the Audit Logging process.

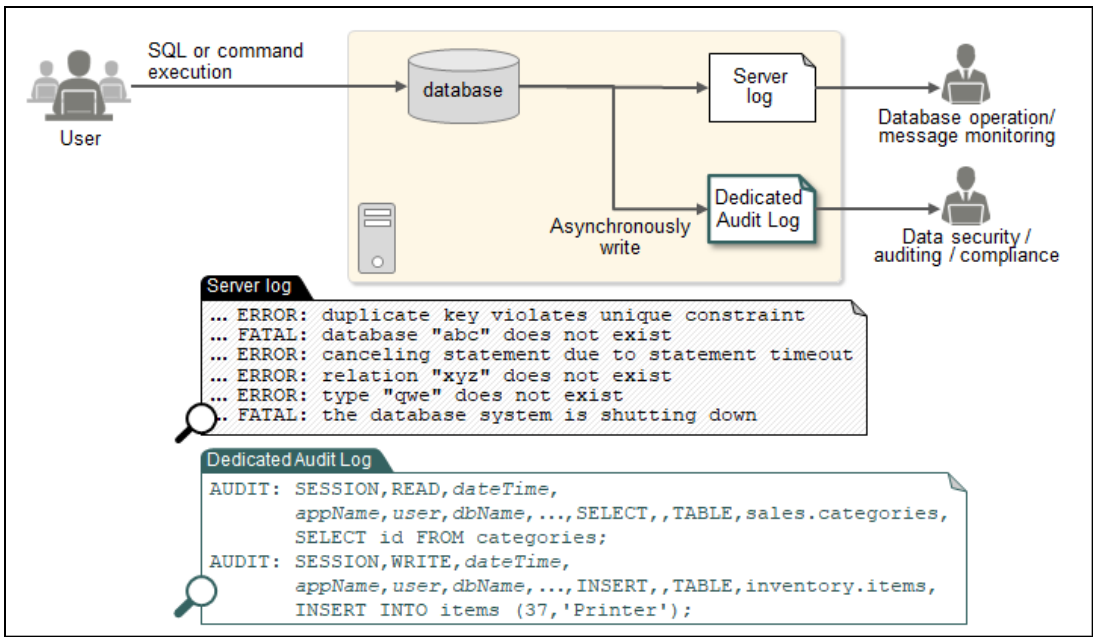

Figure 2-8 FUJITSU Enterprise Postgres Audit Logging

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.

2.2.3 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 2-25 on page 56, 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 2-1 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 2-1 Creating partitions

```
[fsepuser@rdbkpgr1 ~]$ 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 2-2 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 2-2 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 2-2 is shown in Example 2-3 on page 19.

Example 2-3 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 2-4.

Example 2-4 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 2-4 on page 19, 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 2-3 on page 19, 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 2-1 on page 18 and table `t2` in Example 2-5.

Example 2-5 Definition of table t2

```
[fsepuser@rdbkpgr1 ~]$ 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 2-6. 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 2-6 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 2-7, all partitions are accessed and searched, and the execution time is 17,907 ms.

Example 2-7 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 2-6 on page 21, it is possible to enable partition pruning by adding the search condition `id = 'ID1'` in the main query, as shown in Example 2-8.

Example 2-8 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 2-8 is shown in Example 2-9. 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 2-9 Use case 1: Query plan after tuning with partition pruning enabled

```
[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 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 2-10. 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 2-10 Use case 2: SQL before tuning

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id AND t2.value2 = 11;
```

Therefore, as shown in Example 2-11, all partitions are accessed, and the execution time is 19,545 ms.

Example 2-11 Use case 2: Query plan before tuning

```
[fsepuser@rdbkpgr1 ~]$ 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 2-5 on page 20. 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 2-12.

Example 2-12 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 2-13, 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 2-13 Use case 2: Query plan after tuning with partition pruning enabled

```
[fsepuser@rdbkpgr1 ~]$ 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@rdbkpgr1 ~]$ 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 2.3.2, “Performance tuning tips” on page 54.

2.2.4 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 2.3.1, “Experience-based migration technical knowledge” on page 27.

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.

2.3 Success-focused migration methodology

As described in 2.2, “Key considerations for database migration” on page 9, 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

2.3.1 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 2-9.

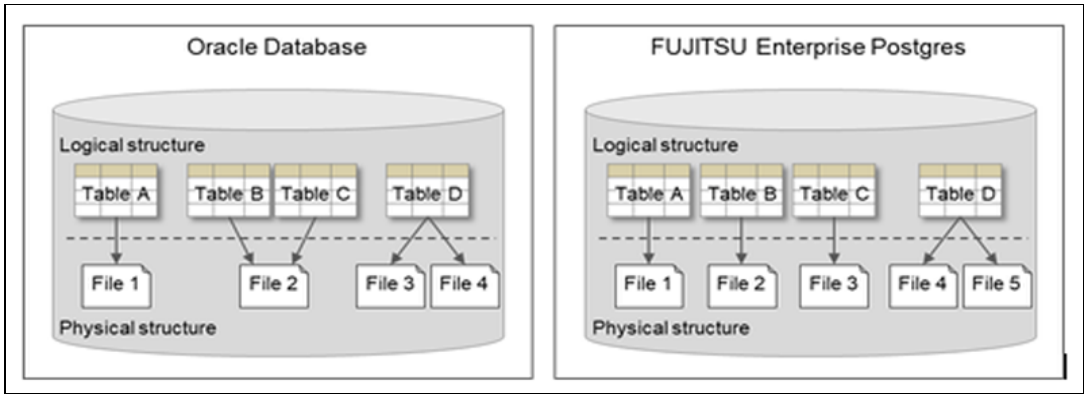

Figure 2-9 File structure of tables

Table 2-2 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 2-2 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: ► 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 stores table data as follows. ► 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 2-3 compares concurrency control between Oracle Database and FUJITSU Enterprise Postgres.

Table 2-3 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 2-4 shows the comparison of transaction processing between Oracle and FUJITSU Enterprise Postgres.

Table 2-4 Transaction processing comparison: Oracle Database and FUJITSU Enterprise Postgres

Oracle Database	FUJITSU Enterprise Postgres
Auto commit: ► A Data Definition Language (DDL) statement commits data automatically. ► A DML statement does not commit data automatically.	Auto commit: ► 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 2-5 shows a comparison of locking between an Oracle Database and a FUJITSU Enterprise Postgres database.

Table 2-5 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: ► 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: ► 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 2-6 shows a side-by-side comparison of Oracle Database and FUJITSU Enterprise Postgres data storage capacity expansion.

Table 2-6 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 2-7, which is key in subsequent tables to indicate the migration complexity level for each task.

Table 2-7 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 2-8.

Table 2-8 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 2-9.

Table 2-9 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 2-10.

Table 2-10 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 2-11 lists the operational tasks and their migration complexity levels.

Table 2-11 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 2-12 lists the migration complexity level of security features.

Table 2-12 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 2-13.

Table 2-13 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	Embedded SQL (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 2-14.

Table 2-14 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 31.

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 36 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 2-10.

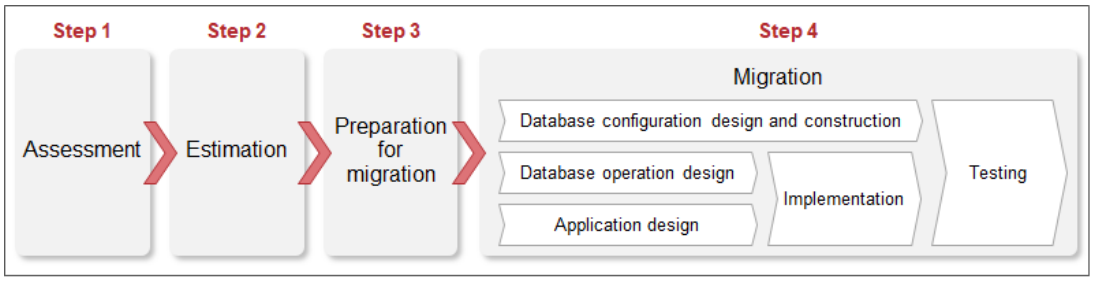

Figure 2-10 Migration process overview

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 2-10 on page 40:

- 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 43, 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 2-15.

Table 2-15 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: ► 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: ► 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: ► 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: ► JDBC. ► ODBC. ► OCI. ► Pro*C.
Data	To identify how long data migration will take, assess the following points: ► 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: ► Database monitoring. ► Authentication. ► Backup and recovery. ► HA and high reliability.
Performance requirements	To identify the system performance requirements, assess the following points: ► 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 2-16 shows two types of general database architecture. When designing, pay attention to the differences, especially for HA systems.

Table 2-16 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 2-17 on page 45.

Table 2-17 Key operational requirements

Operational requirement	Reference
Audit Logging	See 4.5, “Audit Logging” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.
Authentication	See 4.6, “Authentication” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.
HA and high reliability	See Chapter 5, “High availability and high reliability architectures” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.
Backup and recovery	See 9.1, “Backup and recovery overview” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.
Database monitoring	See 9.4, “Monitoring” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.
Database version upgrade	See Appendix A. “Version upgrade guide” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.
Database patching	See Appendix C. “Patching guide” in *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, 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 2-18 shows the key considerations of SQL migration. Table 2-19 on page 46 shows DDL statements. Table 2-20 on page 47 shows DML statements. Table 2-21 on page 48 and Table 2-22 on page 49 show functions. Table 2-22 on page 49 shows other types of SQL statements.

Note: Appendix A, “Converting SQL and PL/SQL to FUJITSU Enterprise Postgres SQL and PL/pgSQL” on page 209 provides specific examples of SQL migrations that are frequently used in Oracle Database.

Table 2-18 SQL data types in database applications

Data types	Description
Representing Character Data	► 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	► 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 2-19 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. ► 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: ► 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: ► 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 `pg_hba.conf`.
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 2-20 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: ► 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 2-21 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. ► 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 Enter-prise Postgres. However, conversion must account for the differences such as syntax so that it can be run on FUITSU Enterprise Postgres.

Table 2-22 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: ► 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. ► **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. ► '^=' 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 2-23 shows key considerations when migrating from Oracle Database PL/SQL to FUJITSU Enterprise Postgres PL/pgSQL.

Table 2-23 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 2-18 on page 45. 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.
	Transactionprocessing 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 52) 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 52. 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 2.3.2, “Performance tuning tips” on page 54.

– 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.

2.3.2 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 2-24 on page 55
  – Table 2-25 on page 56
  – Table 2-26 on page 56

- 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 2-27 on page 56
  - Table 2-28 on page 57
  - Table 2-29 on page 57

- 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 2-30 on page 58
  - Table 2-31 on page 58
  - Table 2-32 on page 58

Table 2-24 Organizing table and index data and updating statistics

Task	Description
Tuning goal	Organize table and index data and update statistics.
Tuning objectives	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	Consider setting up and implementing the following tasks regularly: 1. Preventing data files from enlarging. – 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	In the database configuration design and construction process, consider including settings for performing autovacuuming to attend to VACUUM and ANALYZE concerns. **REINDEX** should be included in the database operations design to be implemented during maintenance work.

Table 2-25 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 likeliness that a better execution plan is selected.
Tuning method	In `postgresql.conf`, 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 2-26 Accelerating SQL execution with resource partitioning and avoiding locks

Task	Description
Tuning goal	Accelerate SQL execution with resource splitting and lock avoidance.
Tuning objectives	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	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	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 2-27 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 `shared_buffer` and `wal_buffers`, and parameters that re related to WAL and checkpoints.
Tuning method	Decide the values of various buffer sizes, including `shared_buffer`, `wal_buffers`, `checkpoint_timeout`, and `max_wal_size` 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 2-28 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	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	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 2-29 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 2-30 Leveraging indexes

Task	Description
Tuning goal	Leveraging indexes.
Tuning objectives	1. Replace sort and scan processes with indexes to improve SQL performance. 2. Improve performance by using indexes more efficiently.
Tuning method	1. Replace processes: – Use index scans instead of the sort process. – Use the Index Only Scan search method. 2. Improve efficiency by using indexes: – 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 2-31 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 `pg_hint_plan` as necessary for better control. Alternatively, stabilize performance by fixing statistics with `pg_dbms_stats` so that the execution plan does not change.
Process	Database configuration design and construction, and application design.
Activities	Consider the `pg_hint_plan` and `pg_dbms_stats` settings during database configuration design and construction and application design.

Table 2-32 Optimizing iterations and communication processing

Task	Description
Tuning goal	Optimize iterations and communication processing.
Tuning objectives	1. Improve performance by caching iterations. 2. Reduce the data amount and the number of times of communication.
Tuning method	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.

2.4 Use cases: Migration from an Oracle Database system

This section explains the target system architecture (see 2.4.1, “Target system architecture” on page 59) 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.

2.4.1 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 2-11.

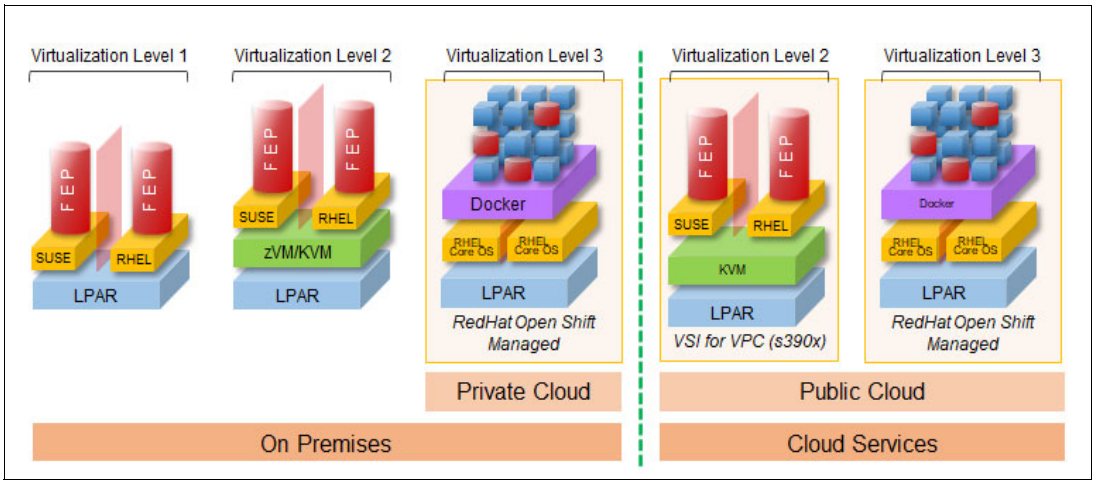

Figure 2-11 Target system architecture

- 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.

2.4.2 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 2.4.3, “Migration scenario for large-scale legacy systems” on page 62 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 2.4.4, “Migration scenario for small and medium-scale systems” on page 75 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.

2.4.3 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.

- 2.2.1, “Business continuity” on page 11
- 2.2.2, “Mitigating security threats” on page 13
- 2.2.3, “SQL performance tuning” on page 17

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 2-14.

Example 2-14 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 2.3.1, “Experience-based migration technical knowledge” on page 27. 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 2-15.

   Example 2-15 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 2-16.

   Example 2-16 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 (10000000.0%) 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 65 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 65 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 2-17.

   Example 2-17 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 65 into FUJITSU Enterprise Postgres.

   The table data that was retrieved in step 2 on page 65 is inserted into FUJITSU Enterprise Postgres by using the **psql** utility to run the script, as shown in Example 2-18.

   Example 2-18 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 2-19) in FUJITSU Enterprise Postgres by using the **psql** utility to define foreign key constraints and indexes.

   Example 2-19 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 2-20.

Example 2-20 Updating statistics by using ANALYZE

```
[fsepuser@rdbkpgr1 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 2.3.1, “Experience-based migration technical knowledge” on page 27. 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 67.

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 2-21.

Example 2-21 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 2-21 on page 67, 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 2-33 shows the differences between the two procedures.

   Table 2-33 The syntax of CREATE PROCEDURE

Oracle Database	FUJITSU Enterprise Postgres

CREATE OR REPLACE PROCEDURE
daily_sales_calc (…)
IS
  …
BEGIN
  …
END;
	CREATE OR REPLACE PROCEDURE
	daily_sales_calc (…)
	LANGUAGE plpgsql
	AS $$
	DECLARE
	  …
	BEGIN
	  …
	END;
	/

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 2-34 on page 69 shows the differences between the two conversions.

Table 2-34 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 2-35.

Table 2-35 Changing the definition of the record type

Oracle Database	FUJITSU Enterprise Postgres

TYPE daily_sales_rec IS RECORD (i_id NUMBER(6, 0), quantity CHAR(10));	CREATE TYPE daily_sales_rec AS?
	(
	  i_id int,
	  quantity char(10)
	);

4. Change the definition of **CURSOR**.

   PL/SQL and PL/pgSQL define **CURSOR** differently, so you must change the definition. Table 2-36 shows the differences between the two record type definitions.

Table 2-36 Changing the definition of the record type

Oracle Database	FUJITSU Enterprise Postgres
CURSOR c1 IS SELECT …	c1 CURSOR FOR SELECT …

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 2-37 shows the differences between the two platforms.

Table 2-37 Example of converting the function

Oracle Database	FUJITSU Enterprise Postgres
TRUNC(calc_day, 'DD');	date_trunc('day', calc_day);

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 2-38.

Table 2-38 How to handle binary data

Oracle Database	FUJITSU Enterprise Postgres
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);	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'));

Following these steps facilitates the migration of the procedure from PL/SQL to PL/pgSQL, as shown in Example 2-22.

Example 2-22 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 2-23 displays the number of orders and sales amounts for each item that was ordered on the date that is specified in the argument.

Example 2-23 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 2-23 on page 71) 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 2-39).

   Table 2-39 The syntax of CREATE FUNCTION

Oracle Database	FUJITSU Enterprise Postgres

CREATE OR REPLACE FUNCTION
daily_sales_data (…)
RETURN …
IS
  …
BEGIN
  …
END;
/
	CREATE OR REPLACE FUNCTION
	daily_sales_data (…)
	RETURNS …
	AS $$
	DECLARE
	  …
	BEGIN
	  …
	END;
	$$ LANGUAGE plpgsql;

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 2-40).

   Table 2-40 Migrating the data type

Oracle Database	FUJITSU Enterprise Postgres
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	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 $$

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 2-41).

   Table 2-41 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 69.

5. Add explicit type conversion (Table 2-42).

   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 2-42 Adding an explicit type conversion

Oracle Database	FUJITSU Enterprise Postgres
`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`	`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`

6. Convert the outer join by using (+) to the SQL standard description (Table 2-43).

   Oracle Database supports (+) to perform an outer join. FUJITSU Enterprise Postgres uses **`LEFT JOIN`** to achieve the same result.

   Table 2-43 Converting an outer join that is specified as (+)

Oracle Database	FUJITSU Enterprise Postgres
`FROM item i, daily_sales s WHERE i.i_id = s.i_id(+)`	`FROM item i LEFT JOIN daily_sales s ON i.i_id = s.i_id`

7. Convert the function (Table 2-44).

   Convert functions as shown in step 5 on page 69.

   Table 2-44 Example of converting functions

Oracle Database	FUJITSU Enterprise Postgres
`TRUNC(calc_day, 'DD');`	`date_trunc('day', calc_day);`
`NVL(s.total_quantity, 0)`	`coalesce(s.total_quantity::numeric, 0::numeric)`

8. Change the syntax for returning records (Table 2-45).

   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 2-45 Change how to return the record data

Oracle Database	FUJITSU Enterprise Postgres

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;
/
	CREATE OR REPLACE FUNCTION
	daily_sales_data (…)
	RETURNS SETOF table_daily_sales_rec
	AS $$
	DECLARE
	  …
	BEGIN
	  …
	  RETURN NEXT rec;
	  …
	END;
	$$ LANGUAGE plpgsql;

Following these steps facilitates the migration of the function from PL/SQL to PL/pgSQL, as shown in Example 2-24.

Example 2-24 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;
```

2.4.4 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.

Data protection is ensured in the container environment, as shown in Figure 2-12.

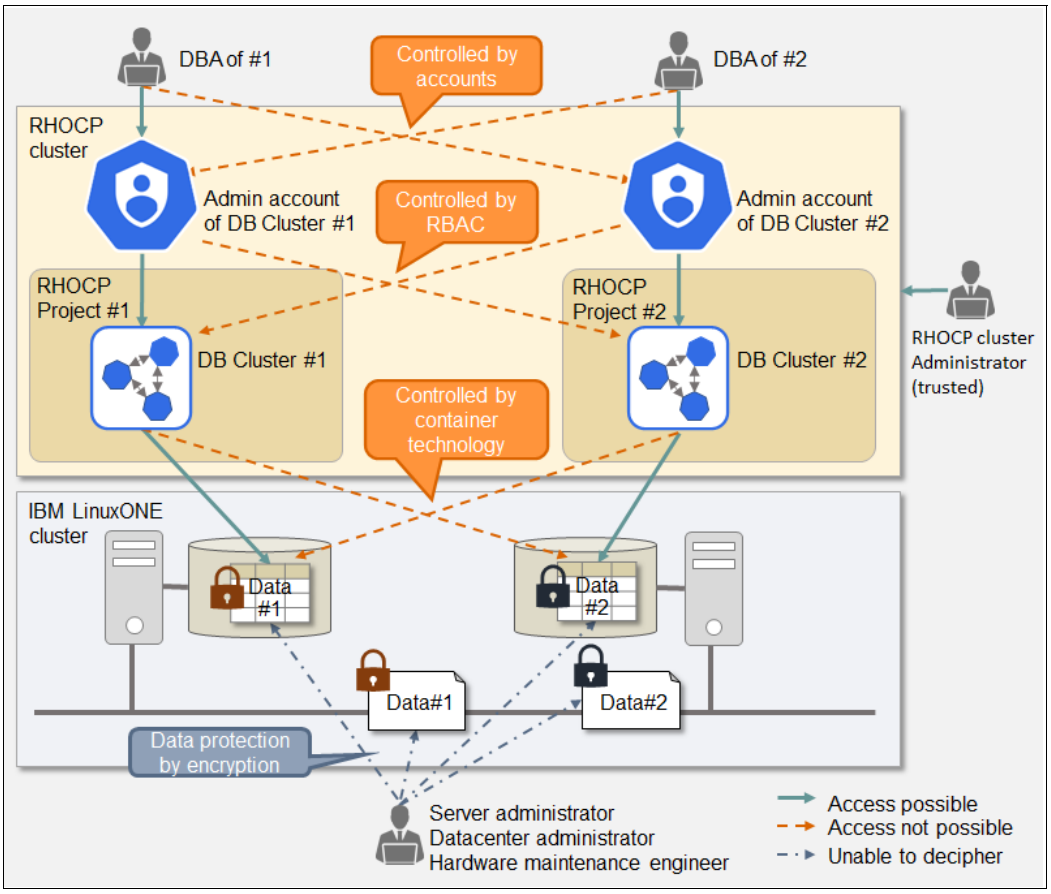

Figure 2-12 Access restriction and encryption for data protection with RHOCP and FUJITSU Enterprise Postgres

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 2-13. 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.

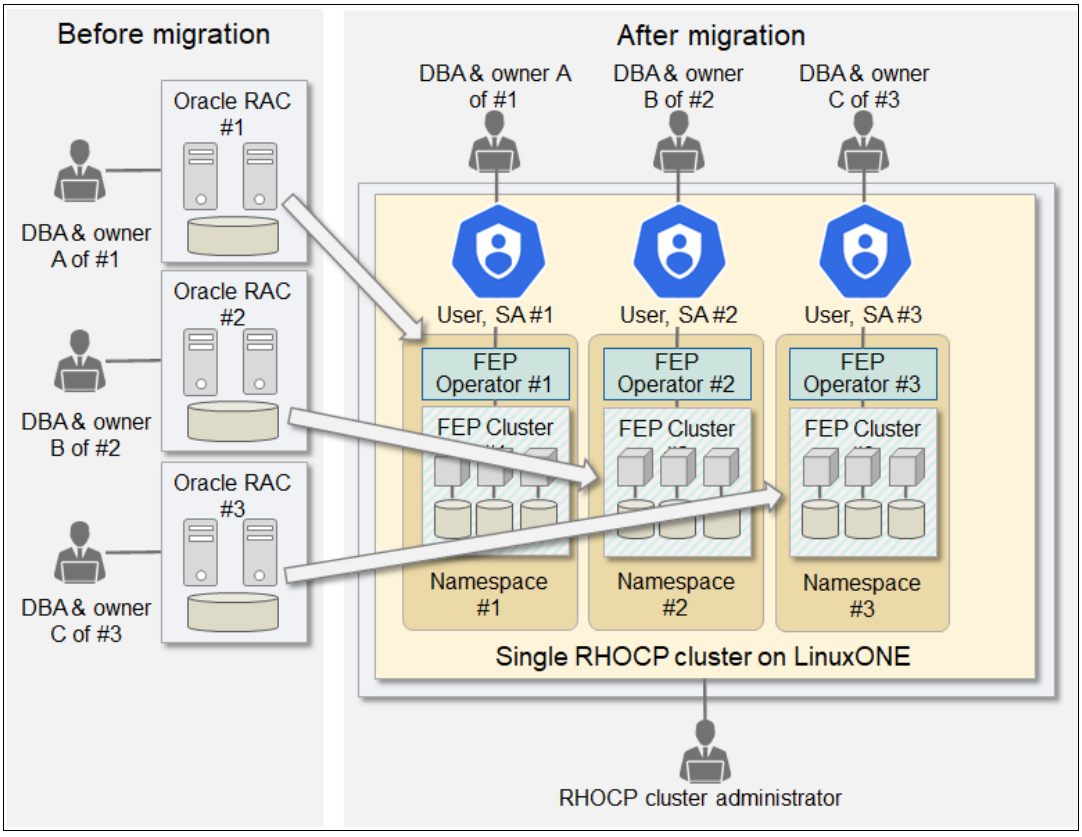

Figure 2-13 Migrating to an RHOCP environment model

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 3, “Leveraging containers” on page 97.

Steps to migrate to an RHOCP environment

The process for migration is illustrated in Figure 2-14.

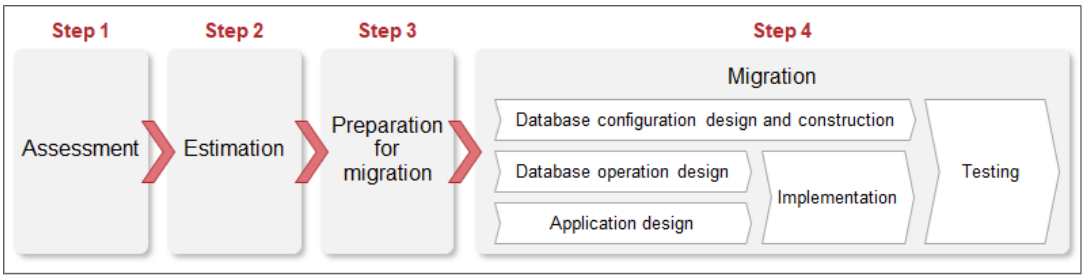

Figure 2-14 Migration process overview

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 2.3.1, “Experience-based migration technical knowledge” on page 27. For more information about migration to FUJITSU Enterprise Postgres, see 2.4.3, “Migration scenario for large-scale legacy systems” on page 62.

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 81.
  - 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 3.4, “Operation” on page 106 and 3.5, “Fluctuation” on page 134.
- 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 RHOCP environment, so ensure that the applications work in a failover.

For more information about confirming the settings, see 3.4, “Operation” on page 106.

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 85 - 19 on page 91 in “Deploying communication path encryption by using MTLS” on page 85. Then, when setting the parameters for step 20 on page 95, include the parameter changes that are described in step 3 on page 82 in “Deploying with database encryption by using TDE” on page 82. Complete the final steps in “Deploying communication path encryption by using MTLS” on page 85.

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 2-15.

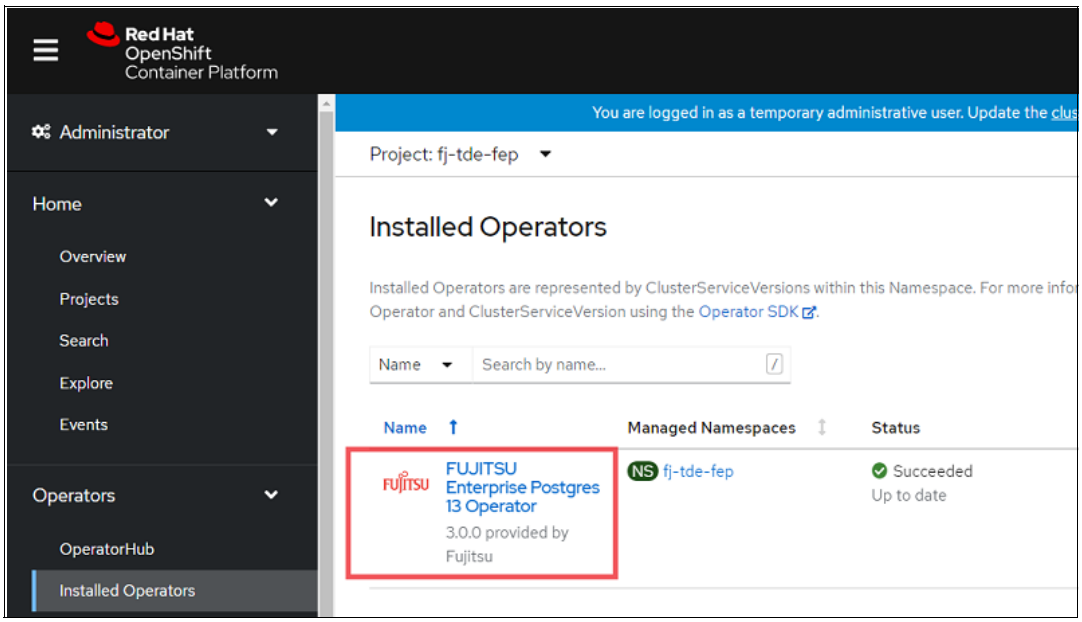

Figure 2-15 Installed Operators

3. In the Operator window, select **Create Instance**, as shown in Figure 2-16.

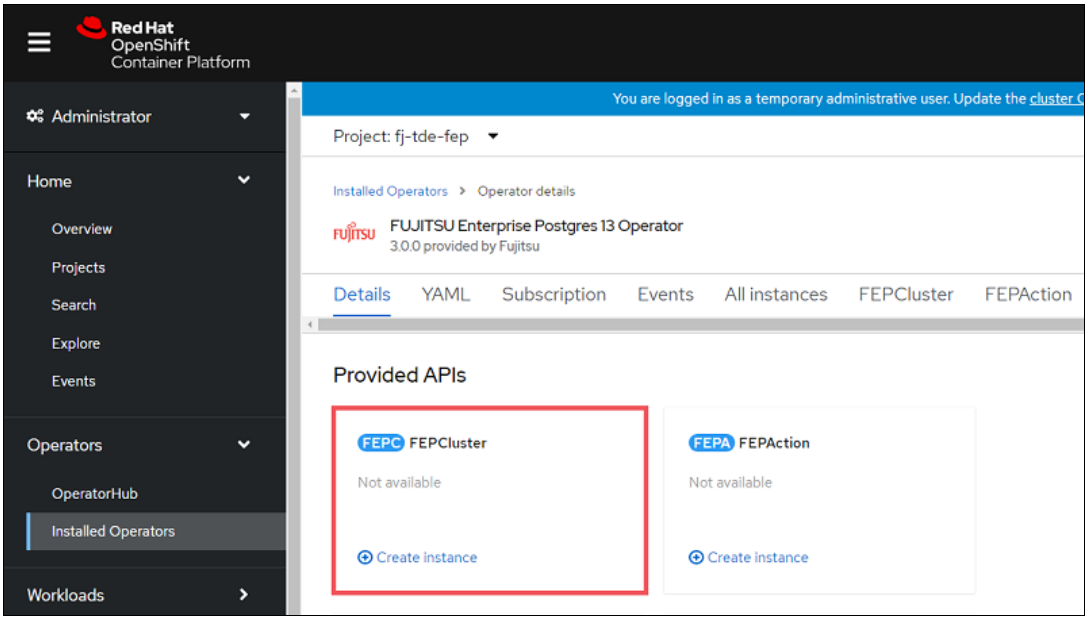

Figure 2-16 Creating a FUJITSU Enterprise Postgres cluster

In the Create FEPCluster window, click the **YAML** tab. Update the values as shown in Table 2-46 and CR configuration parameters as shown in Figure 2-17 on page 84. Update the deployment parameters and click **Create** to create a cluster.

Table 2-46 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 archival Vol: 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 *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

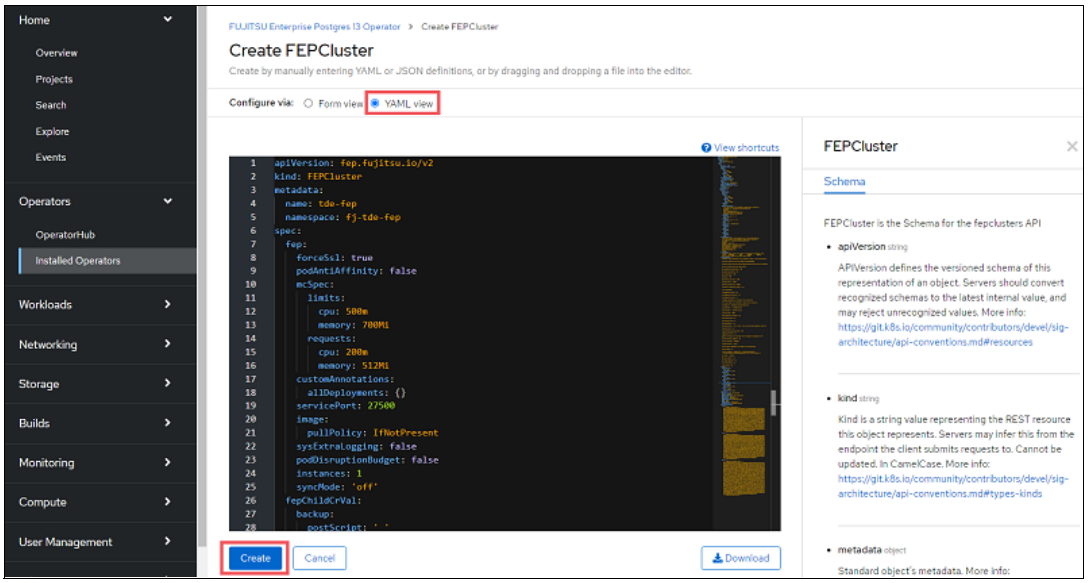

Figure 2-17 Updating the deployment parameters

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.

4. 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 2-25.

Example 2-25 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');
```

5. Verify that the created table space is the target of the encryption by using the command that is shown in Example 2-26.

Example 2-26 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)
```

6. Create a TDE-protected database, as shown in Example 2-27.

Example 2-27 Creating a TDE-protected database

```
postgres=# CREATE DATABASE securedb TABLESPACE secure_tablespace;
```

7. From the FUJITSU Enterprise Postgres client, connect to the `securedb` database and create `secure_table`. Then, insert the data ‘Hello World’ (Example 2-28).

Example 2-28 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');
```

8. Verify that the encrypted table can be referenced transparently (Example 2-29).

Example 2-29 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:

1. 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 2-47.

Table 2-47 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`	`abcdefghijk`
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 2-48.

   Table 2-48 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 2-30. The output of that command is shown in Example 2-31.

Example 2-30 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 2-31 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 2-32.

Example 2-32 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 2-33.

Example 2-33 Creating a password

```
$ oc create secret generic mydb-fep-private-key-password
--from-literal=keypassword=abcdefghijk -n my-namespace
```

4. Create the FUJITSU Enterprise Postgres server private key (Example 2-34).

Example 2-34 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 2-35.

Example 2-35 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 2-36).

Example 2-36 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 2-37).

Example 2-37 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 2-38).

Example 2-38 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 2-39).

Example 2-39 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 2-40).

Example 2-40 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 2-41).

Example 2-41 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 2-42).

Example 2-42 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 2-43).

Example 2-43 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 2-44).

Example 2-44 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 2-45).

Example 2-45 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 2-18 on page 91.

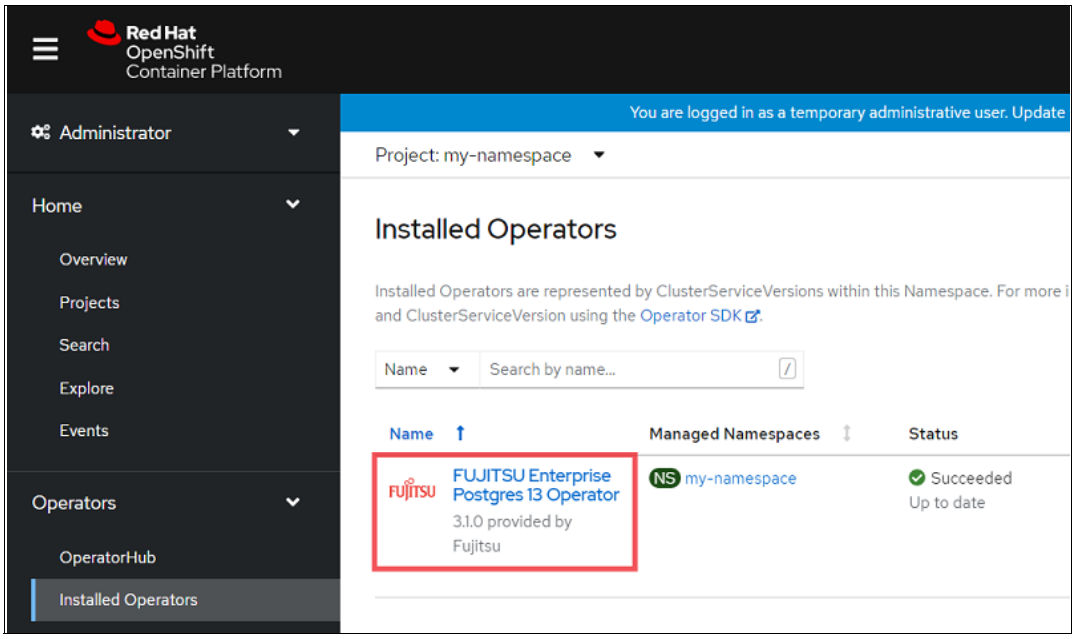

Figure 2-18 Installed Operators

19. In the Operator details window, select **Create Instance**, as shown in Figure 2-19.

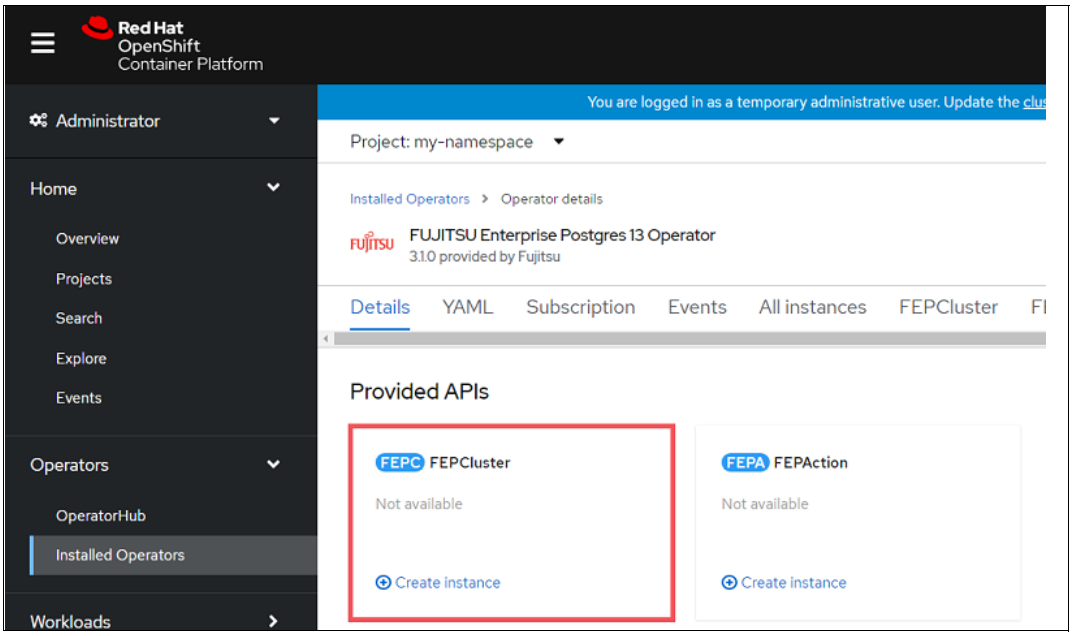

Figure 2-19 Creating a FUJITSU Enterprise Postgres cluster

In the Create FEPCluster window, click the **YAML** tab. Update the values as shown in Table 2-49. Update the deployment parameters and click **Create** to create a cluster, as shown in Figure 2-20 on page 95.

Table 2-49 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 `tls.crt`.

	`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 `ca.crt`.

`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 `tls.crt`.

	`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 `ca.crt`.

	`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 `pg_hba.conf`.

`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 *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

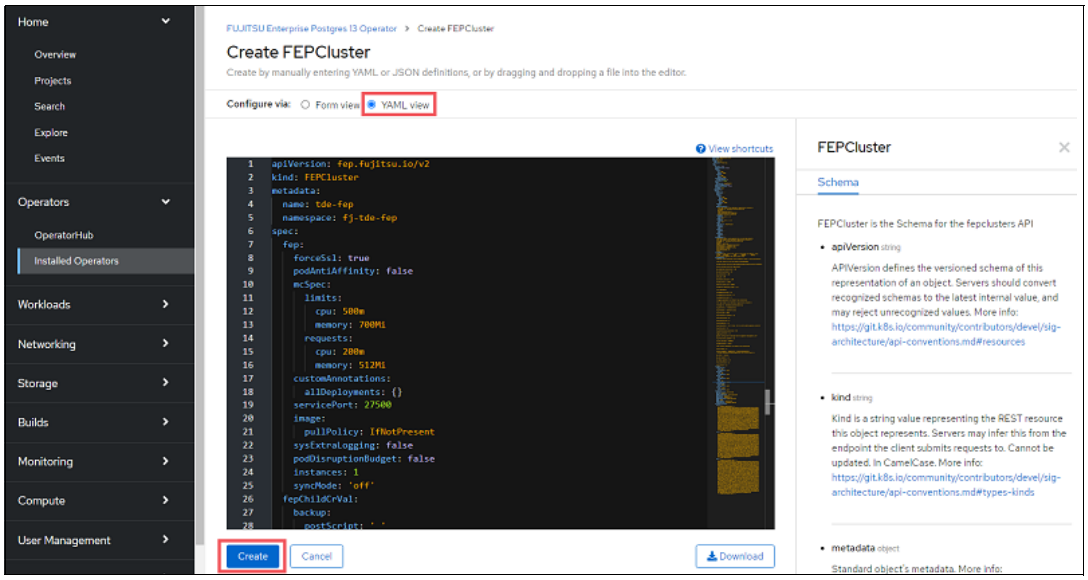

Figure 2-20 Updating deployment parameters

20. The database cluster is deployed, and the deployment status can be checked by selecting **Workloads** → **Pods**, as shown in Figure 2-21. The status shows *Running* after the cluster is ready.

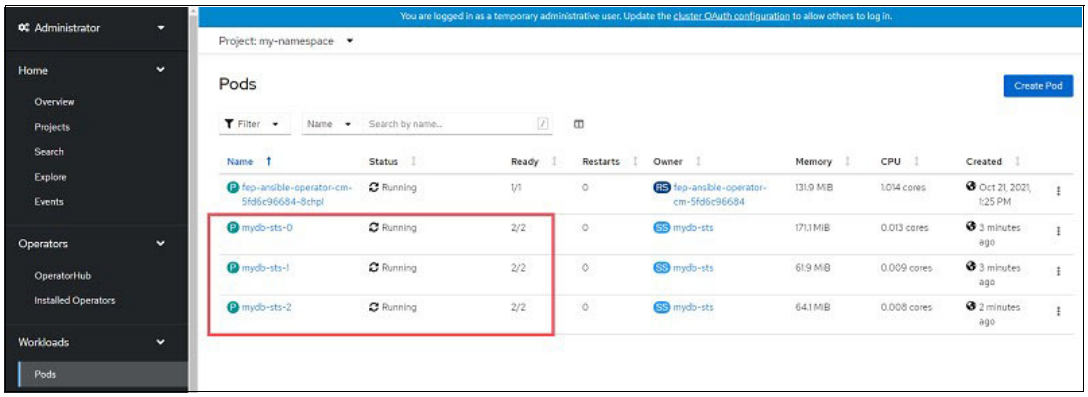

Figure 2-21 Database cluster deployment result

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 2-46.

Example 2-46 Connecting to the database cluster

```
$ psql 'host= mydb-primary-svc.my-namespace port=27500 user=app1user
sslcert=/client/tls.crt sslkey=/client/tls.key sslrootcert=/client/myca.pem
sslmode=verify-full'
```

3

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.

3.1 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:

- FUJITSU Enterprise Postgres leveraging container technology
- 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

Additionally, this chapter includes the following topics:

3.2 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.

3.2.1 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.

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.

As shown in Figure 3-1, 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.

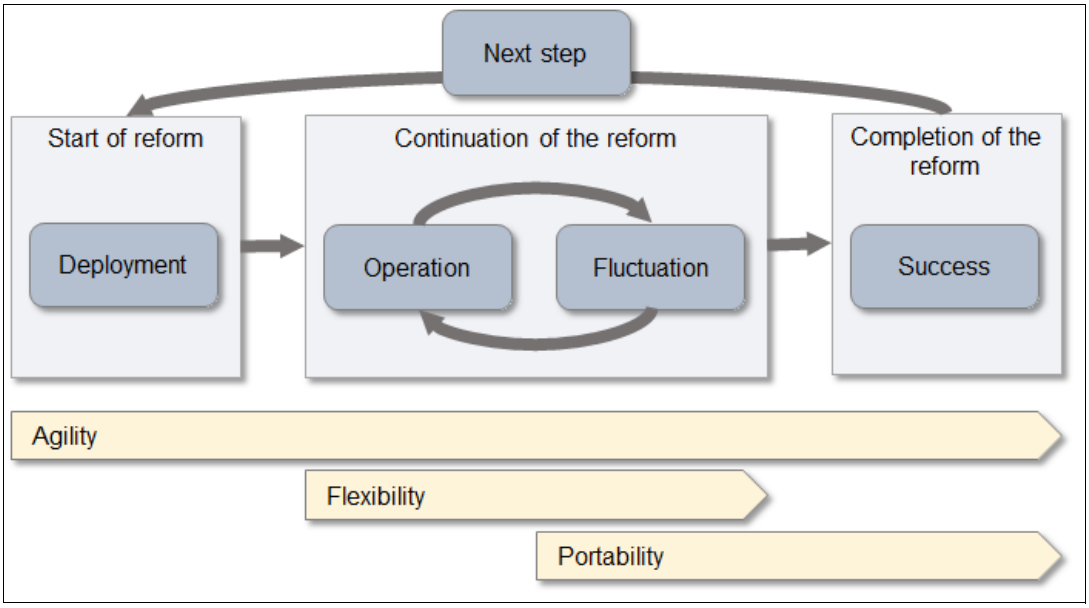

Figure 3-1 Customer journey through data and services modernization

The following sections describe quick deployment, low-cost database operations, maintaining optimal performance against fluctuation, and the next steps.

3.3 Deployment

This section describes how FUJITSU Enterprise Postgres Operator deploys databases quickly.

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.

Sections 3.3, “Deployment” on page 100 through 3.5, “Fluctuation” on page 134 assume that the configuration that is shown in Figure 3-2 on page 101 is used.

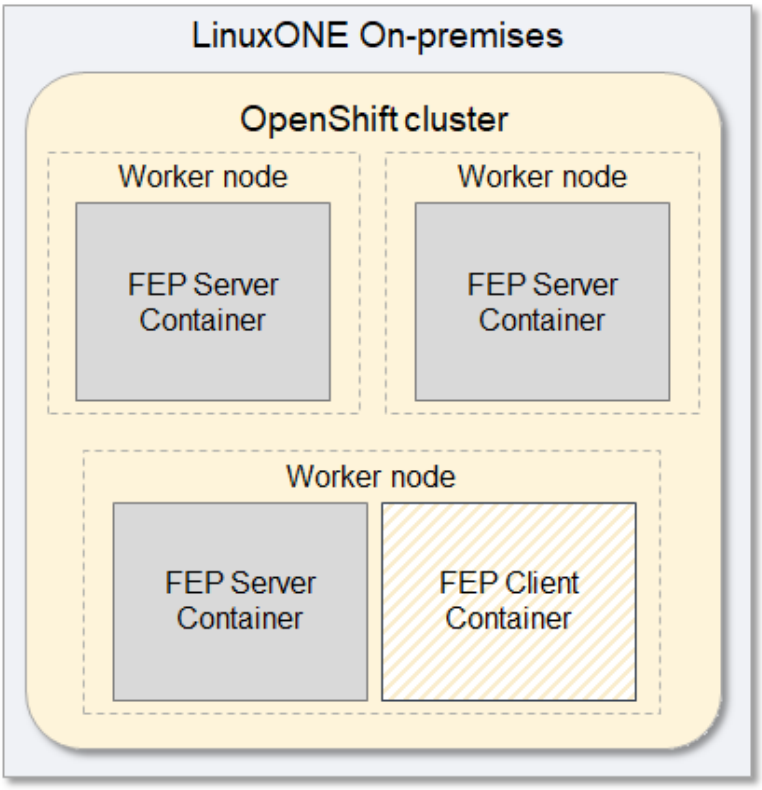

Figure 3-2 System deployment overview

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*.

3.3.1 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 3-3 on page 103.

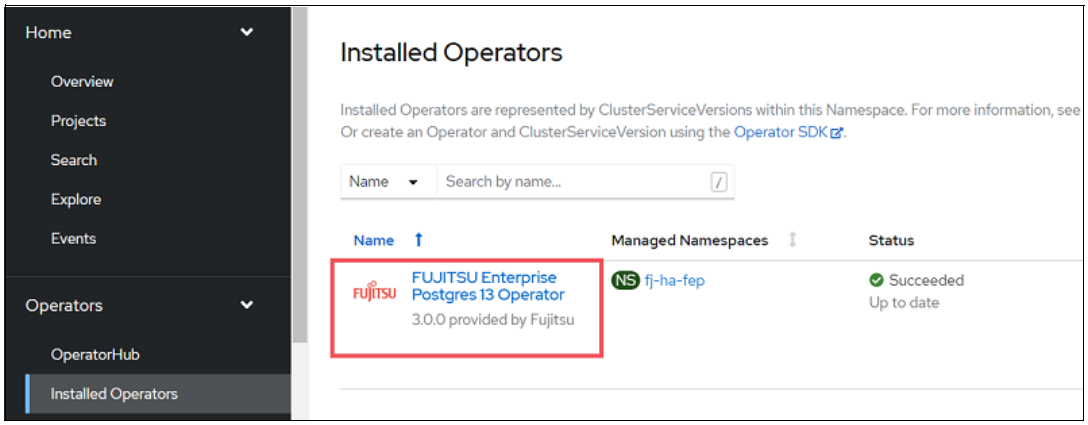

Figure 3-3 Selecting FUJITSU Enterprise Postgres13 Operator

3. In the Operator details window, select **Create Instance**, as shown in Figure 3-4.

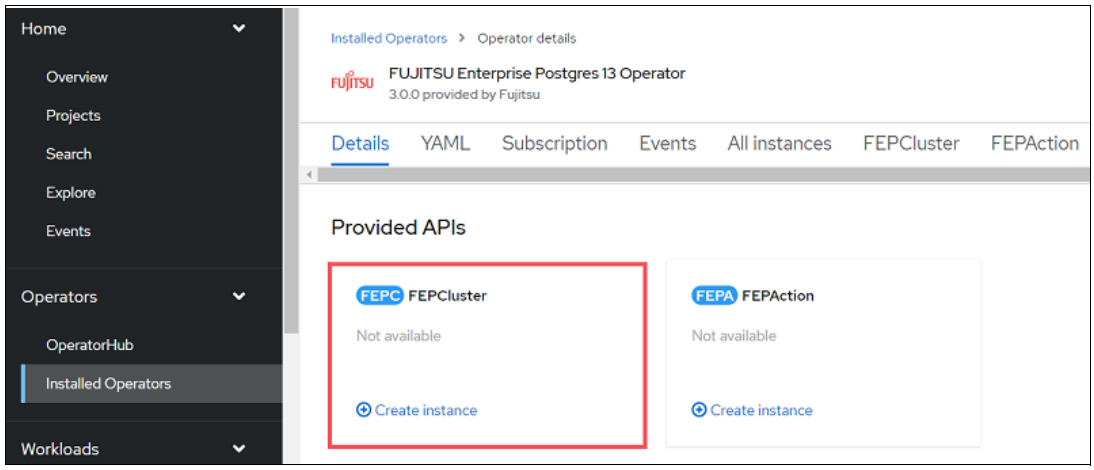

Figure 3-4 Creating a FUJITSU Enterprise Postgres cluster

4. In the Create FEPCluster window, click the **YAML** tab, and update the parameter values as shown in Table 3-1. To create a cluster, click **Create**, as shown in Figure 3-5 on page 105.

Table 3-1 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.
	tdepassphrase: tde-passphrase	Passphrase for Transparent Data Encryption (TDE).
spec: fepChildCrVal: storage:	dataVol: size: 2Gi storageClass: gold walVol: size: 1200Mi storageClass: gold tablespaceVol: size: 512Mi storageClass: gold archivewalVol: 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 *Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE*, SG24-8499.

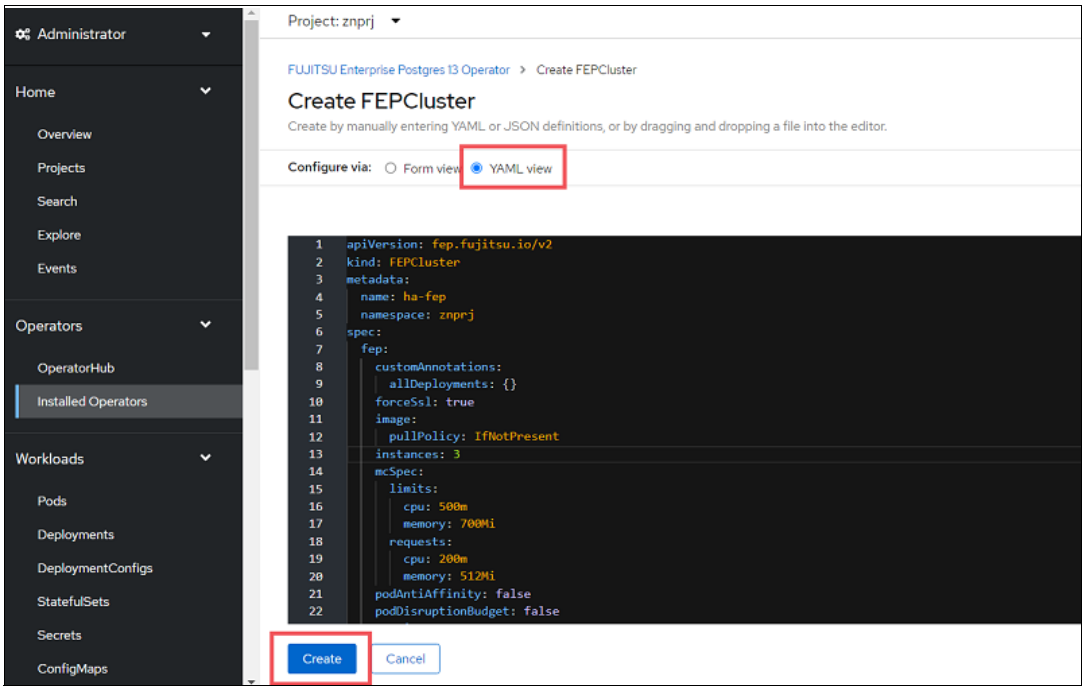

Figure 3-5 Updating parameters for deployment

5. The HA cluster is deployed, and the deployment status can be checked by selecting **Workloads** → **Pods**, as shown in Figure 3-6. `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.

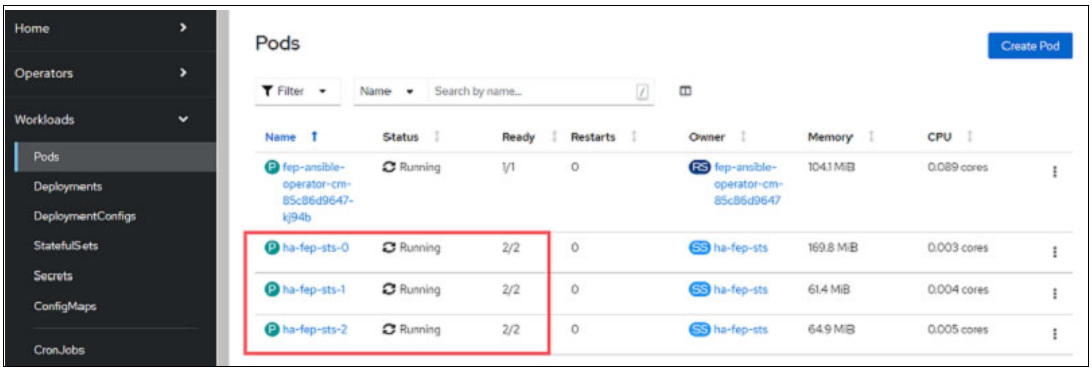

Figure 3-6 HA cluster deployment result

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*.

3.4 Operation

This chapter 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.

3.4.1 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 3.4.3, “Monitoring” on page 120.

1. In the Red Hat OpenShift Console, select **Operators** → **Installed Operators**, as shown in Figure 3-7 on page 107.

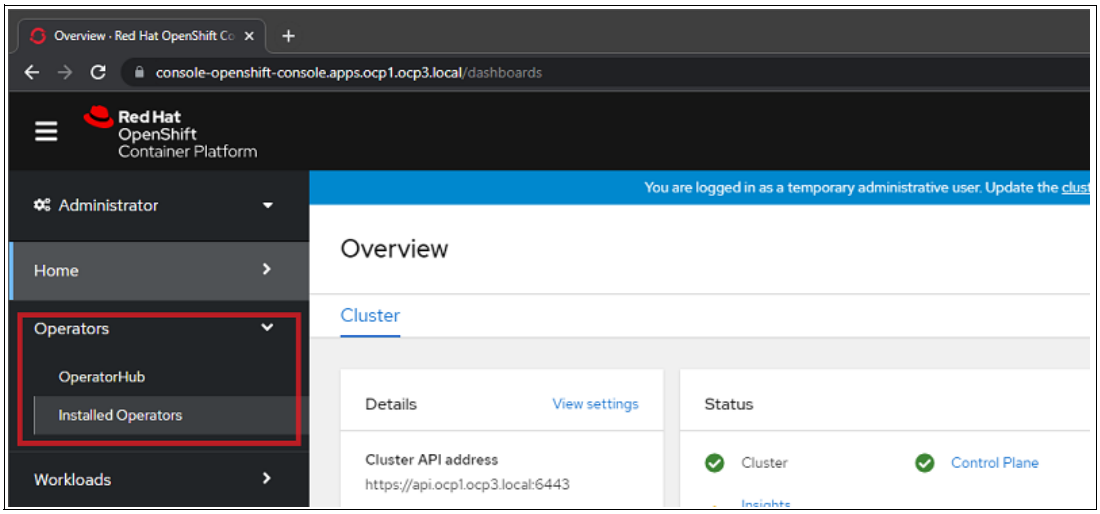

Figure 3-7 Navigating to the Installed Operators window

2. Select **FUJITSU Enterprise Postgres 13 Operator**, as shown in Figure 3-8.

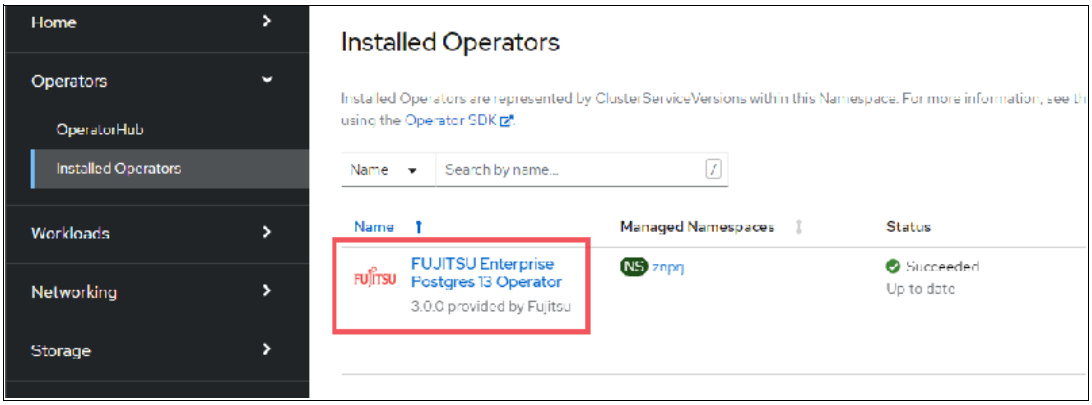

Figure 3-8 Selecting FUJITSU Enterprise Postgres 13 Operator

3. Select the **FEPCluster** tab, and select **ha-fep**, as shown in Figure 3-9.

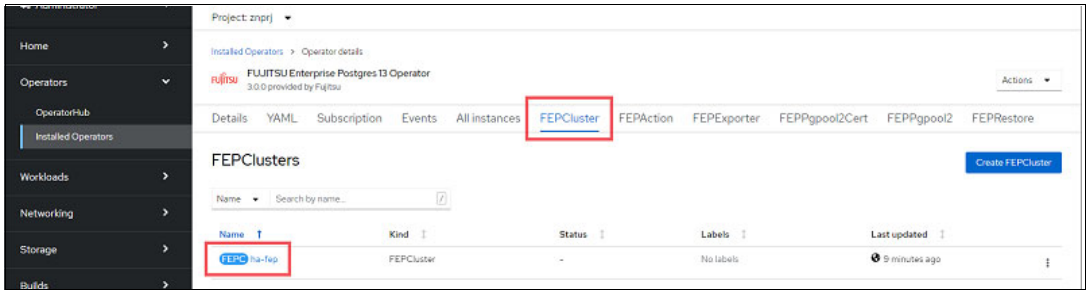

Figure 3-9 Selecting ha-fep on the FEPCluster tab

4. In the FEPCluster details window, select the **YAML** tab, as shown in Figure 3-10. Set the parameters as shown in Table 3-2.

   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.

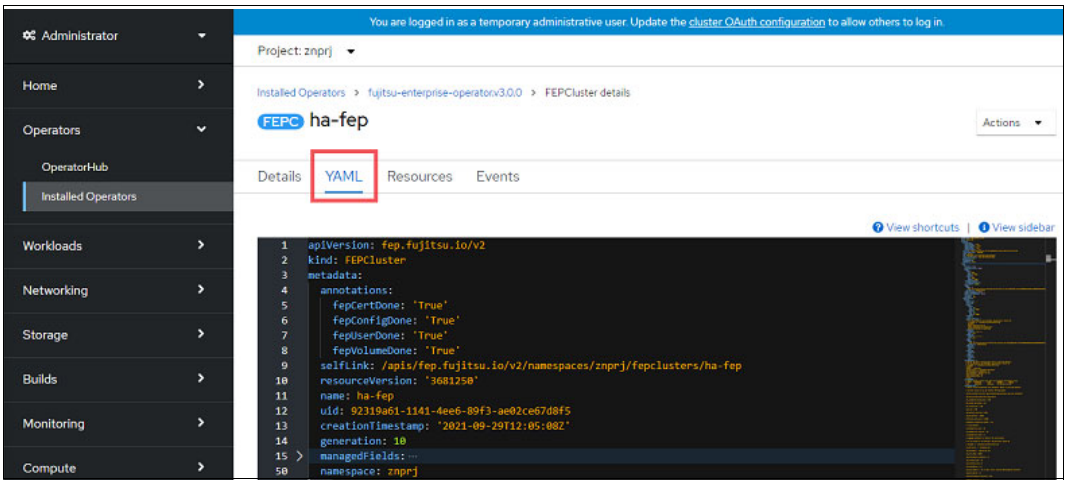

Figure 3-10 Selecting the YAML tab in the FEPCluster details window

Table 3-2 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	Set the first backup schedule in Cron format. 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. The date and time are Coordinated Universal Time time. 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: 0 5 * * *
spec: fepChildCrVal: backup: schedule1: type	full	Set the backup type of the first backup schedule to full backup. full: Perform a full backup (back up the contents of the database cluster). incr: Perform an incremental backup (back up only the database cluster files that were changed since the last backup).
spec: fepChildCrVal: backup: schedule2: schedule	0 6 * * *	Set the second backup schedule in cron format. The date and time are Coordinated Universal Time time. In this example, we set the time to 13:00 every day in the Coordinated Universal Time -5 time zone. For example, if you want to take a backup every day at 13:00, set the following value: 0 6,18 * * *
spec: fepChildCrVal: backup: schedule2: type	incr	Set the backup type of the second backup schedule to incremental backup. full: Perform a full backup (back up the contents of the database cluster). incr: Perform an incremental backup (back up only the database cluster files that changed since the last backup).

5. Click **Save** to apply the changes, as shown in Figure 3-11. A message displays that confirms that the changes were saved successfully, as shown in Figure 3-12.

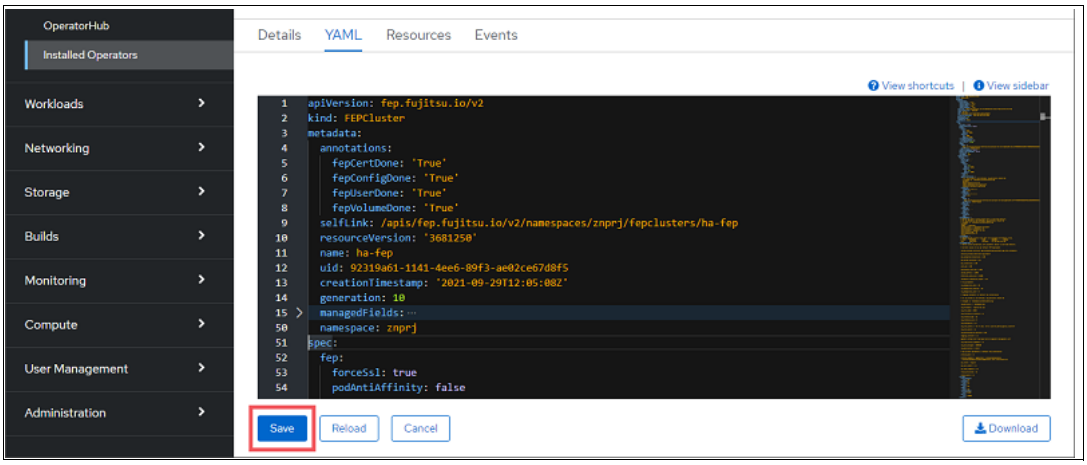

Figure 3-11 Saving configuration changes

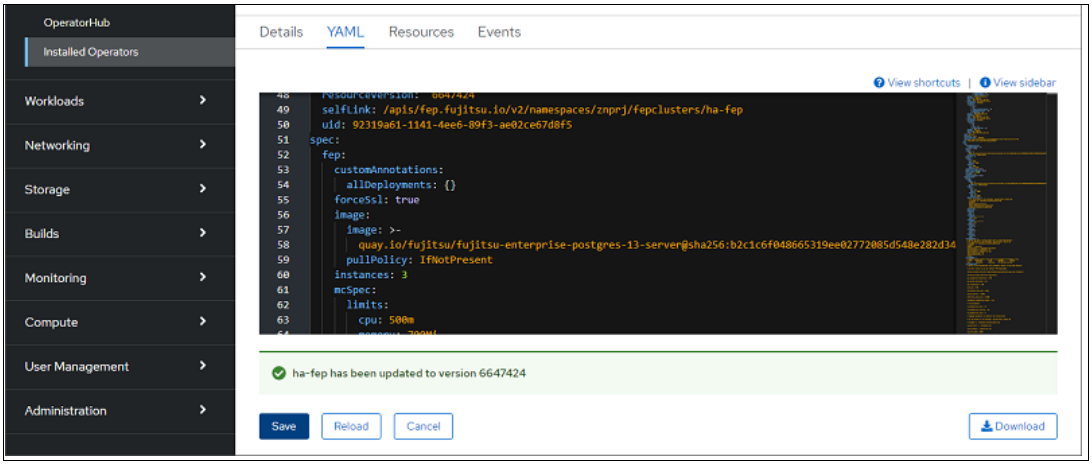

Figure 3-12 Confirmation message

Note: If saving is unsuccessful, click **Reload** to apply the changes again, and then click **Save**.

6. Wait for the operator to apply the saved changes.
7. Check that the backup schedule was created in the CronJobs window, as shown in Figure 3-13 on page 111.

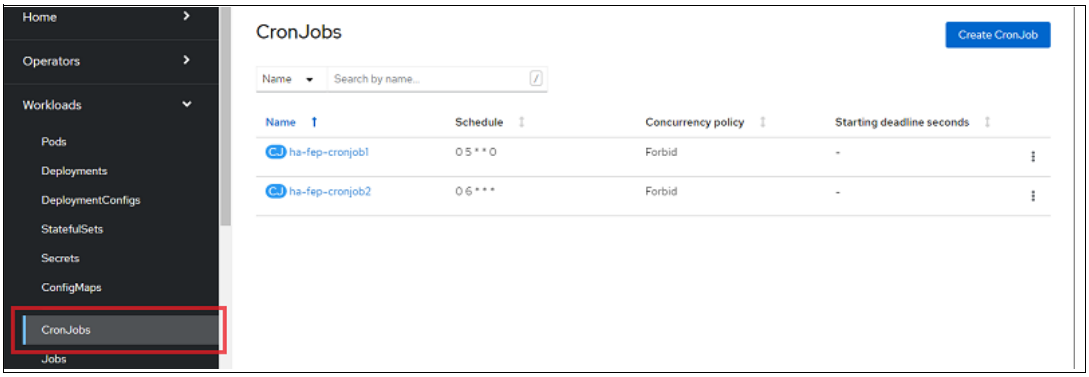

Figure 3-13 Confirming that the backup schedules were created

Verifying the automatic backup

To verify the automatic backup, complete the following steps:

1. Prepare data (Example 3-1).

   By using the FUJITSU Enterprise Postgres client, create a database that is named `publisher` with `pgbench`.

   Example 3-1 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 3-14.

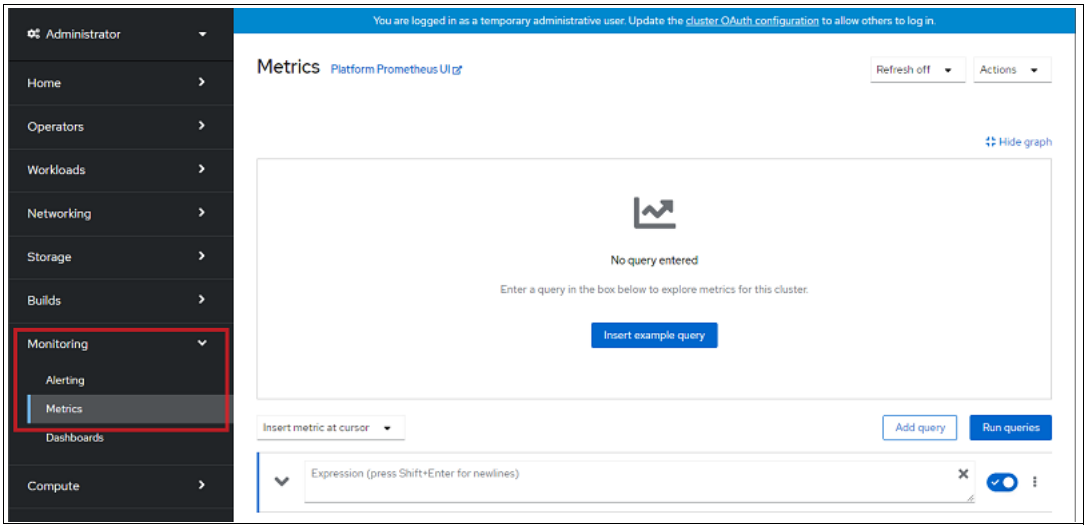

Figure 3-14 Metrics window

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 3-15.

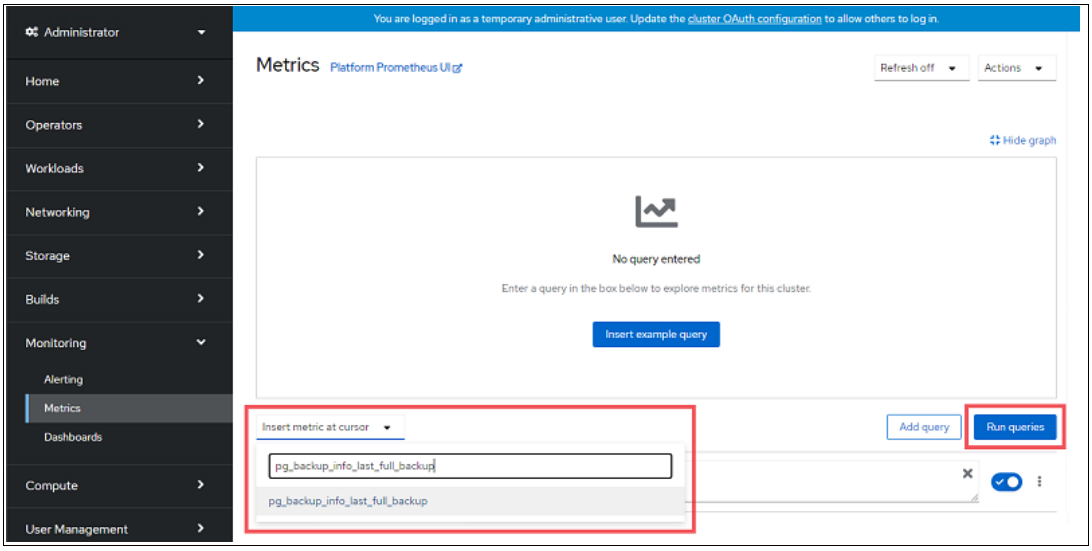

Figure 3-15 Selecting pg_backup_info_last_full_backup

4. Click **Run queries**. A chronological graph appears at the top, and a table appears at the bottom, as shown in Figure 3-16. 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:

   a. Copy the timestamp value.

   b. 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
      ```

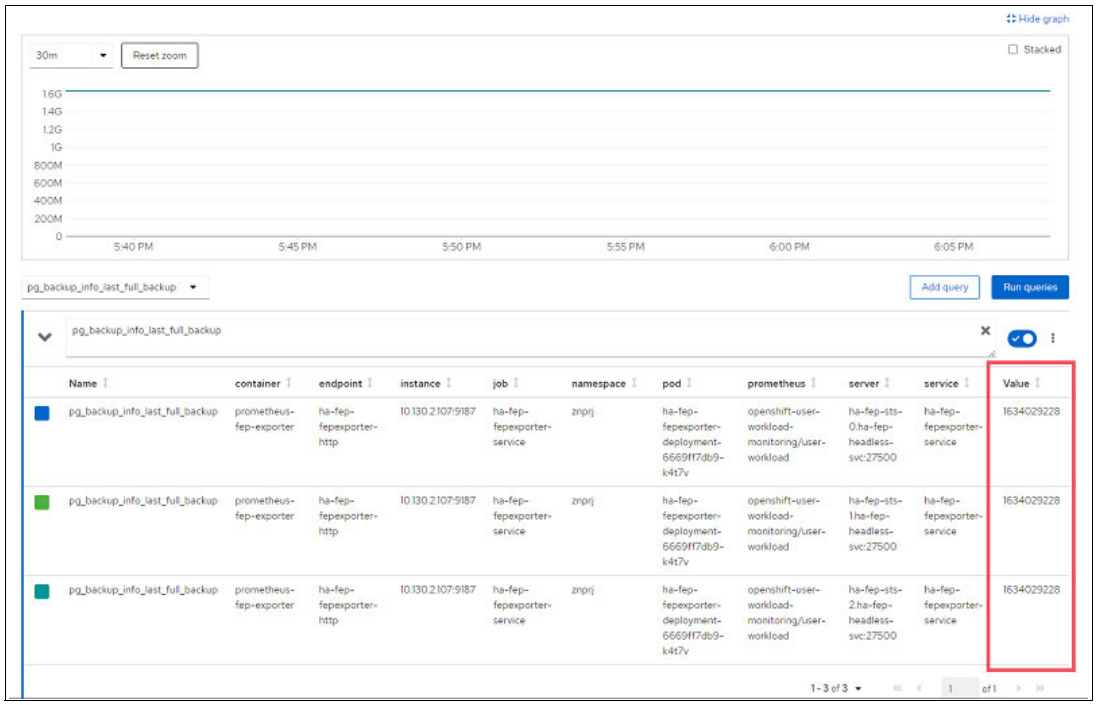

Figure 3-16 Displaying timestamps of the latest backup

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 3-17.

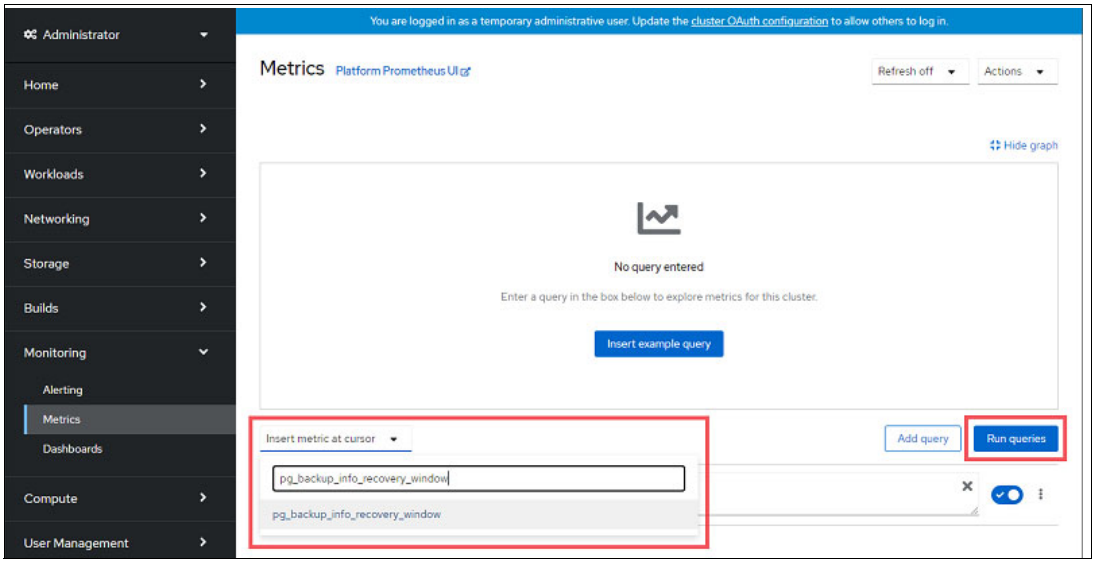

Figure 3-17 Selecting pg_backup_info_recovery_window

6. Click **Run queries**. The timestamps of recovery windows appear, as shown in Figure 3-18.

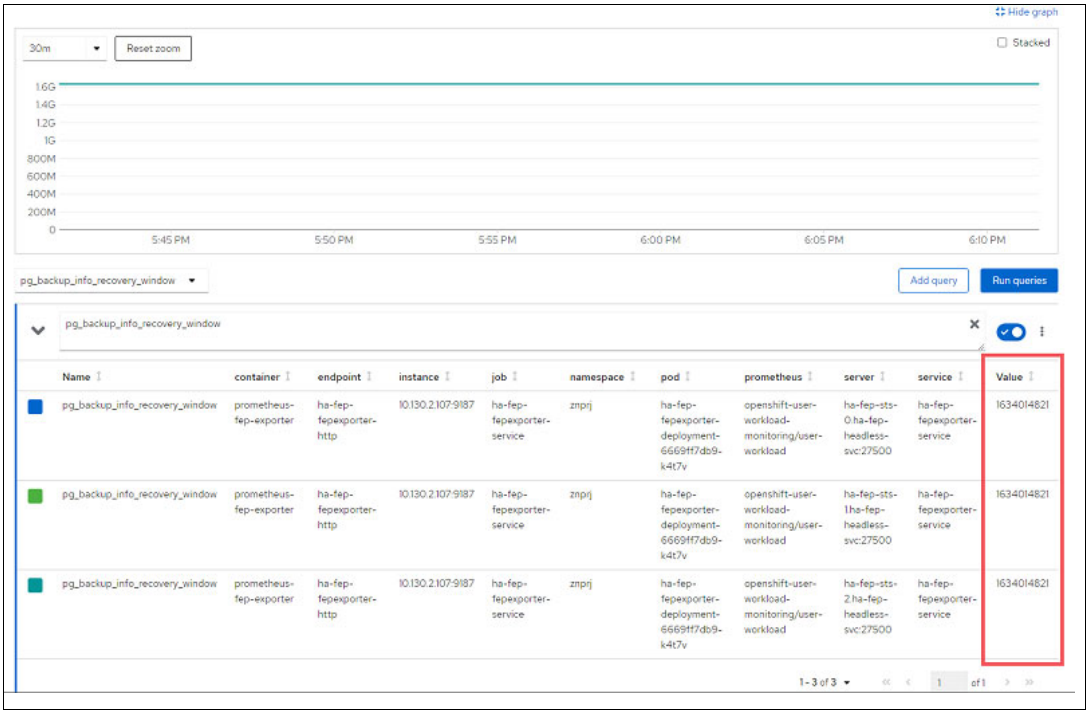

Figure 3-18 Checking recovery windows

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 3-19.

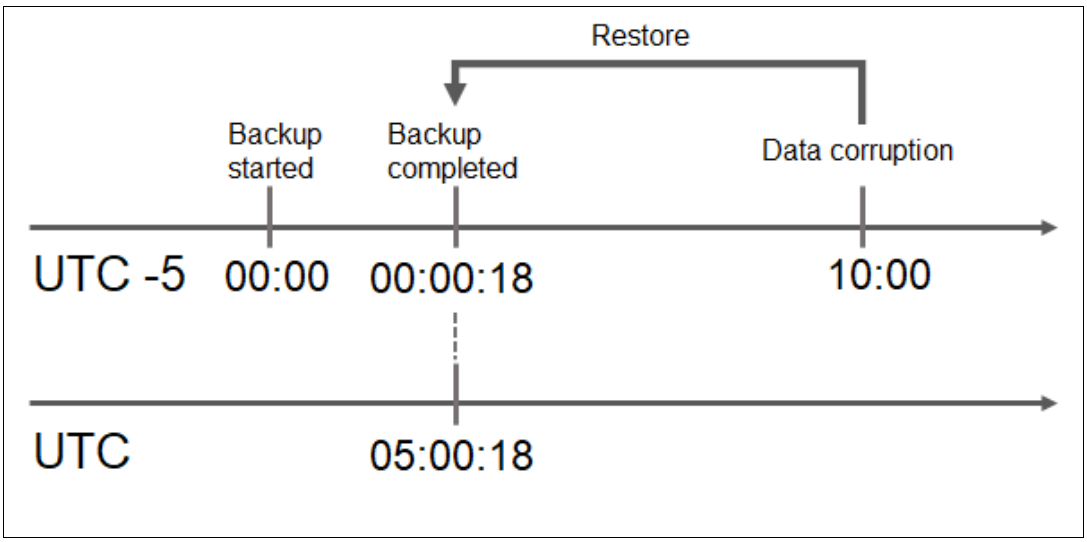

Figure 3-19 Example scenario

Complete the following steps:

1. On Red Hat OpenShift Console, clicks **Installed Operators**, as shown in Figure 3-20.

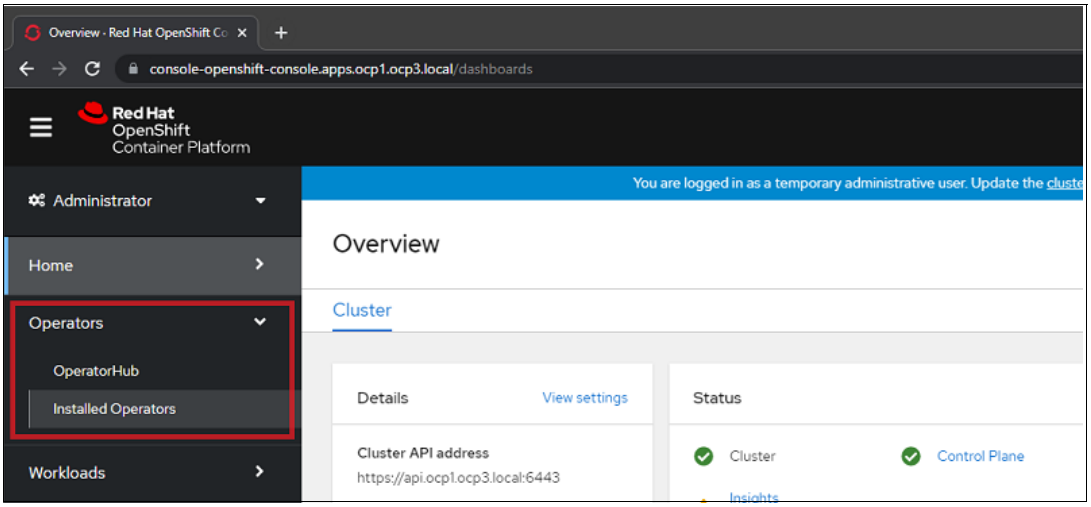

Figure 3-20 installed Operators window

2. Select **FUJITSU Enterprise Postgres 13 Operator**, as shown in Figure 3-21.

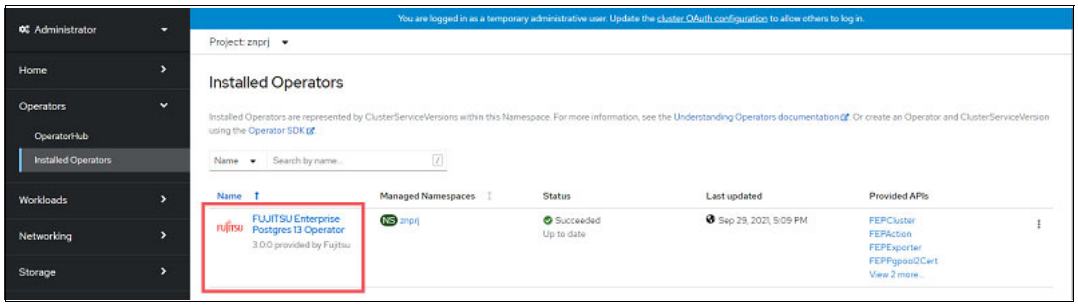

Figure 3-21 Selecting FUJITSU Enterprise Postgres 13 Operator

3. Select **Create Instance** for FEPRestore, as shown in Figure 3-22.

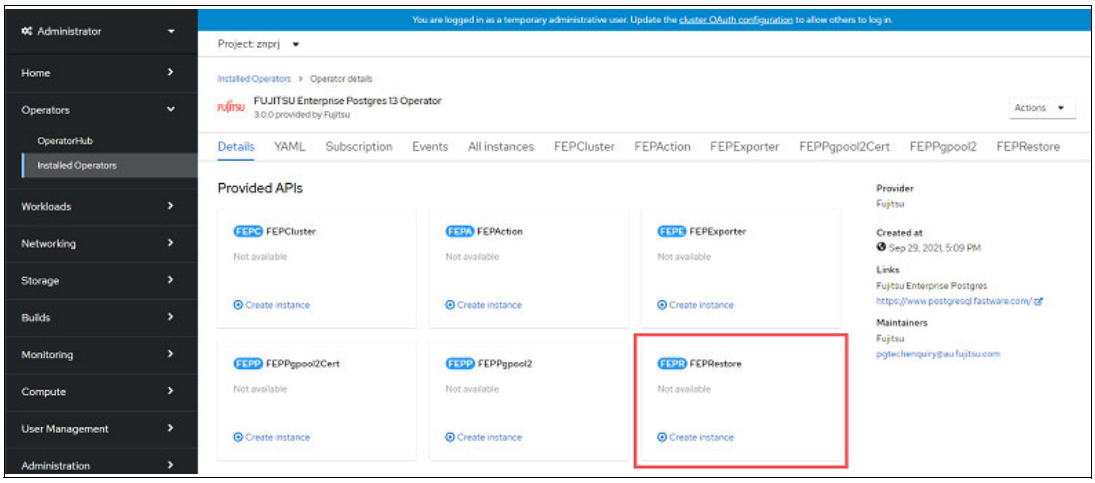

Figure 3-22 Creating FEPRestore instance

4. In the Create FEPRestore window, click the **YAML** tab. Update the values as shown in Table 3-3 and click **Create**, as shown in Figure 3-23 on page 117. Wait for the FEPCluster to be re-created and restored.

Table 3-3 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 113. Make sure to convert the time to Coordinated Universal Time.

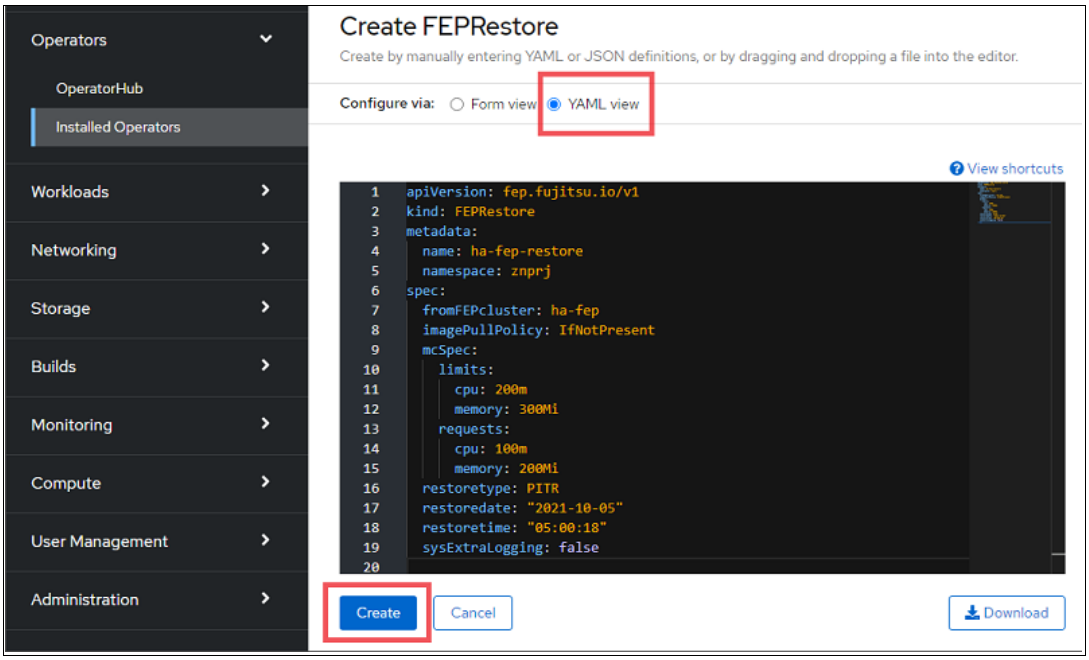

Figure 3-23 Changing the configuration and creating FEPRestore

5. The restore is complete when all FUJITSU Enterprise Postgres cluster pods are re-created and the pod status is `Running`.
6. With FUJITSU Enterprise Postgres client, check that the database is restored (Example 3-2).

Example 3-2 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)
```

3.4.2 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 3.3, “Deployment” on page 100 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 `feprole=master` and `app=ha-fep-sts`. Possible candidates appear when you start typing, so select the appropriate label, as shown in Figure 3-24. The list of master pods displays, and we can confirm that `ha-fep-sts-0` is a master pod.

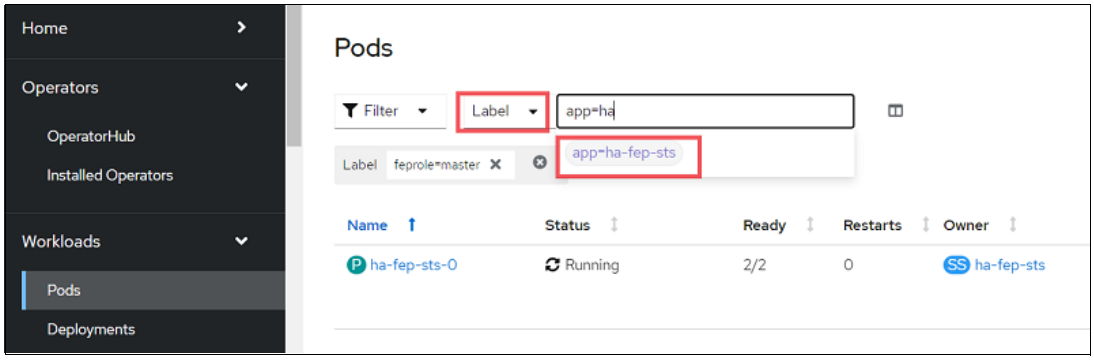

Figure 3-24 Identifying the current master pod

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 3-25 on page 119.

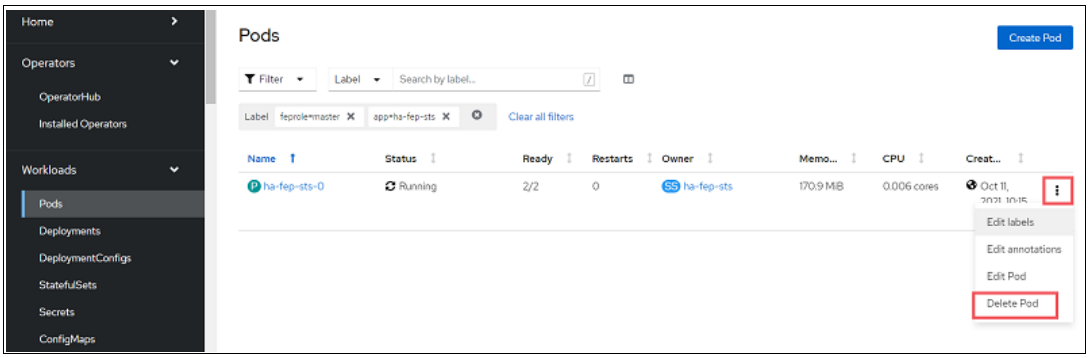

Figure 3-25 Removing the master pod

4. The status of `ha-fep-sts-0` pod changes to *Terminating*, as shown in Figure 3-26.

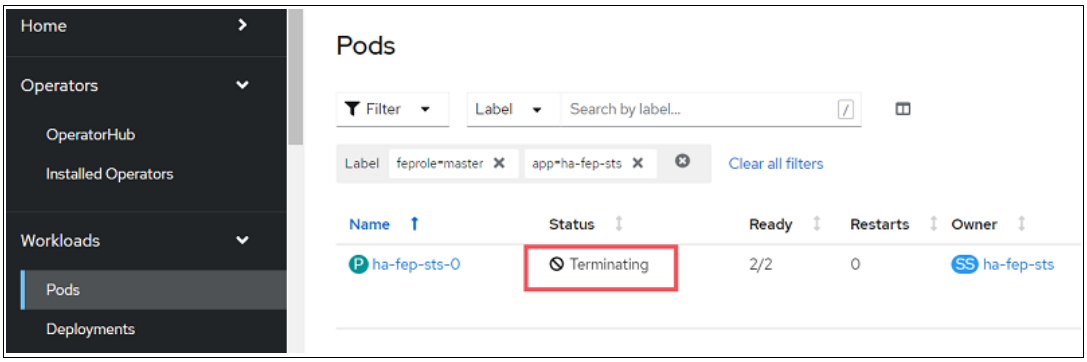

Figure 3-26 Checking the master pod status

5. The pod that was promoted to master appears, as shown in Figure 3-27. In this simulation, we can see that `ha-fep-sts-2` pod was promoted to the master.

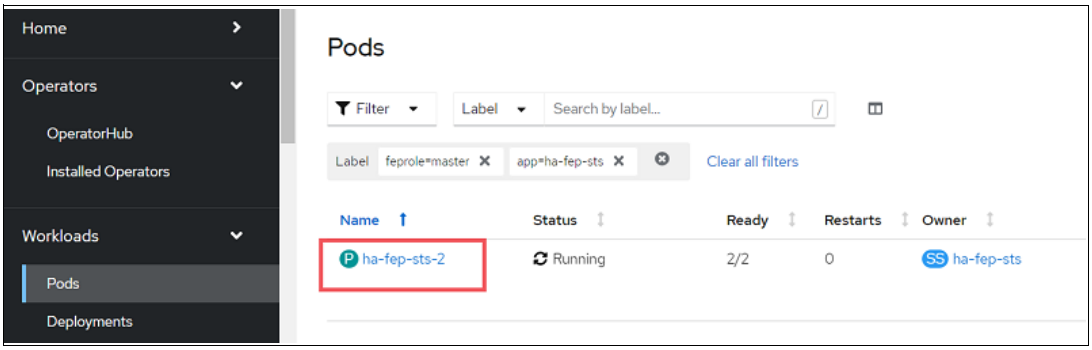

Figure 3-27 Master pod after automatic failover

6. To check the results of automatic recovery, open the replica pods in the list by removing the `feprole=master` label, as shown in Figure 3-28. The status of `ha-fep-sts-0`, which is the old master pod, changed to *Running*. We can confirm that automatic recovery completed.

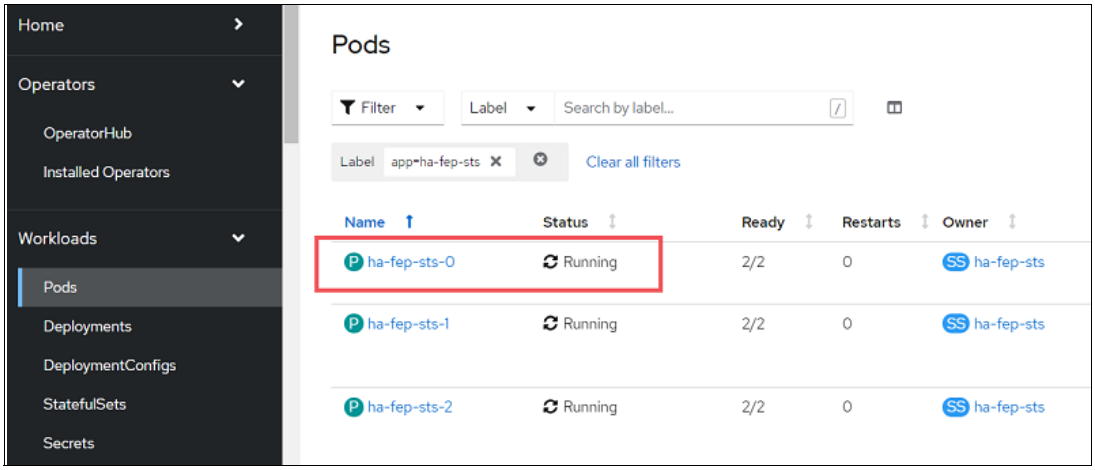

Figure 3-28 Checking automatic recovery

7. Type `feprole=replica` into the search box. The list of replica pods displays, as shown in Figure 3-29. 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.

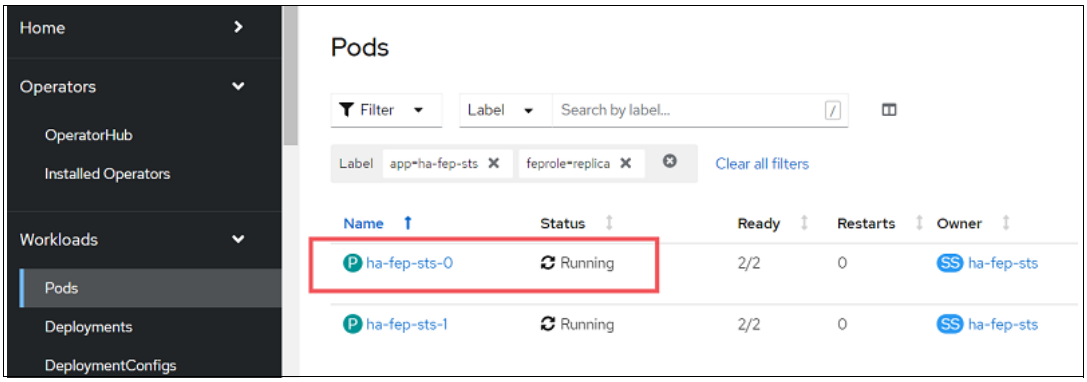

Figure 3-29 Replica pods after automatic failover

3.4.3 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 3-4 by performing step 4 on page 103. For a deployed cluster, add this parameter into the existing FEPCluster CR. To save the changes, click **Save**.

Table 3-4 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 3-30.

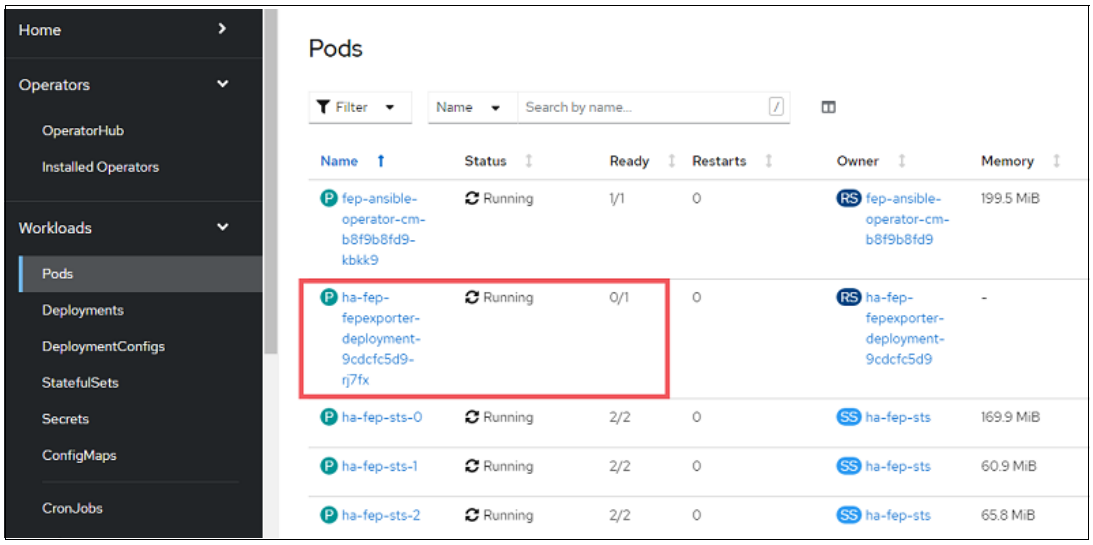

Figure 3-30 Checking the Exporter status

Checking monitoring metrics

Complete the following steps:

1. Check the monitoring metrics on Prometheus by selecting **Monitoring** → **Dashboards**, as shown in Figure 3-31.

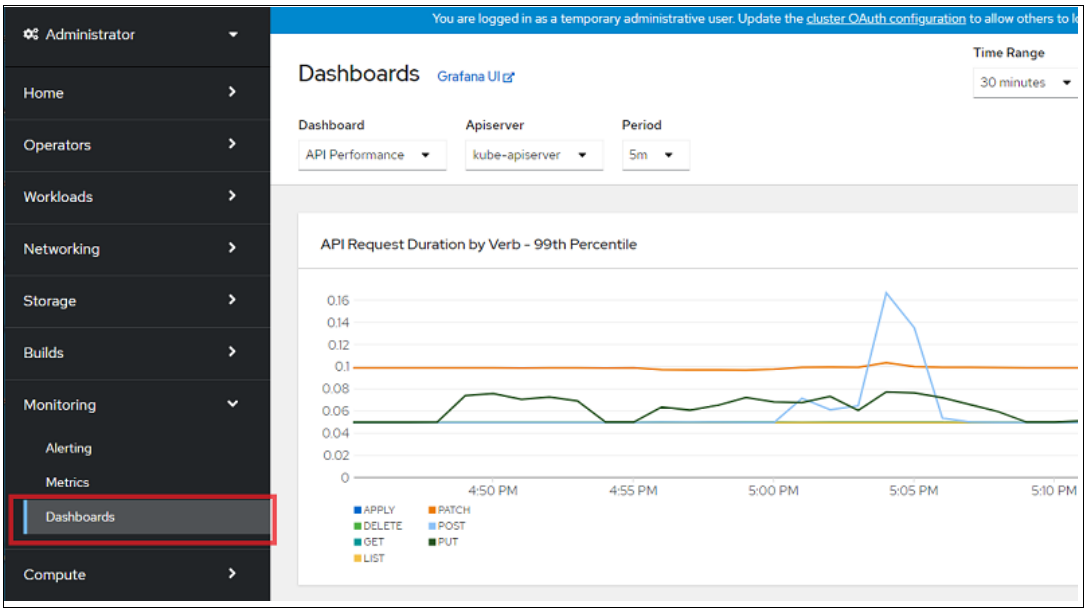

Figure 3-31 Navigating to the monitoring dashboard window

2. In the drop-down list for **Dashboard**, select **Kubernetes / Compute Resources / Pod**, as shown in Figure 3-32 on page 123.

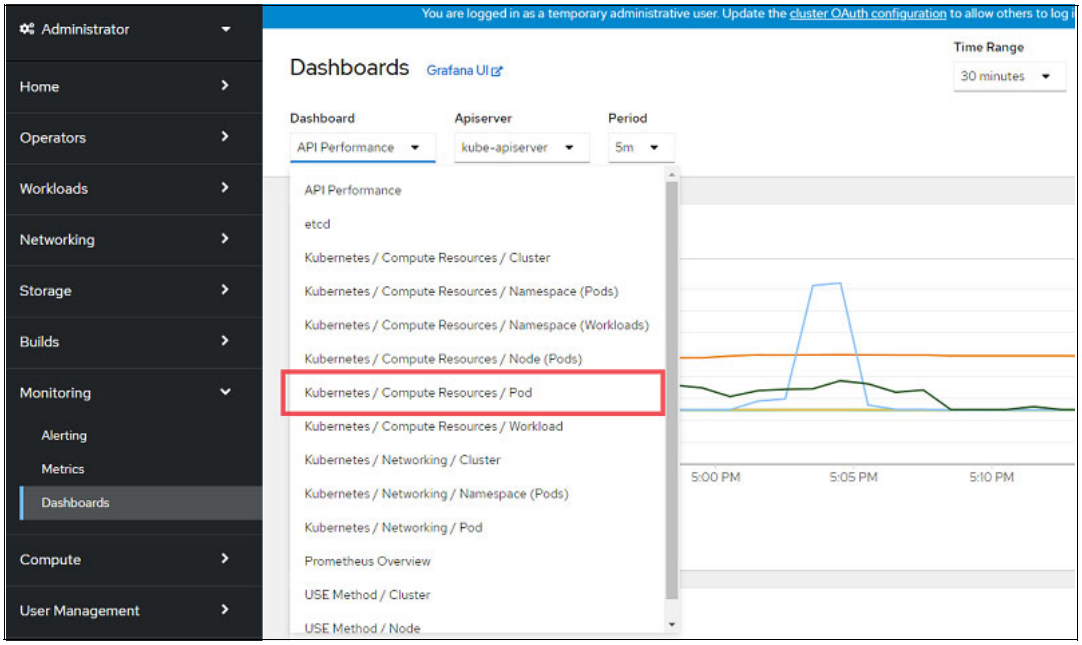

Figure 3-32 Selecting monitoring metrics

3. In the drop-down list for **Namespace**, select the project that was created in 3.3.1, “Automatic instance creation” on page 102. In the drop-down list for **Pod**, select **ha-fep-sts-0**. CPU utilization can be checked, as shown in Figure 3-33. Based on the information that is displayed in this window, database administrators can decide whether a resource should be added.

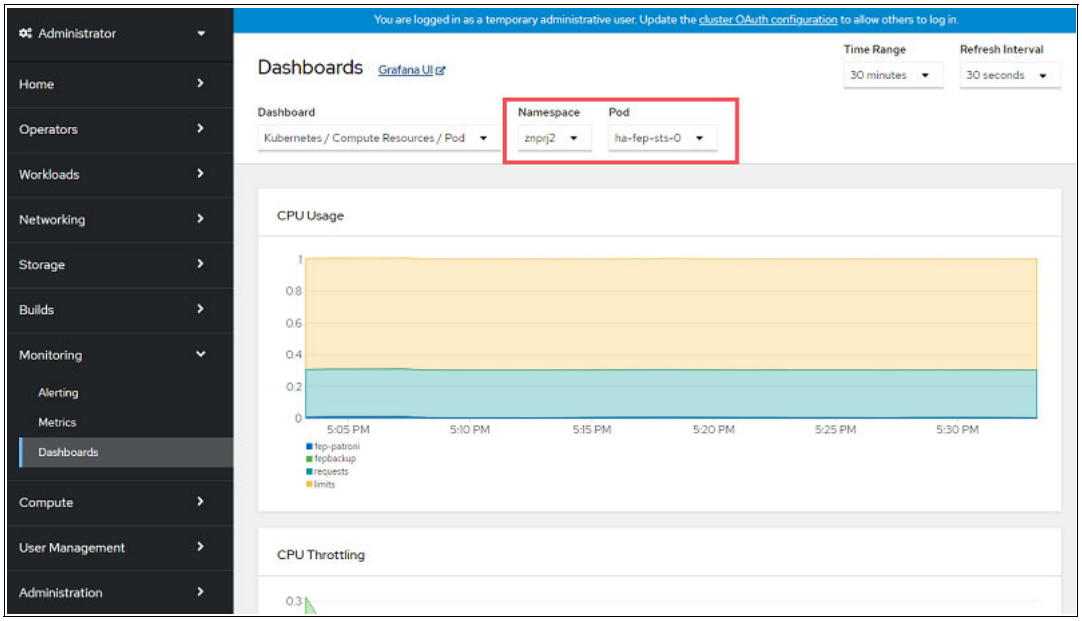

Figure 3-33 Checking the CPU utilization on Prometheus

4. 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 3-34.

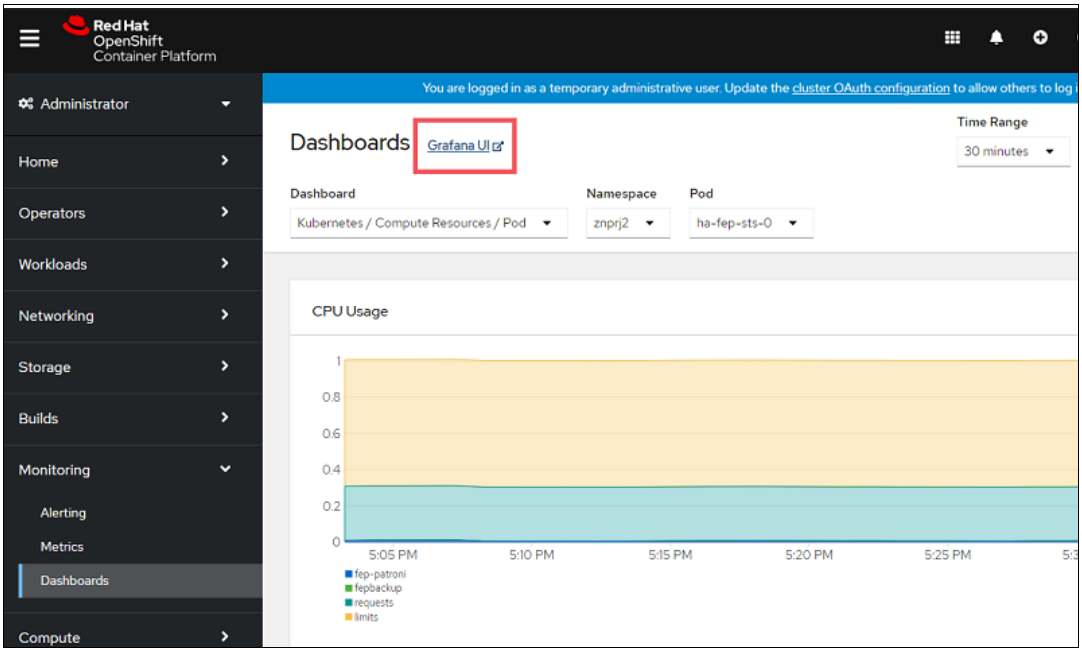

Figure 3-34 Selecting Grafana UI

5. Grafana opens in a separate window. When prompted to log in, enter your credentials. If permission is requested, grant permission.
6. On the Grafana home window, select the search icon, as shown in Figure 3-35.

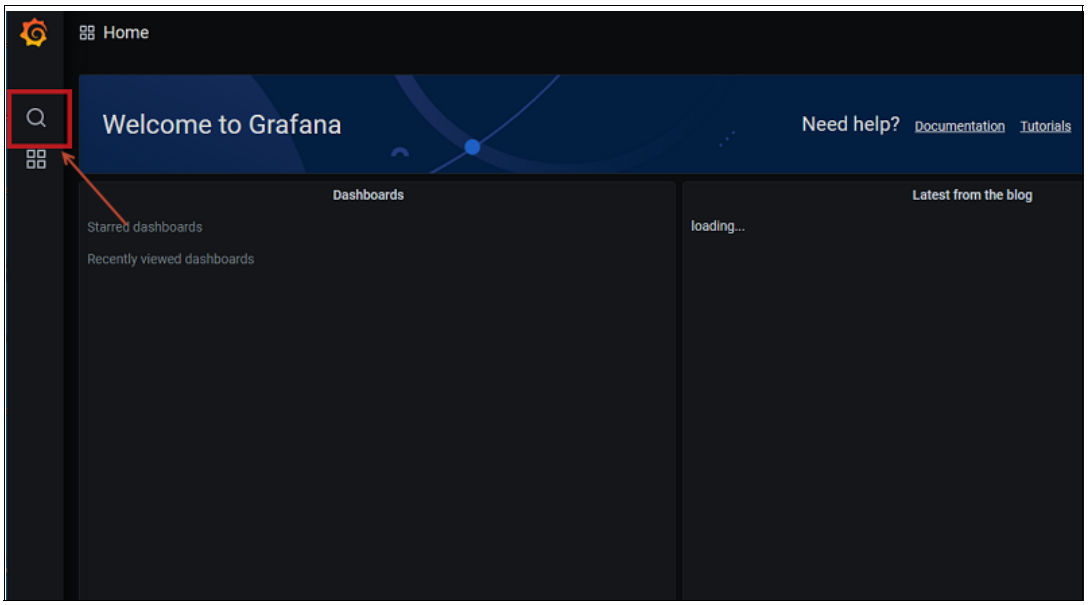

Figure 3-35 Selecting the search icon on the Grafana UI home page

7. Select **Kubernetes / Compute Resources / Pod** under the **Default** folder, as shown in Figure 3-36 on page 125.

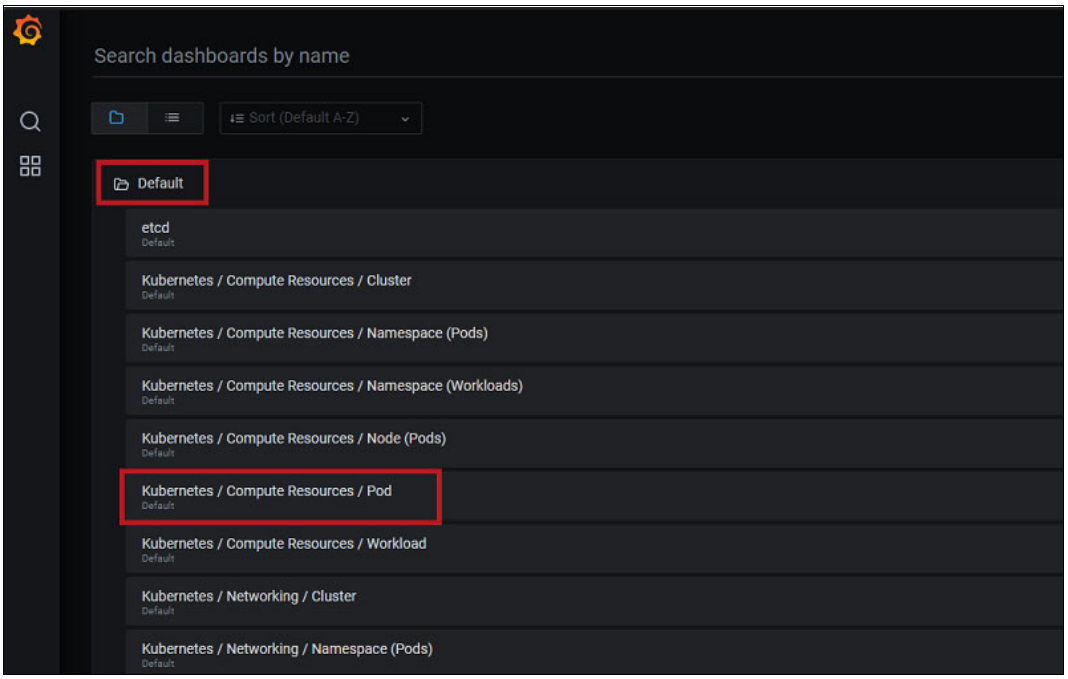

Figure 3-36 Selecting CPU utilization on the Grafana UI

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 3-37. 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 131.

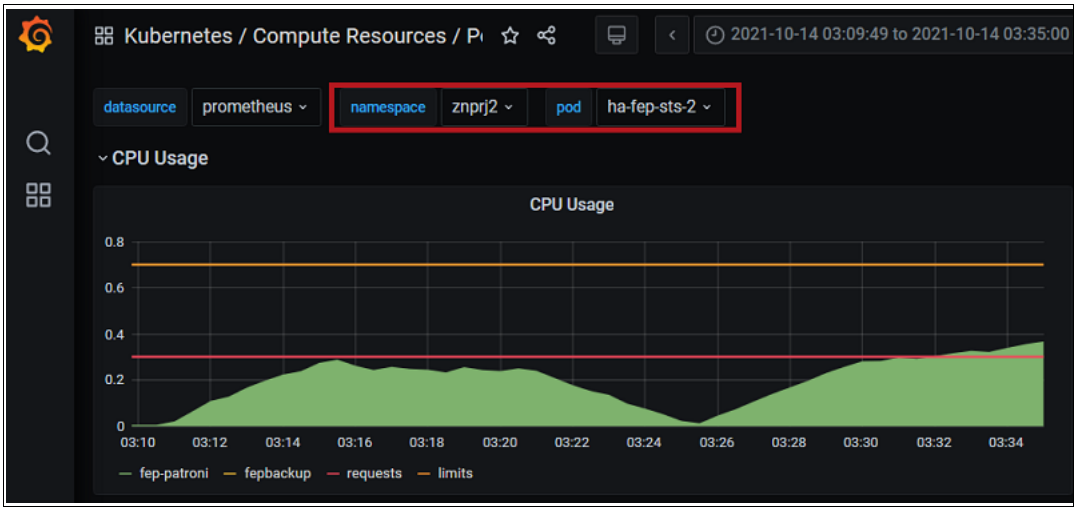

Figure 3-37 Checking the CPU utilization on the Grafana UI

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 3-38.

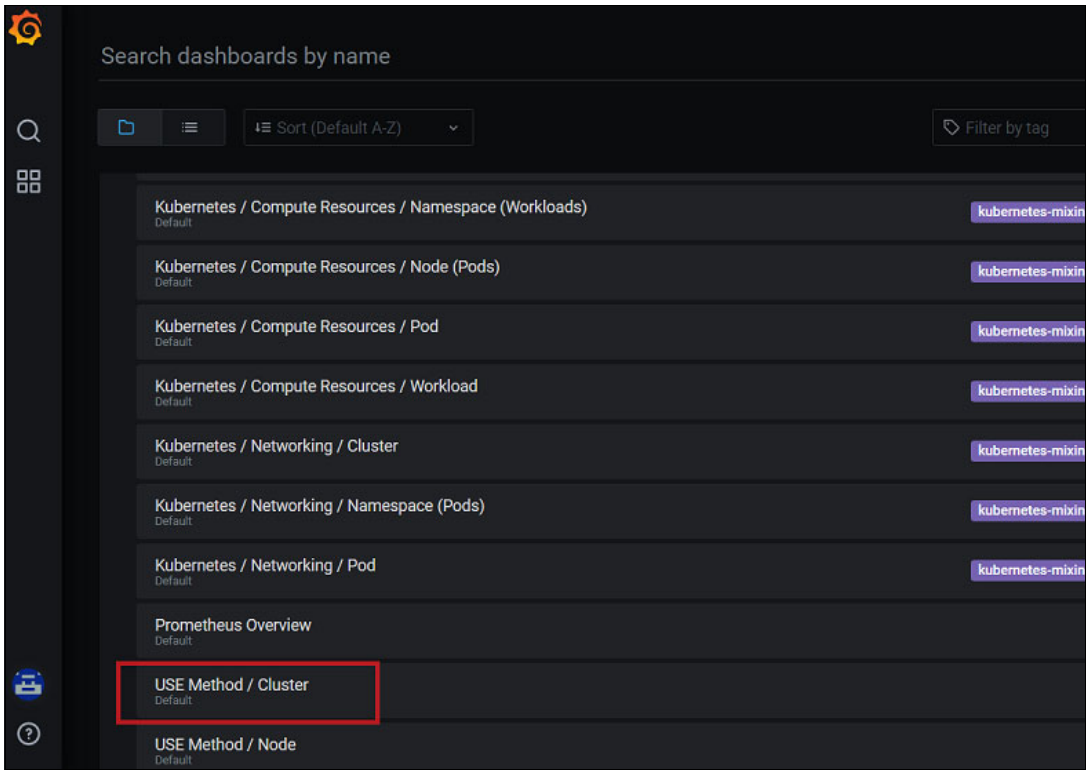

Figure 3-38 Selecting disk information on the Grafana UI

10. Scroll down to view a graph of the disk usage, as shown in Figure 3-39.

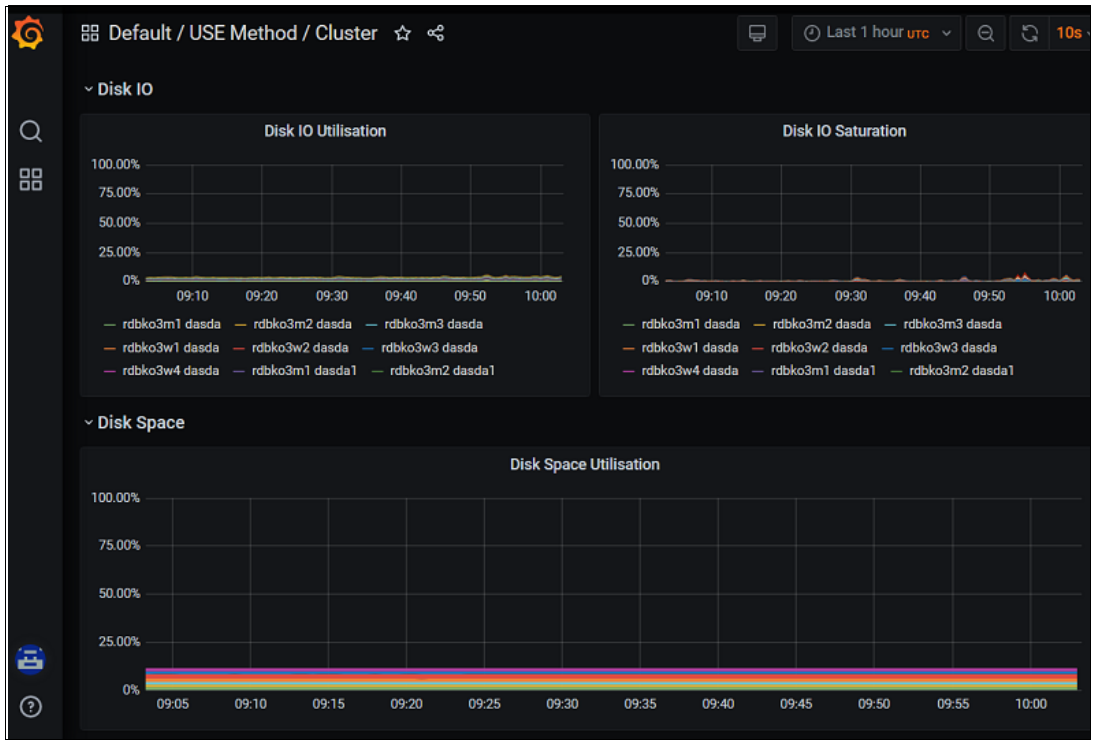

Figure 3-39 Checking the disk usage on the Grafana UI

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 3-40.

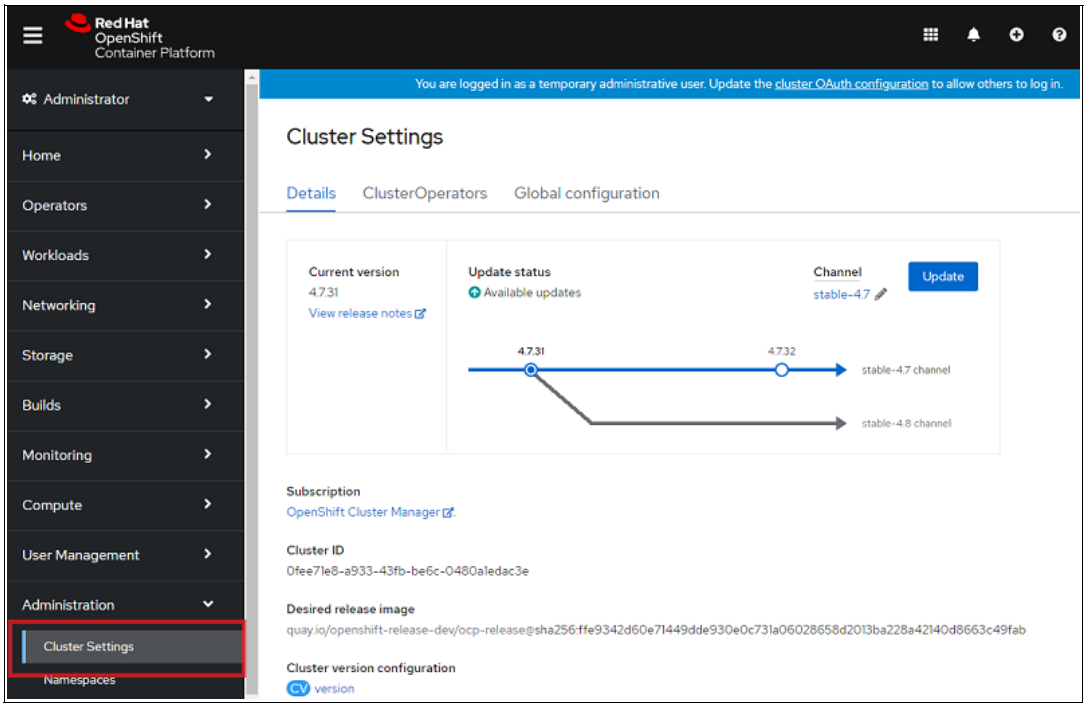

Figure 3-40 Navigating to the Cluster Settings window

2. Select the **Global configuration** tab and select **Alertmanager**, as shown in Figure 3-41.

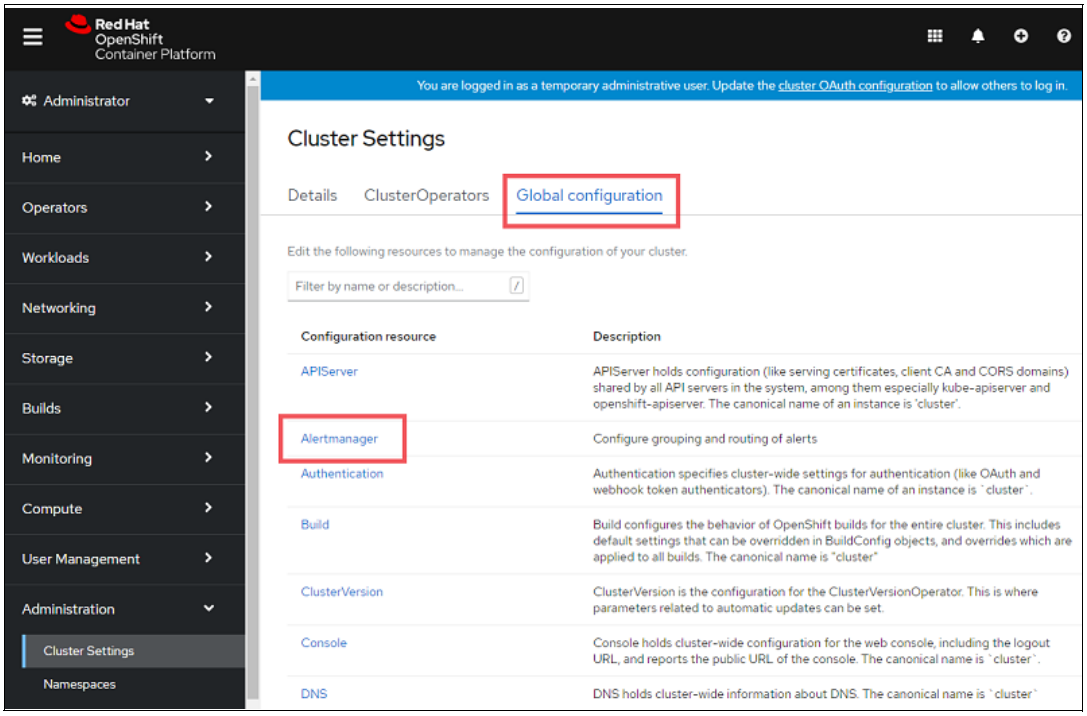

Figure 3-41 Selecting Alertmanager

3. Select **Create Receiver**, as shown in Figure 3-42.

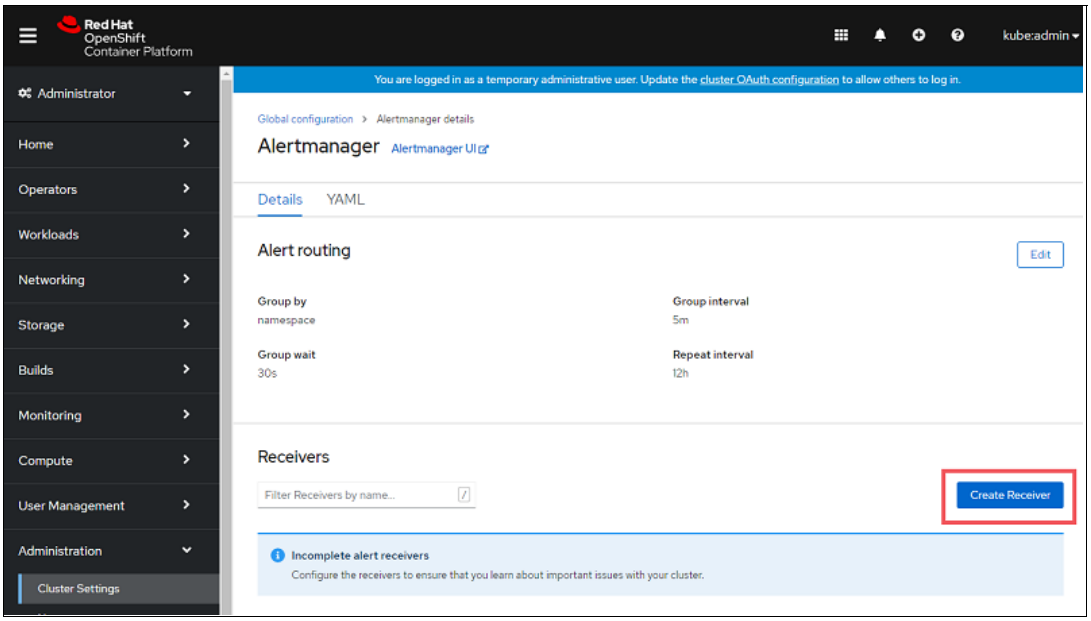

Figure 3-42 Selecting Create Receiver

4. Enter the **Receiver name** and select **Email** for the **Receiver type**, as shown in Figure 3-43. The detail fields appear, as shown in Figure 3-44.

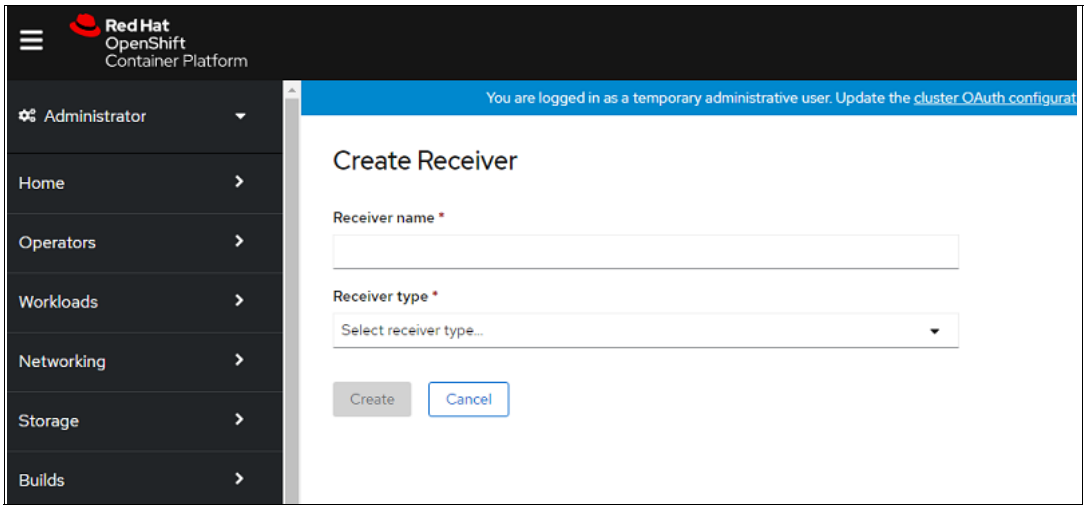

Figure 3-43 Creating an email receiver

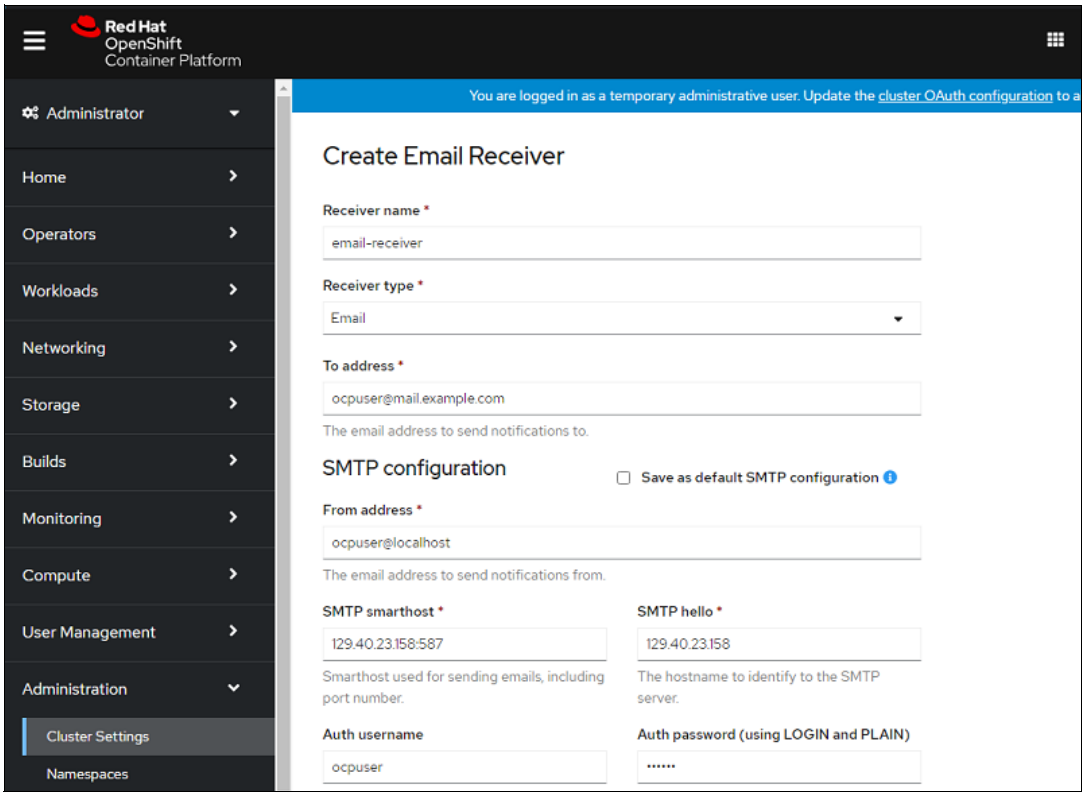

Figure 3-44 Detailed fields for an email receiver

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 3-45.

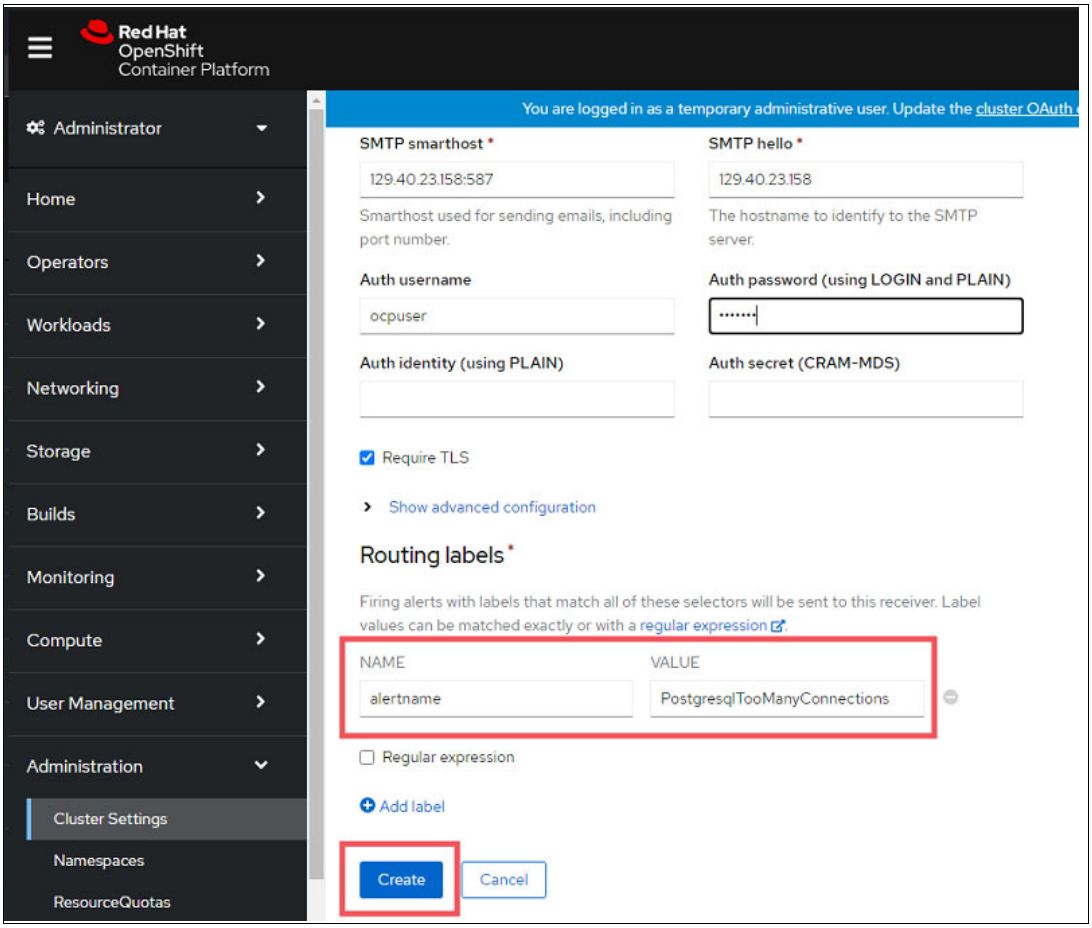

Figure 3-45 Routing alerts to the receiver

Confirming alert settings

Receivers are notified that the number of connections reached 90% of the maximum number of connections, as shown in Figure 3-46 on page 131.

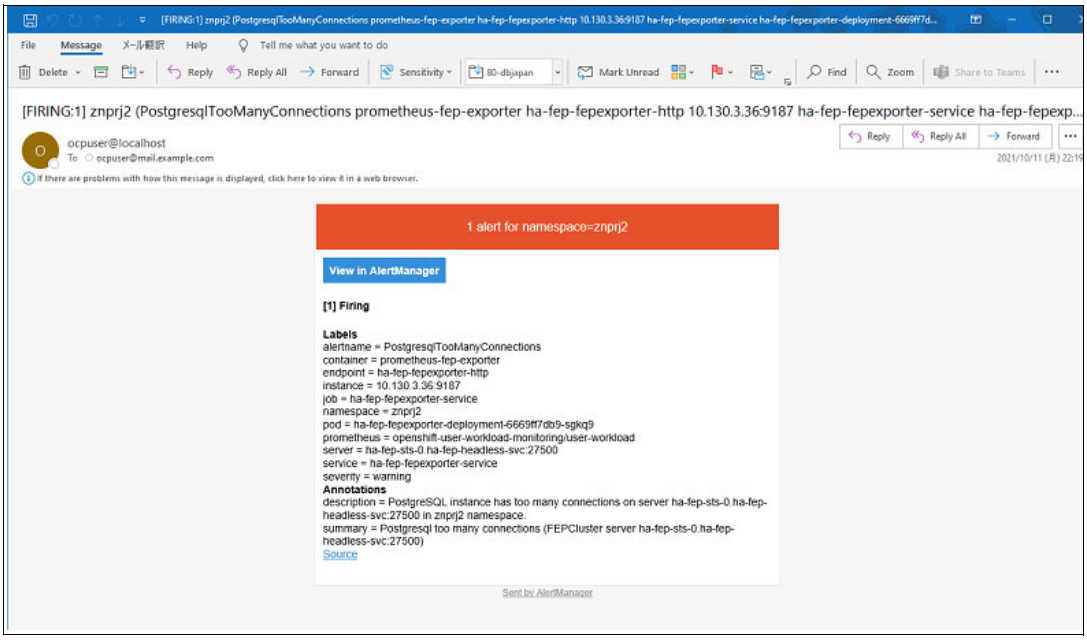

Figure 3-46 Example email notification

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. You can use this version to create and view custom Grafana dashboards.

   FUJITSU Enterprise Postgres provides a custom dashboard that displays metrics for the FUJITSU Enterprise Postgres cluster. In this example, we use this custom dashboard to check the metrics. Download the JSON file that defines the custom dashboard from the location that is indicated in the web page.

2. Click the dashboard icon on the Grafana UI, as shown in Figure 3-47.

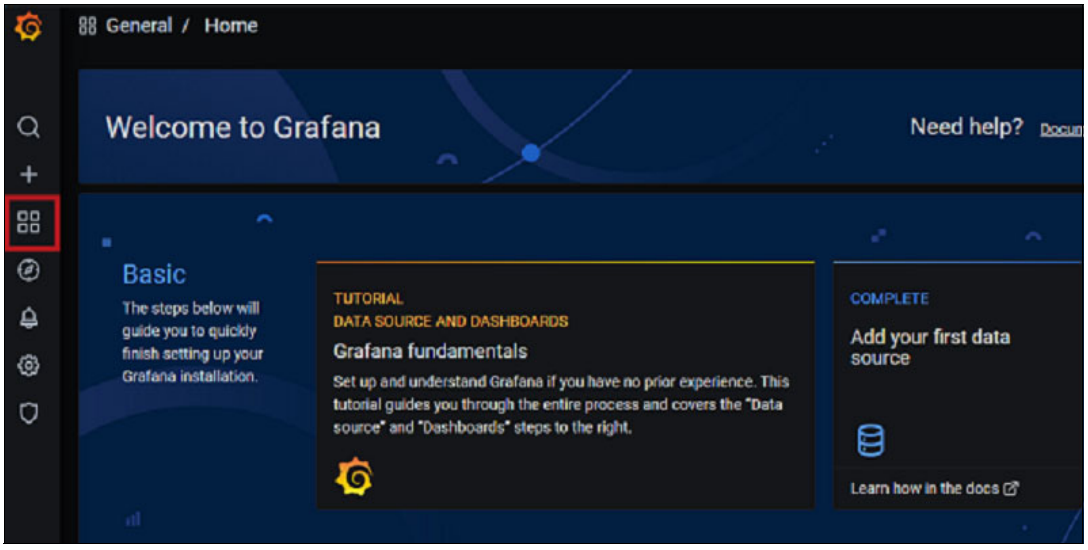

Figure 3-47 Selecting the dashboard icon on the Grafana UI home page

3. Select **Manage**, as shown in Figure 3-48.

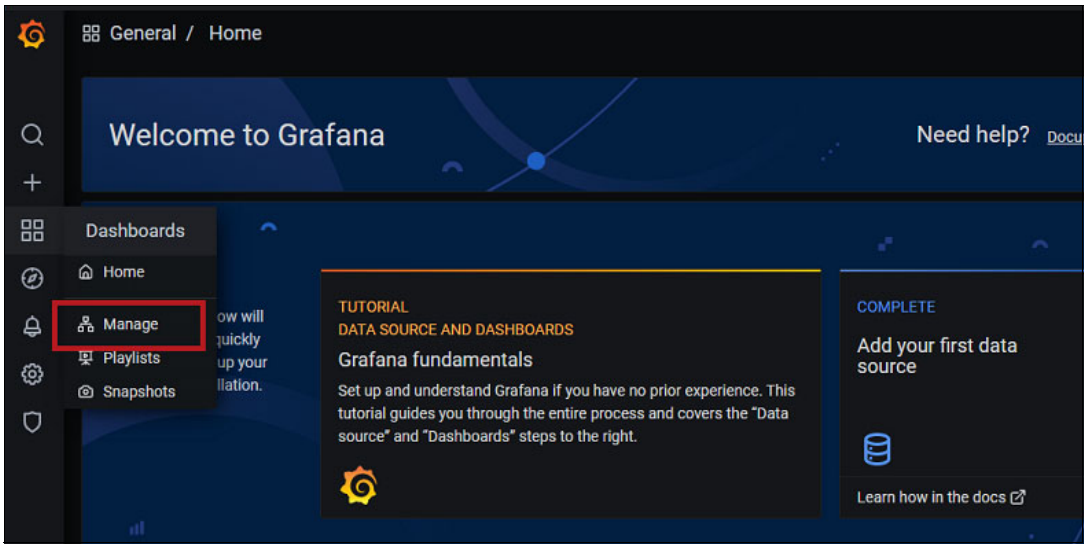

Figure 3-48 Selecting Manage

4. Click **Import**, as shown in Figure 3-49.

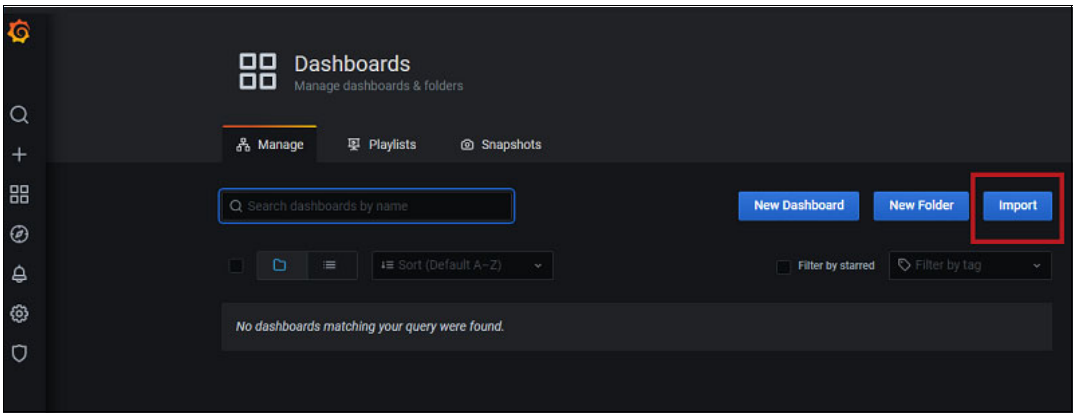

Figure 3-49 Selecting Import in the Grafana dashboard Manage window

5. Click **Upload JSON file**, and upload the JSON file that you downloaded in Step 2 on page 131, as shown in Figure 3-50 on page 133.

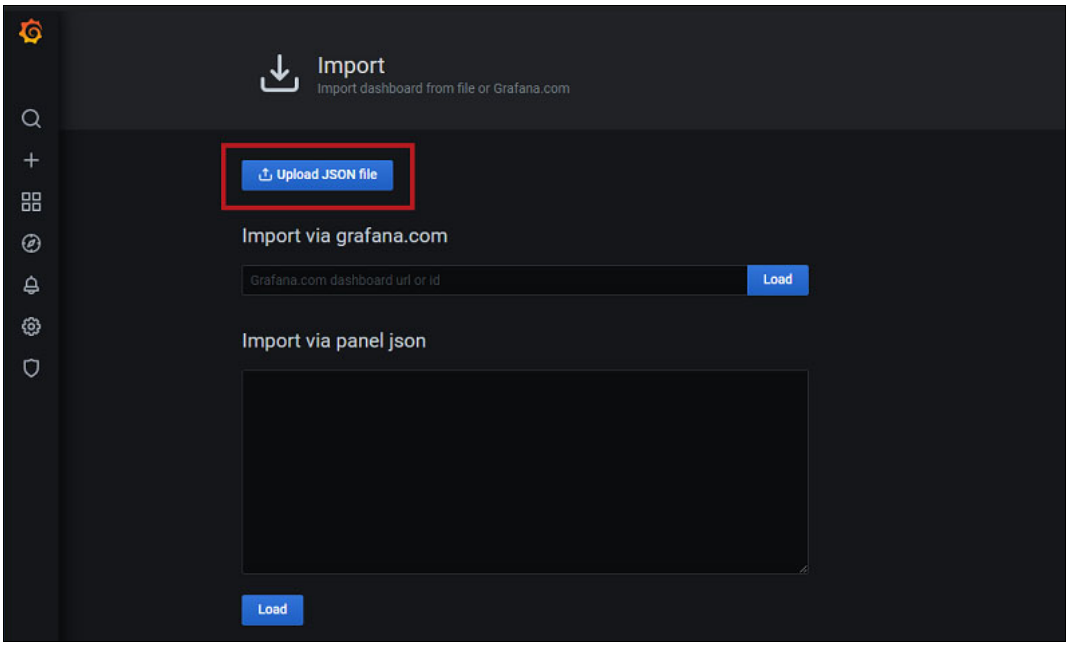

Figure 3-50 Uploading a custom dashboard

6. Click **Import**, as shown in Figure 3-51.

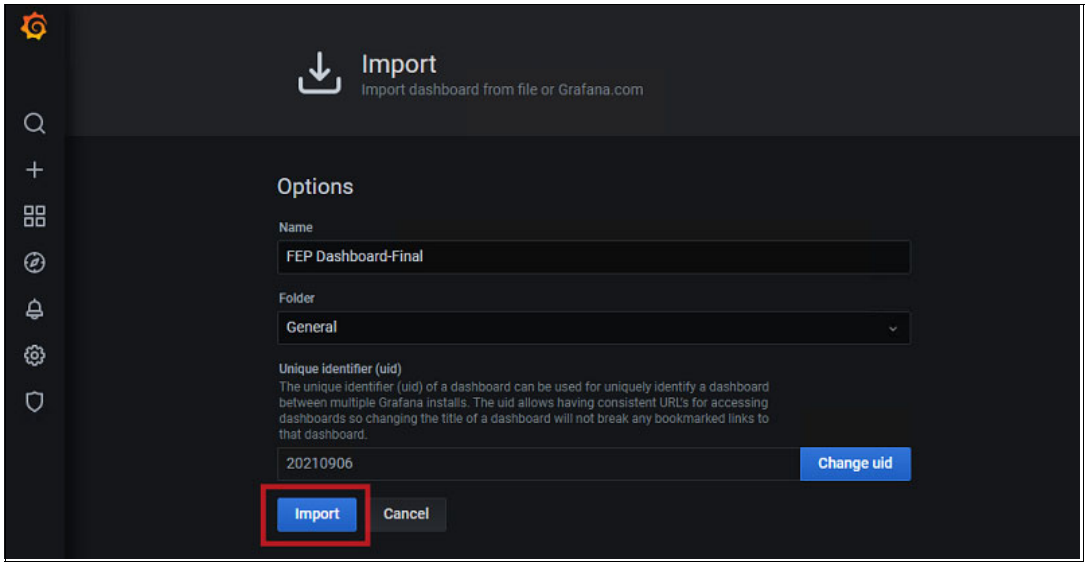

Figure 3-51 Importing a custom dashboard

7. A customized dashboard for FUJITSU Enterprise Postgres appears, as shown in Figure 3-52.

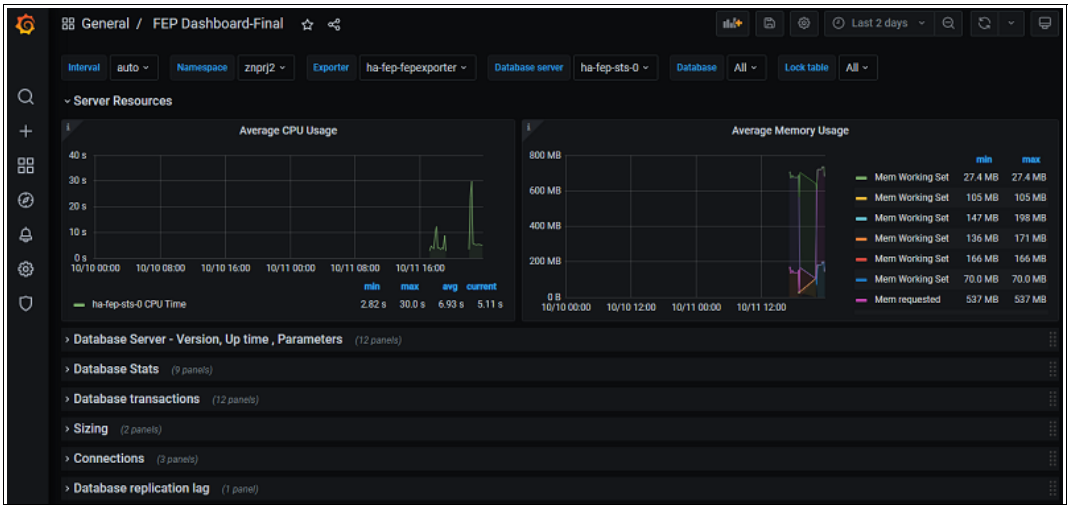

Figure 3-52 FUJITSU Enterprise Postgres custom dashboard

3.5 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.

3.5.1 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 3-3.

Example 3-3 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 3-53.

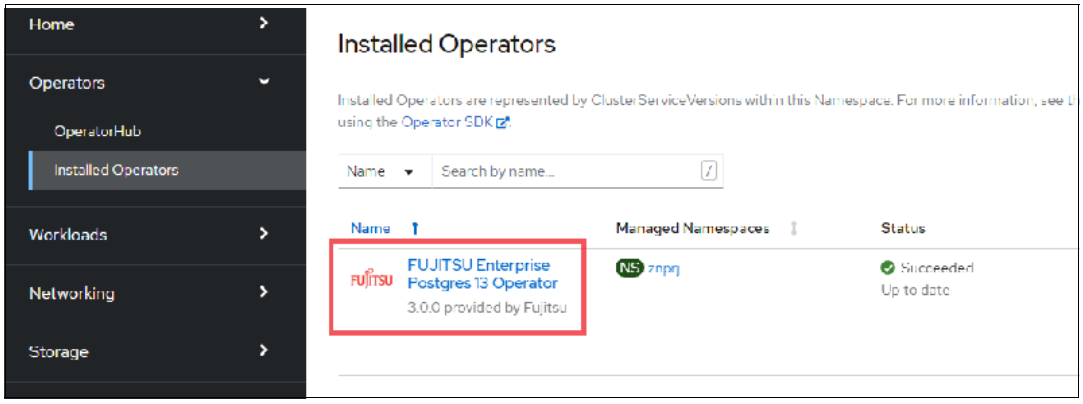

Figure 3-53 Selecting FUJITSU Enterprise Postgres 13 Operator

3. Go to the **FEPCluster** tab and select **ha-fep**, as shown in Figure 3-54.

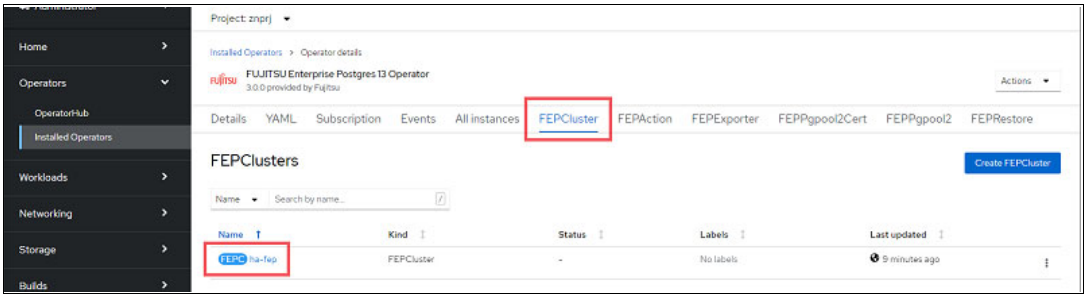

Figure 3-54 Selecting the name of FEPCluster

4. In the FEPCluster Details window, select the **YAML** tab and set the parameters that are shown in Table 3-5, as shown in Figure 3-55. 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 3-5 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.

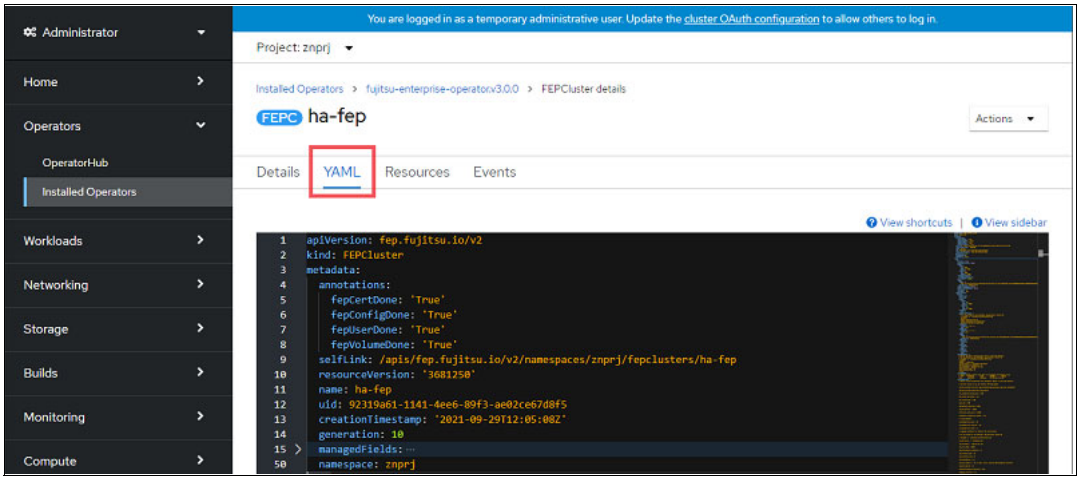

Figure 3-55 Updating the parameters

5. Click **Save** to apply the changes, as shown in Figure 3-56 on page 137.

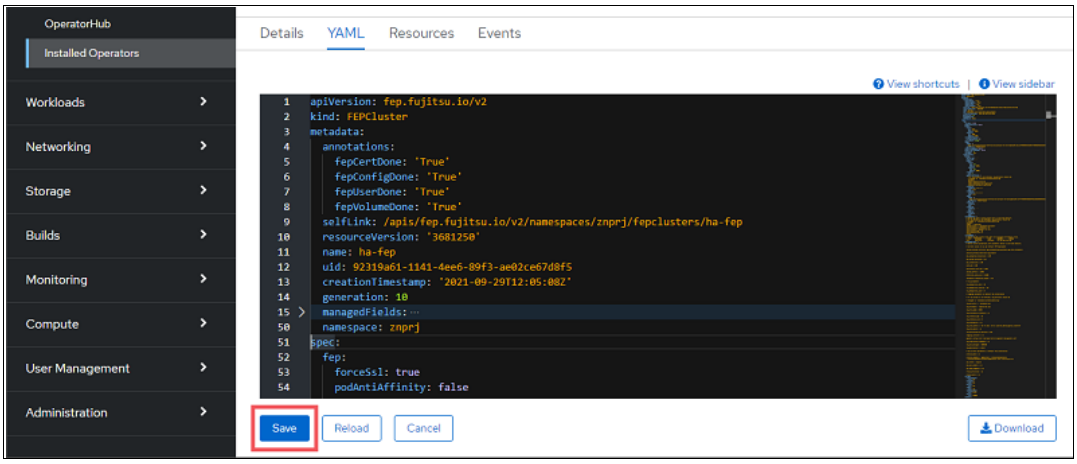

Figure 3-56 Saving the changes

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 3-57. Performance stability is ensured by expanded capacity even in the case of further increase of the workload.

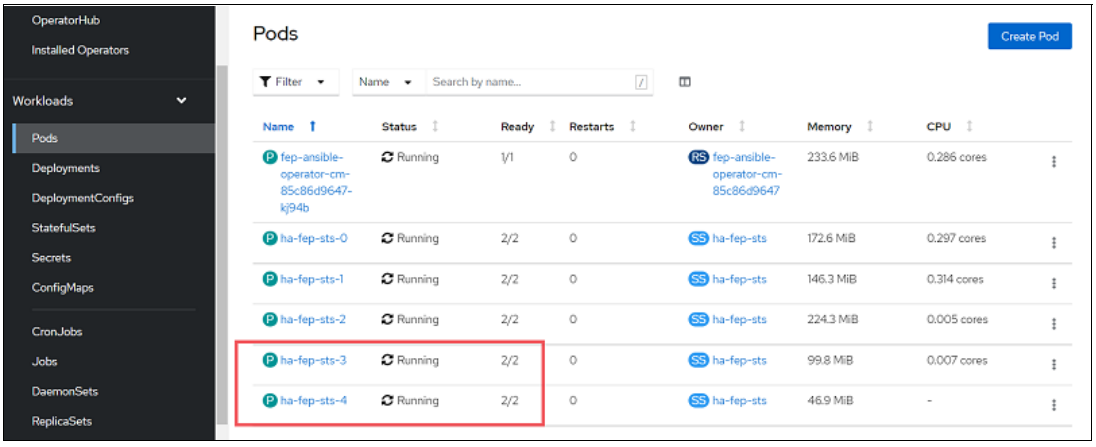

Figure 3-57 Checking the added pods

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*.

3.6 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.

3.6.1 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 3-58.

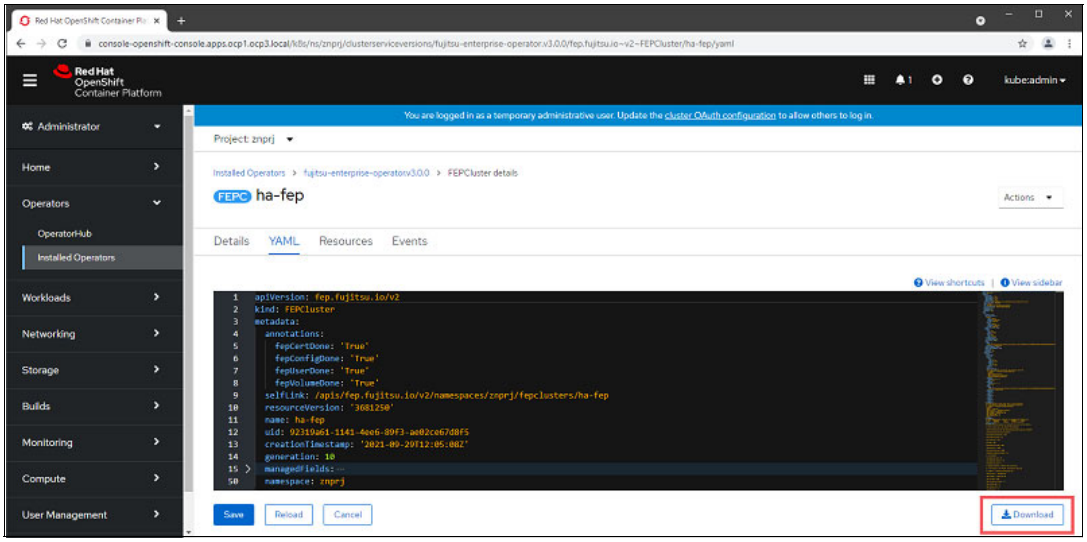

Figure 3-58 Downloading the CR configuration

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 3-59.

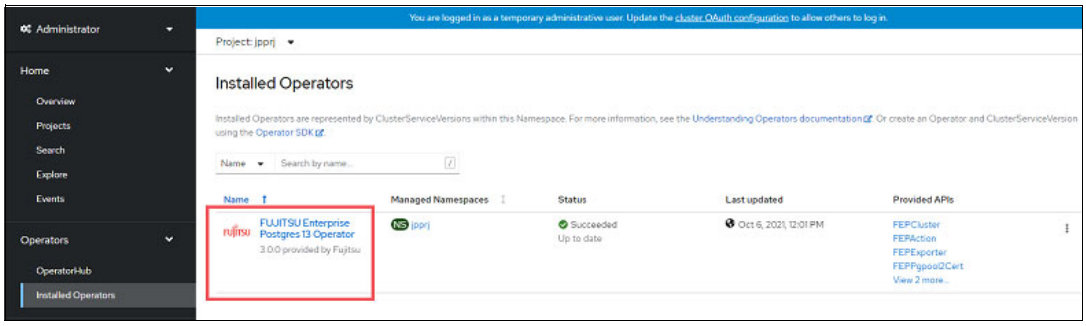

Figure 3-59 Selecting FUJITSU Enterprise Postgres 13 Operator

4. Click **Create Instance**, as shown in Figure 3-60.

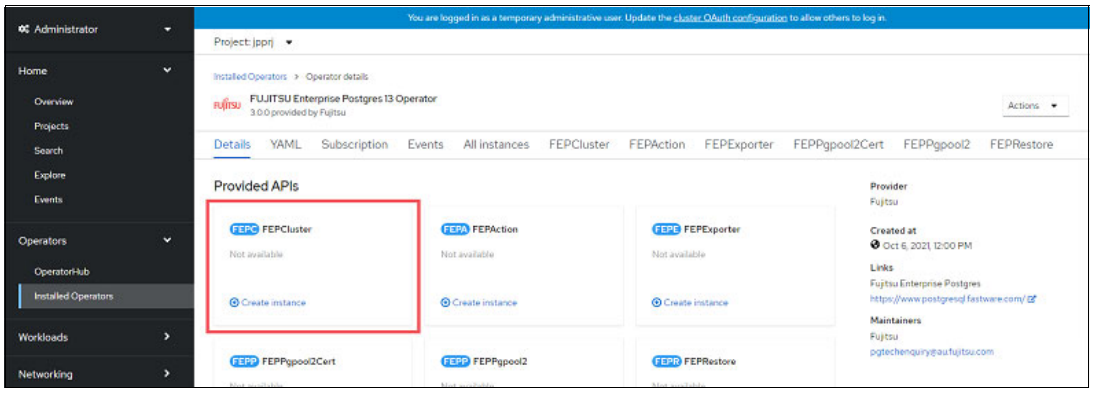

Figure 3-60 Creating a cluster

5. Copy the file that was downloaded on the existing system in step 1 on page 139 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 3-6. Click **Create** to create a cluster, as shown in Figure 3-61 on page 142.

Table 3-6 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.

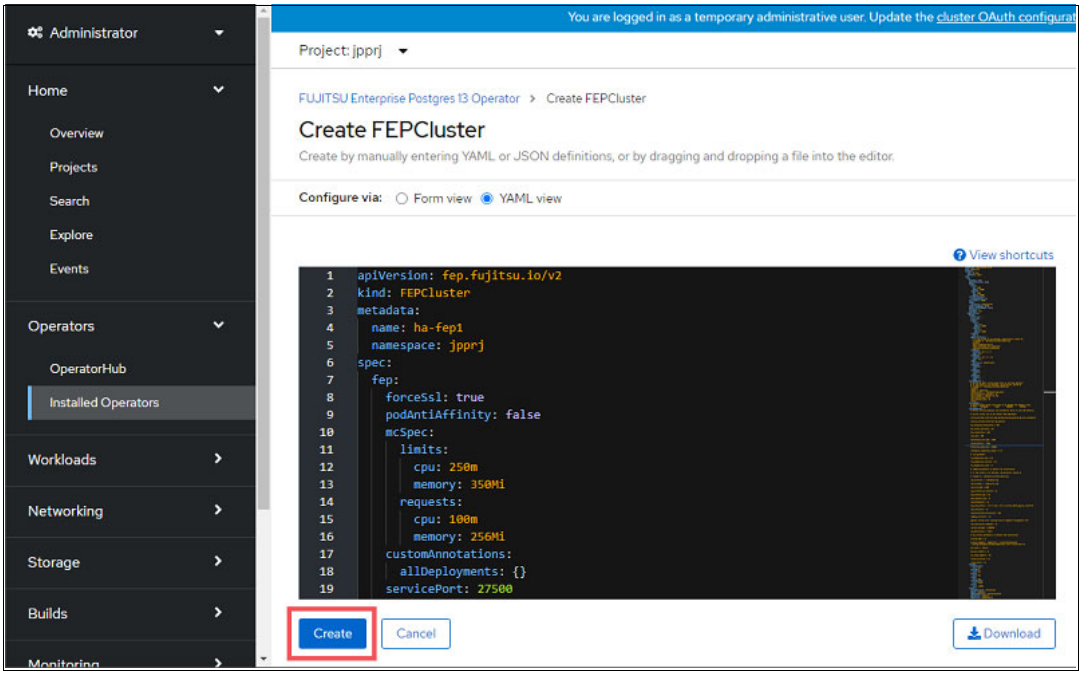

Figure 3-61 Deploying HA cluster

7. 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 3-62.

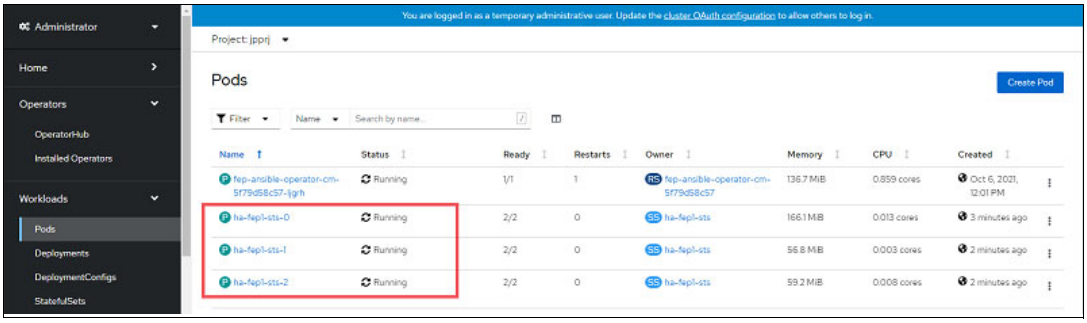

Figure 3-62 Checking the HA cluster deployment

3.6.2 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 144 is shown in Figure 3-63.

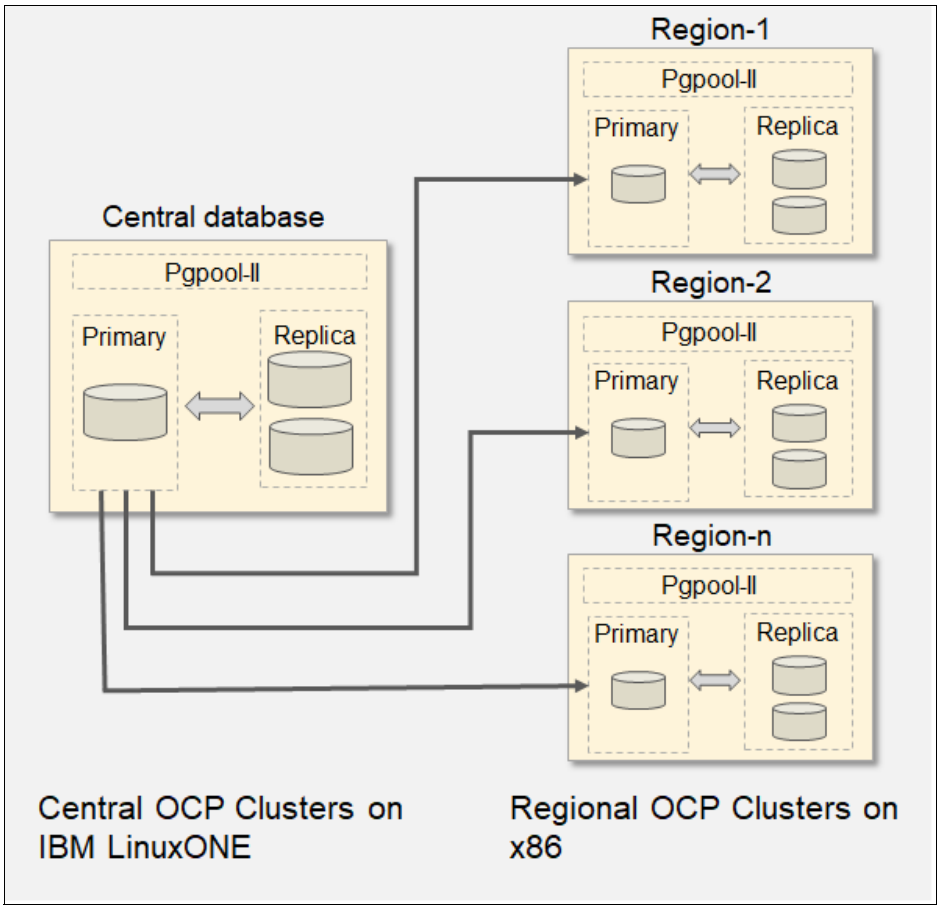

Figure 3-63 Sample expansion in the next step

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 3-64.

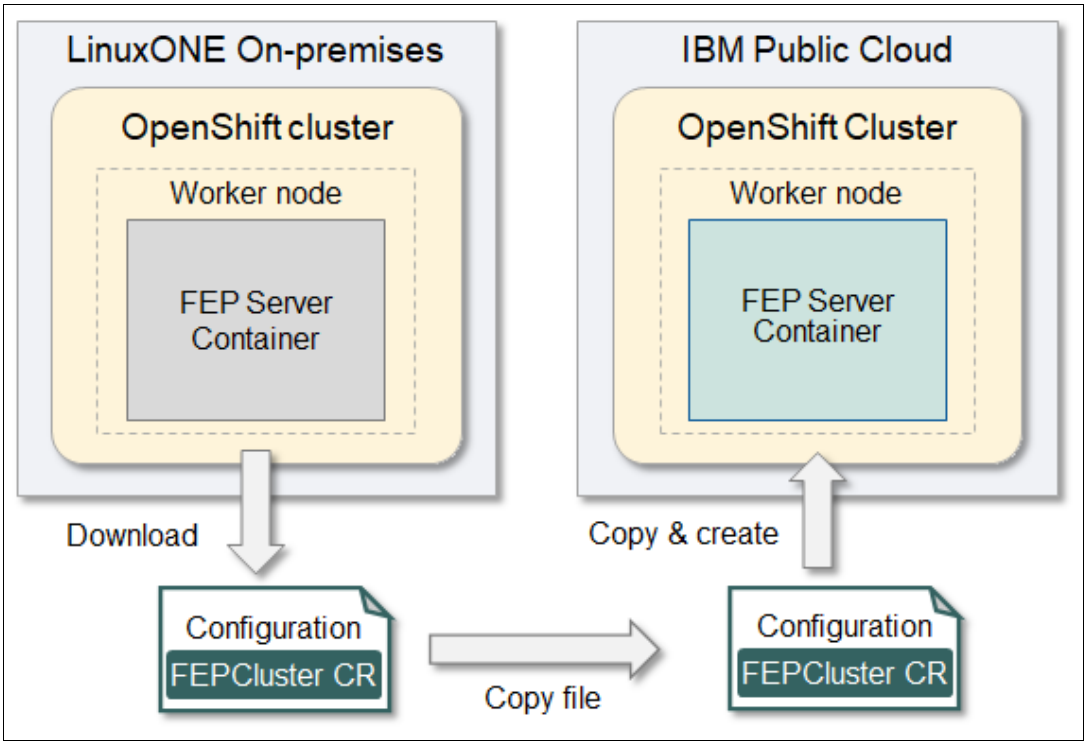

Figure 3-64 New system deployment overview

Note: To understand how to quickly deploy a new database by using an existing template, see 3.6.2, “Quick deployment of new databases for business expansion” on page 142.

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 3-65 on page 145.

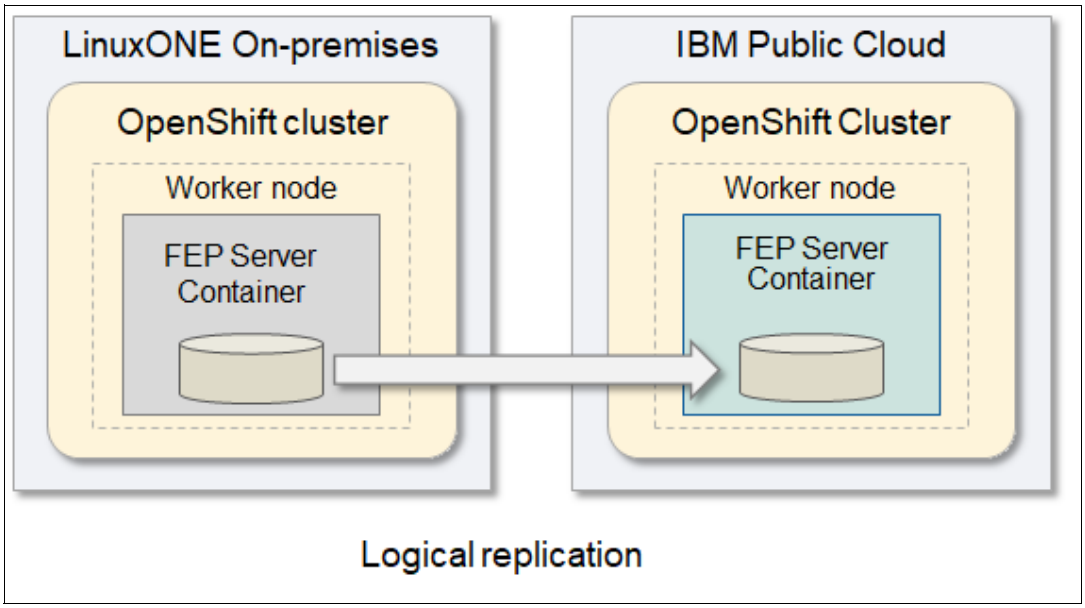

Figure 3-65 Overview of data sharing with logical replication

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 3-66.

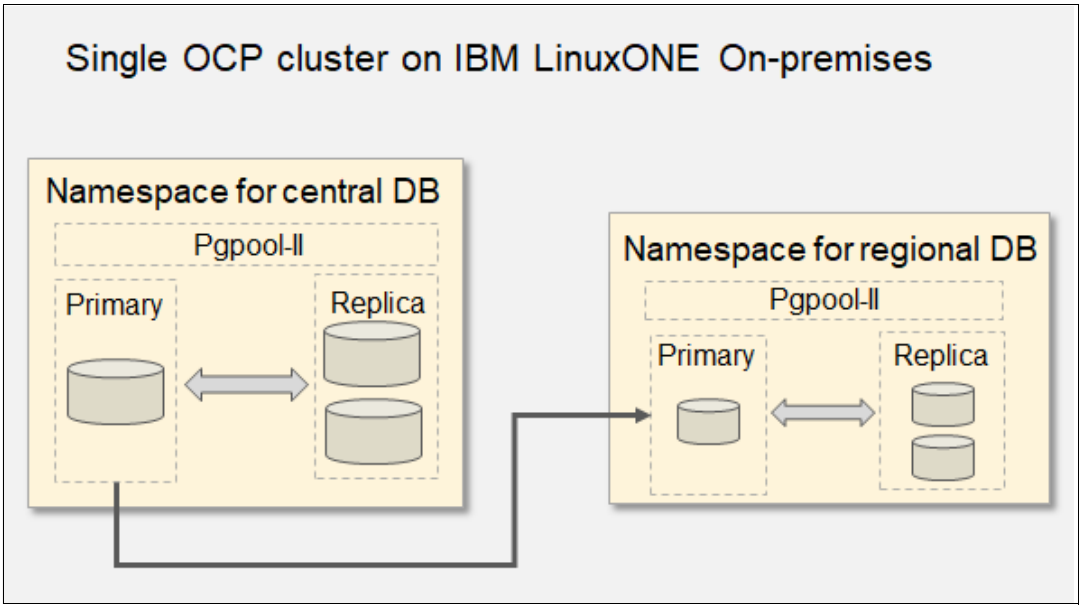

Figure 3-66 Use case in the next step

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 149 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 150 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 3-67.

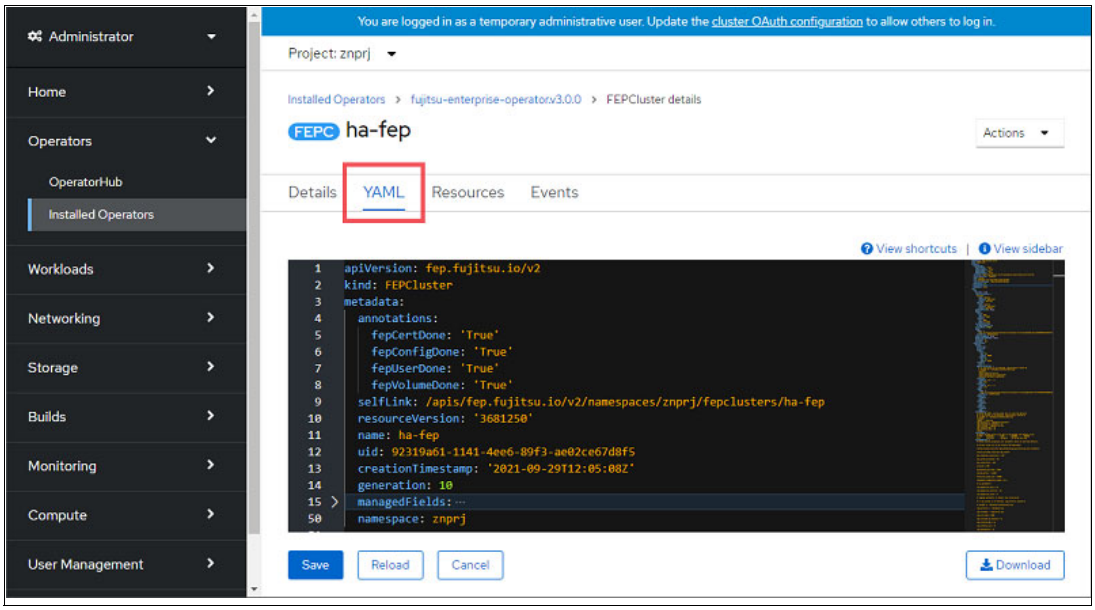

Figure 3-67 Changing the FEPCluster CR configuration of the publisher

   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 3-7.

Table 3-7 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 3.4.1, "Automatic backup" on page 106.
		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-o`, `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 3-8.

Table 3-8 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 3-9.

Table 3-9 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 3-10.

Table 3-10 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.

e. Save the changes that you made to FEPCluster CR configuration by clicking **Save** in the **YAML** tab, as shown in Figure 3-68.

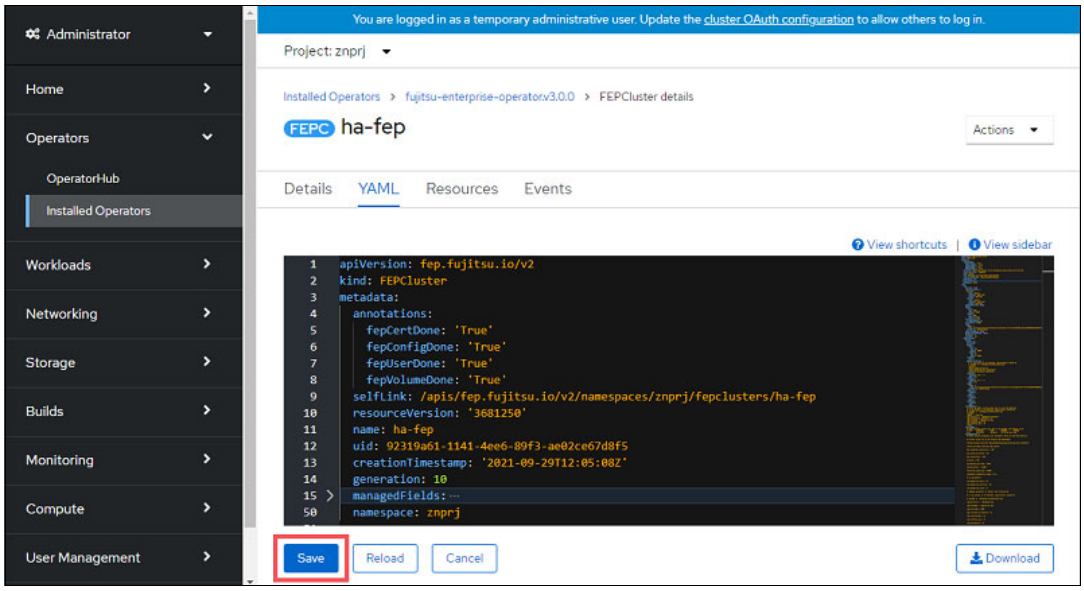

Figure 3-68 Saving the FEPCluster CR configuration changes

f. To reflect the parameter changes, restart PostgreSQL.

Note: A manual restart of the FUJITSU Enterprise Postgres process on all the FUJITSU Enterprise Postgres pods that use the FEPAction CR is required for changes to `postgresql.conf` to take effect. A restart causes a short outage on the cluster, so this action should be performed while considering a service interruption to the published cluster.

Select **Create instance** for **FEPAction**, as shown in Figure 3-69.

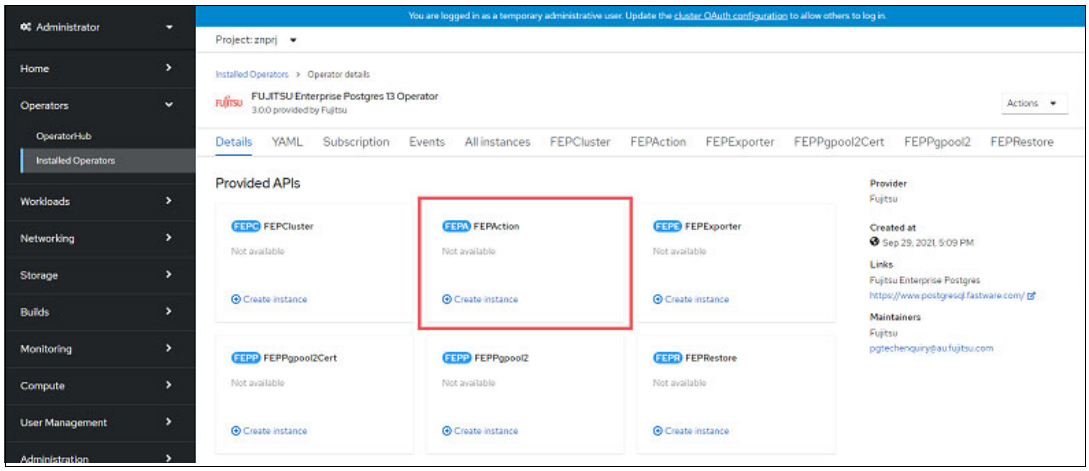

Figure 3-69 Selecting Create instance for FEPAction

Click the **YAML** tab and modify the values that are shown in Table 3-11 on page 149 and click **Create**, as shown in Figure 3-70 on page 149.

Table 3-11 FEPAction CR configuration parameter changes

Field	Value	Details
`metadata:`	`name: ha-fep-action` `namespace: znprj`	`name` is the object name of FEPAction CR. Specify a unique value among the same resource type (`kind: FEPAction`) within a namespace. `namespace` is the name of the namespace where the target FEPCluster for the restart is.

`spec:`
`  fepAction:`
	`args:`
	`  ha-fep-sts-o`
	`  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`.

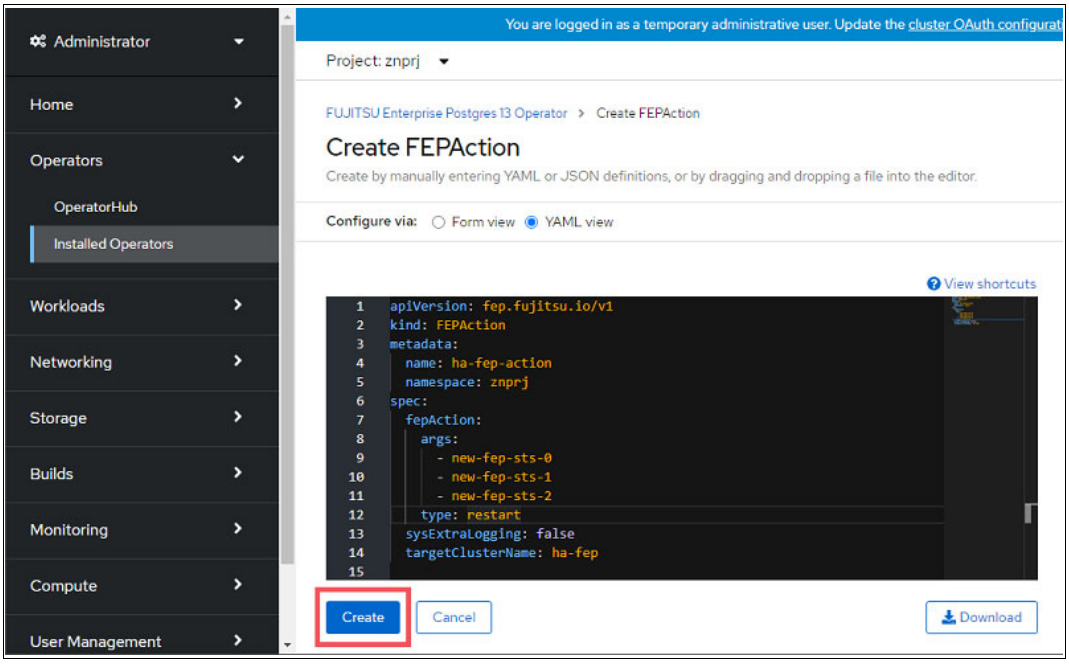

Figure 3-70 Restarting FEPCluster by creating a FEPAction instance

2. Now, a database that is named `publisher`, which was created in 3.4.1, “Automatic backup” on page 106, 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 3-4).

Example 3-4 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:

   a. From the FUJITSU Enterprise Postgres client, connect to `publisher`, as shown in Example 3-5.

   Example 3-5 Connecting to publisher

```
$ psql -h ha-fep-primary-svc -p 27500 -U postgres -d publisher
```

   b. 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 3-6.

   Example 3-6 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;
```

   c. Create a publication that is named `mypub`. Define the publication for the database and tables that will be replicated, as shown in Example 3-7.

   Example 3-7 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 3-7:

   - 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.

   d. Verify the publication that you created with the query that is shown in Example 3-8. The output of this command is shown in Example 3-9. To verify the publication, use the query that is shown in Example 3-10 on page 151.

   Example 3-8 Verifying the publication

```
publisher=# SELECT * FROM pg_publication_tables;
```

   Example 3-9 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 3-10 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 3.6.2, “Quick deployment of new databases for business expansion” on page 142, 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 3-71, add `customCertificates` under `spec.fepChildCrVal`, as shown in Table 3-12 on page 152. Click **Save** to save this change.

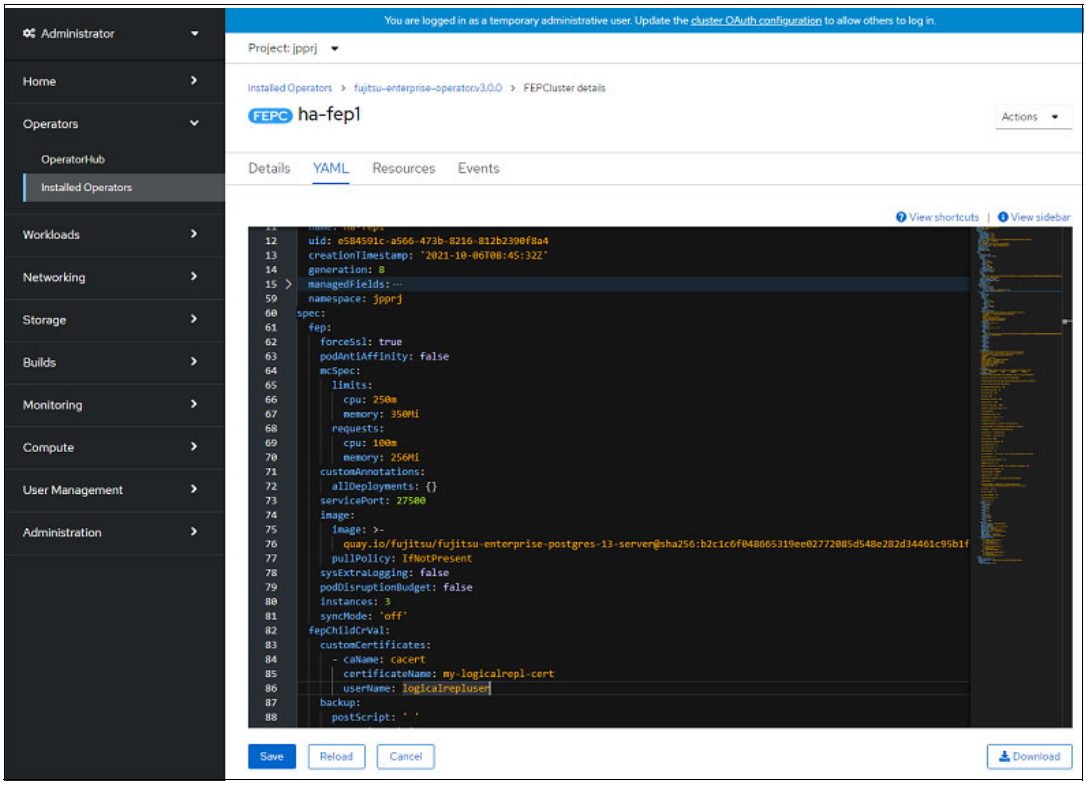

Figure 3-71 Changing the FEPCluster CR configuration

Table 3-12 FEPCluster CR configuration parameter changes

Field	Value	Details

spec:
  fepChildCrV
al:
	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 3-11.

Example 3-11 Creating a database

```
$ createdb -h ha-fep1-primary-svc -p 27500 -U postgres subscriber
```

3. 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 3-12.

Example 3-12 Creating tables

```
$ psql -h ha-fep1-primary-svc -p 27500 -U postgres -d subscriber -f
/tmp/publisher.schema.dump
```

4. 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 3-13.

Example 3-13 Connecting to the database

```
$ psql -h ha-fep1-primary-svc -p 27500 -U postgres -d subscriber
```

5. Define a subscription by using the command that is shown in Example 3-14. 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 3-14 Defining a subscription

```
subscriber=# CREATE SUBSCRIPTION mysub CONNECTION
'host=ha-fep-primary-svc.jpprj.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);
```

6. Check the created subscription with the command that is shown in Example 3-15. A sample output is provided in Example 3-16 on page 153.

Example 3-15 Checking the subscription

```
subscriber=# SELECT * FROM pg_subscription;
```

Example 3-16 Sample output of pg_subscription

```
oid  | subdbid | subname | subowner | subenabled | subconninfo | subslotname | subsynccommit |
subpublications
-------+---------+---------+----------+------------+-----------+-------------+---------------+-------------
 16466 |   16446 | mysub   |       10 | t          | host= ha-fep-primary-svc.jpprj.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 3-17 Checking the logical replication of pgbench_accounts

```
subscriber=# SELECT COUNT(*) FROM public.pgbench_accounts;


count
---------
 1000000
(1 row)
```

Example 3-18 Checking the logical replication of pgbench_branches

```
subscriber=# SELECT COUNT(*) FROM public.pgbench_branches;


count
-------
    10
(1 row)
```

Example 3-19 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.

4

Application use cases: Geospatial processing

This chapter 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 4-1 shows typical use-cases involving financial, healthcare, retail, energy, commercial and environmental studies.

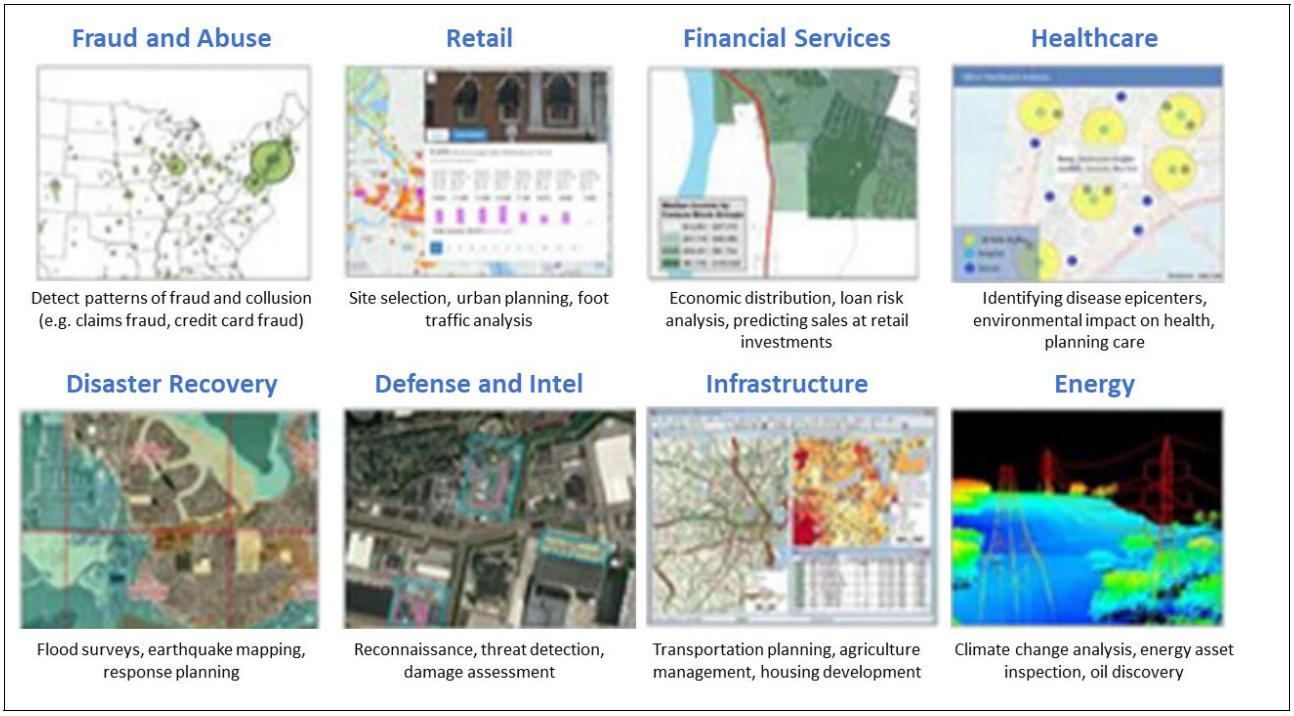

Figure 4-1 Geospatial data source examples

4.1 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 4-2 on page 157 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 4-2 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.

4.2 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.

4.3 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.

4.3.1 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 4-3 transform data types from one format to another one.

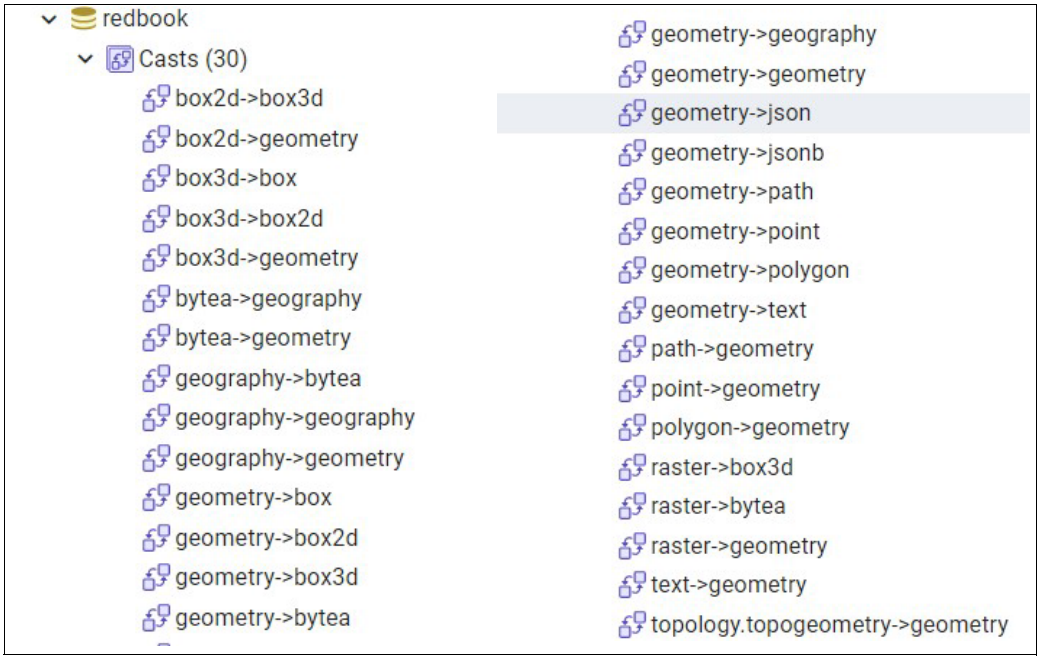

Figure 4-3 Casts showing data type transformations

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.

4.3.2 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 4-1 shows an example of using the function ST_DWithin, and Figure 4-4 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 4-1 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 character varying (30)	total 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 4-4 PostGIS function example of ST_DWithin

4.3.3 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.

[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_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.
- 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 4-2 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 4-2 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 4-3.

Example 4-3 Results of STBuildArea

```
Result dataset (2,146 chars)
"00000000030000000200000214062C0000……
DB010CD765405F4000000000004056800000000000"
```

The resultant graph is a “donut”, as shown in Figure 4-5.

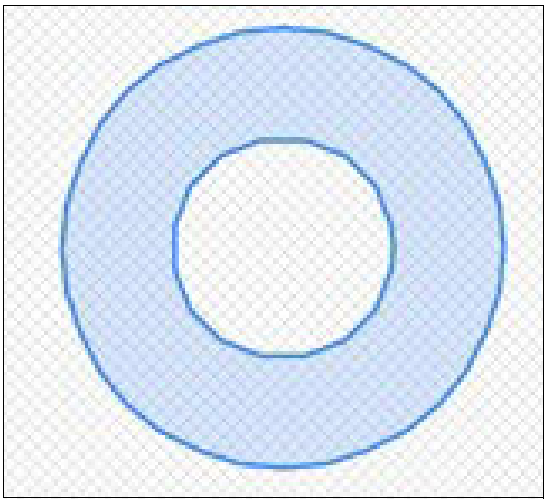

Figure 4-5 PostGIS function ST_BuildArea

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.

4.3.4 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 4-6 on page 165 provides an example of vector and raster grid models.

[2] https://spatialvision.com.au/blog-raster-and-vector-data-in-gis/

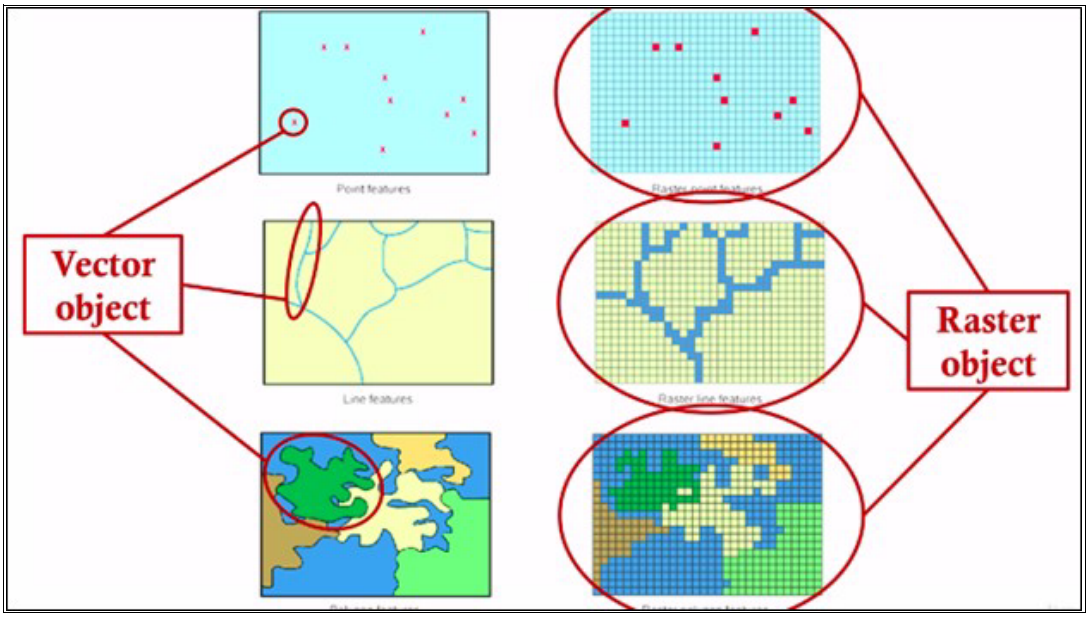

Figure 4-6 Vector and raster grid models

- 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 i 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 4-7 shows raster data that is overlaid onto maps.

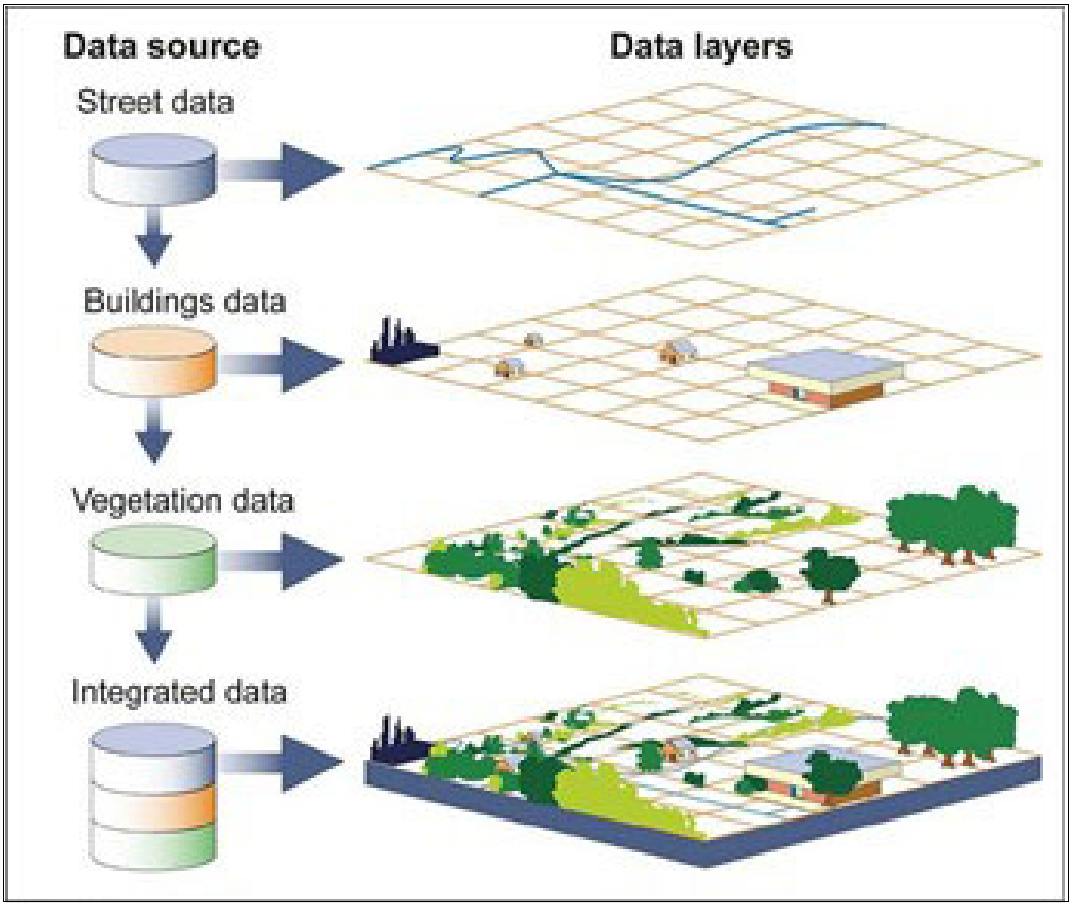

Figure 4-7 Raster data overlaid onto maps

4.3.5 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.

4.3.6 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.

4.4 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 4-8 shows a cycle network analysis that was done by Space Syntax for the Department for Transport.

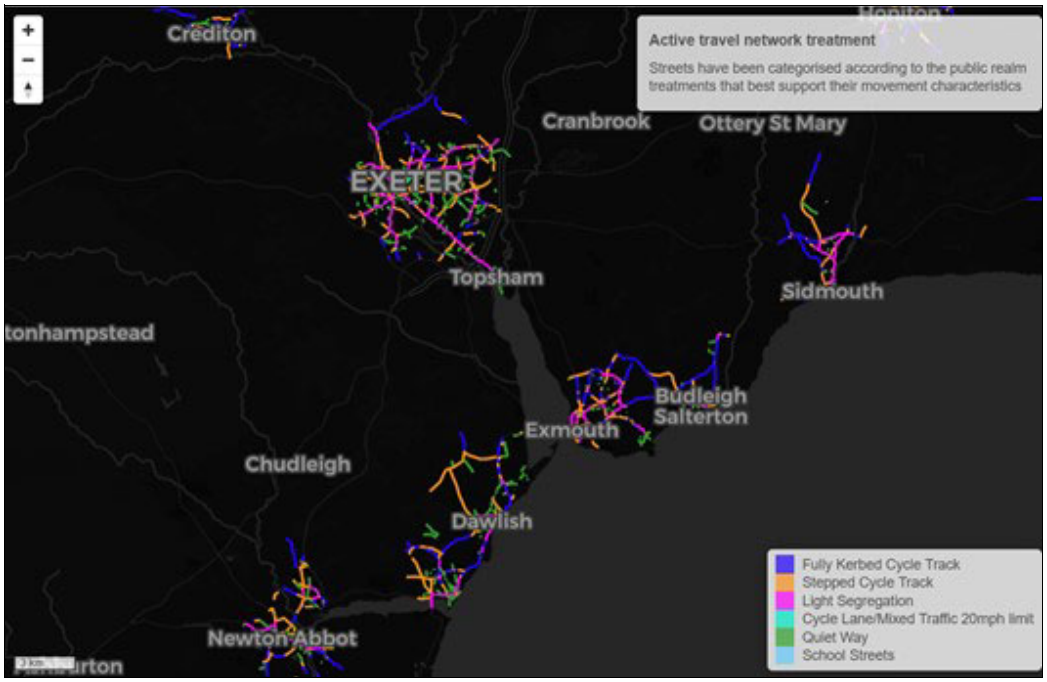

Figure 4-8 Space Syntax: Department for Transport cycle network analysis

Using a combination of queries that uses PostGIS spatial functions, Figure 4-9 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.

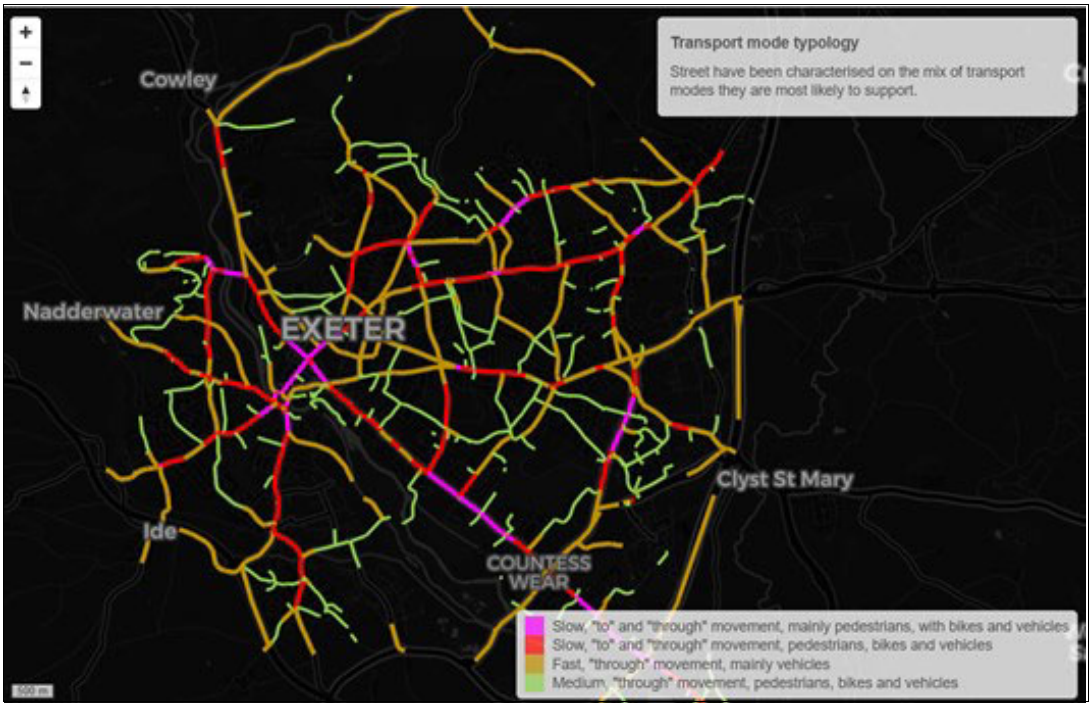

Figure 4-9 Space Syntax: Cycle network that is categorized according to character of movement

The queries that are shown in Figure 4-10 on page 169 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.

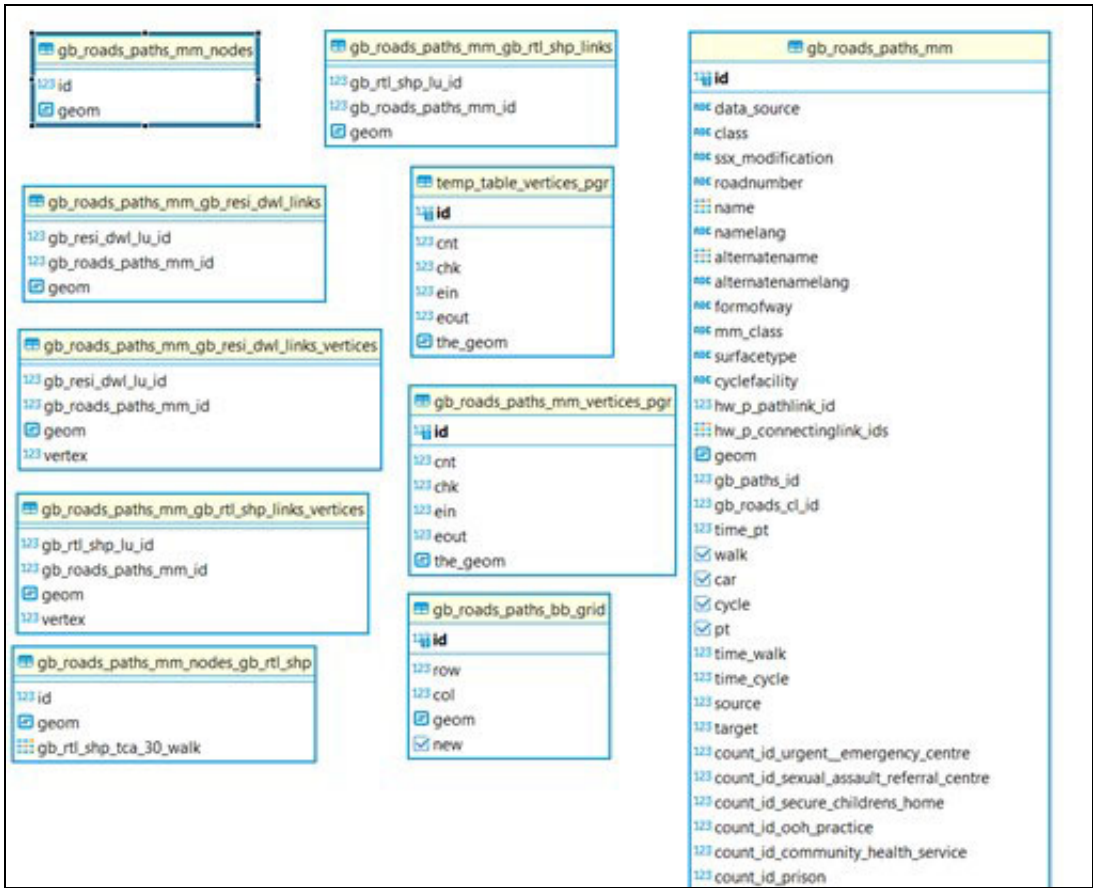

Figure 4-10 UK-based geospatial data model

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 4-4 on page 170, 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 4-4 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 4-4 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  l
          ON o.lu_id = l.'|| 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 4-11 shows the results of the program.

#	Node	Plan
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 4-11 Query results

The results were captured as data sets from which individual locations were transposed into graphical representations (Figure 4-12).

tile_id	geom	chunk_seq	chunk_size	count_origin	error	minutes	id
114	0020000003000006C340000001000000054	1	2500	1179		2.008836	1
114	0020000003000006C340000001000000054	2	2500	1147		2.792562	2
114	0020000003000006C340000001000000054	3	2500	1389		5.370199	3
114	0020000003000006C340000001000000054	4	2500	1407		6.513321	4
114	0020000003000006C340000001000000054	5	2500	1554		6.451666	5

Figure 4-12 Results data set

4.4.1 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.

5

MongoDB as a service with Linux on IBM Z

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 provides[1].

[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 5-1 shows the sample comparison that is used for this chapter.

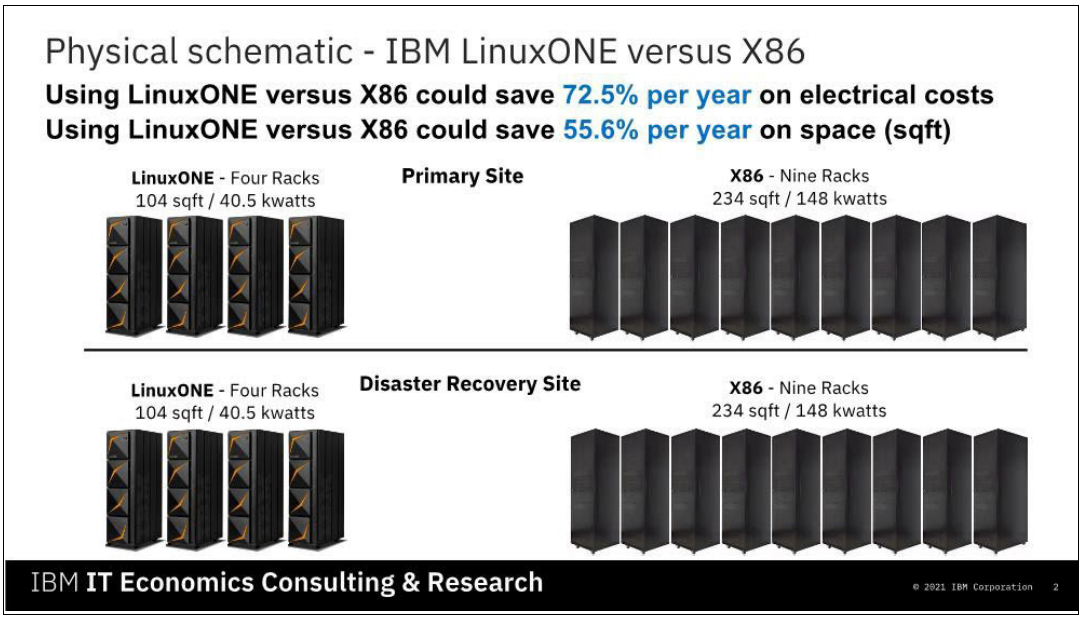

Figure 5-1 IBM LinuxONE versus x86

5.1 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.

Figure 5-2 shows a representation of the emulated data center.

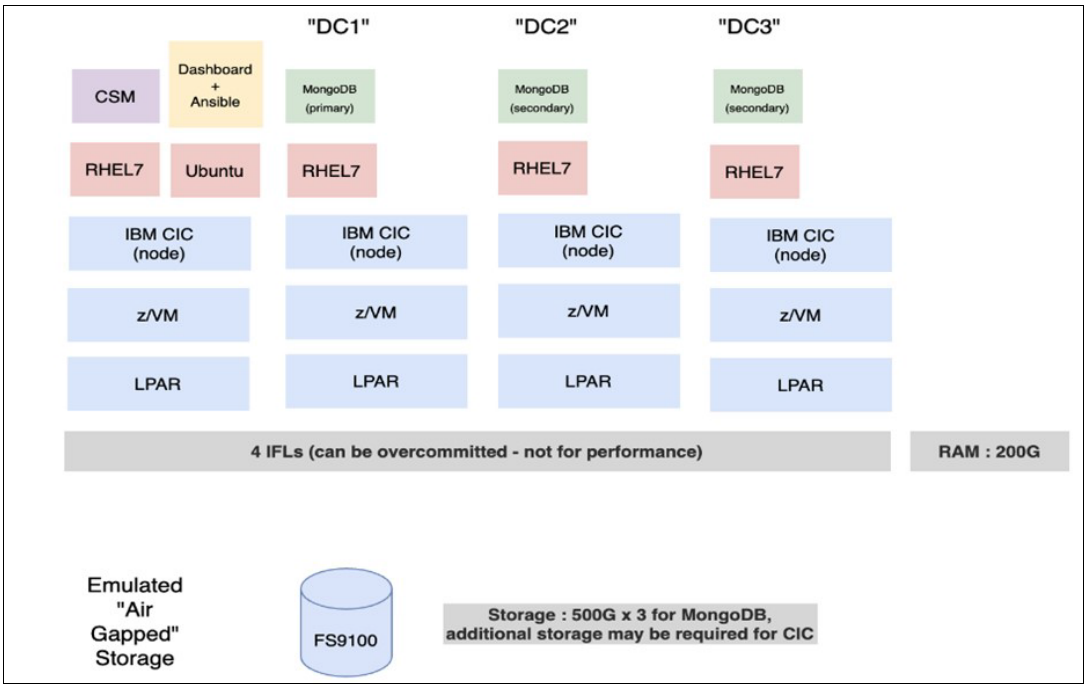

Figure 5-2 Emulated data center

Figure 5-3 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.

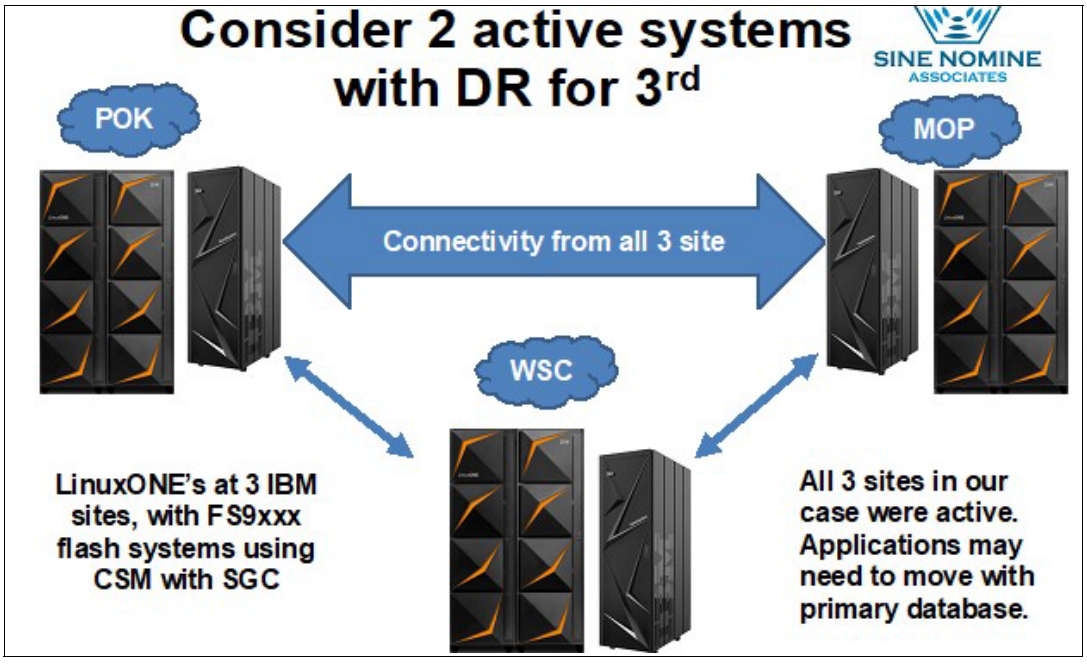

Figure 5-3 Expanded lab environment

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 5-4 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 MondoDB 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.

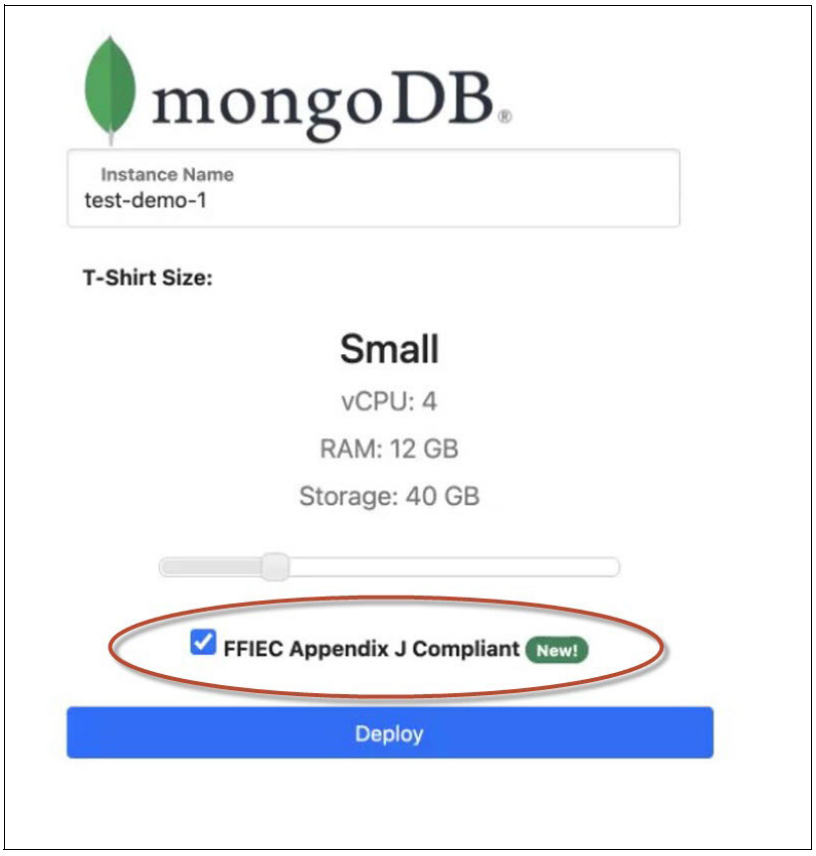

Figure 5-4 Selecting the Appendix J Compliant checkbox

5.1.1 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.

5.1.2 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 5-5 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.

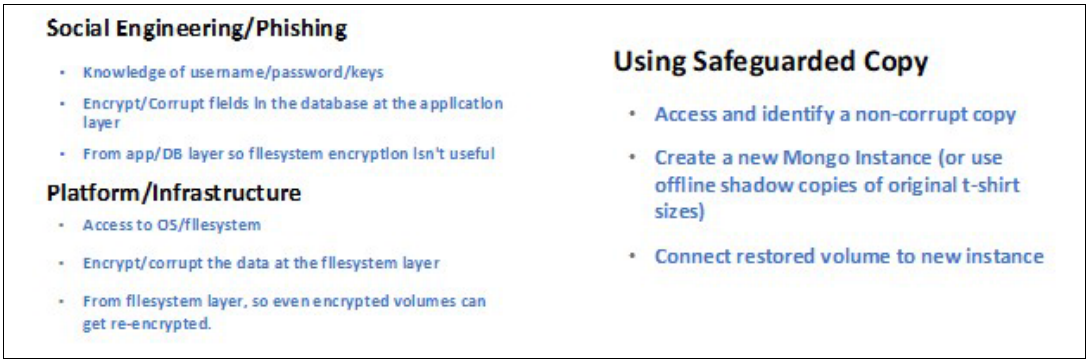

Figure 5-5 Main types of attacks with recovery efforts

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 5-6 on page 179 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.

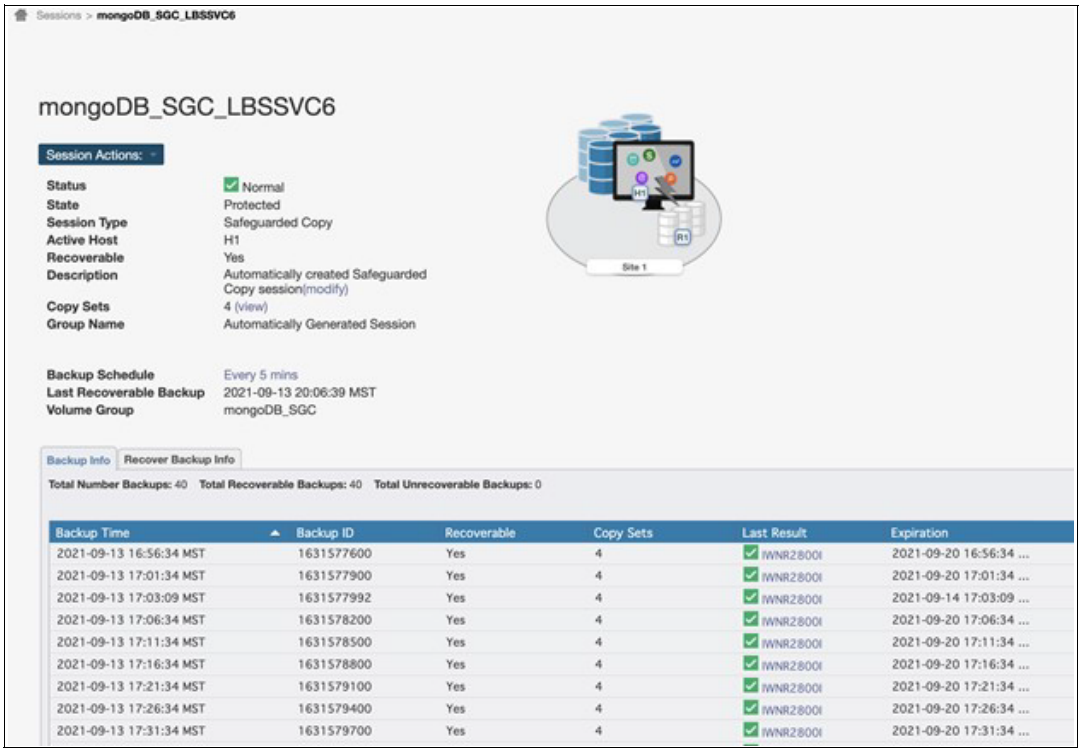

Figure 5-6 IBM Copy Services Manager with 5-minute frequency and 1-day retention

5.2 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 (5.2.1, “Base image deployment overview” on page 180).
2. Deploying a set of virtual machines (VMs) that make up a replica set (5.2.2, “Replica set virtual machine instantiation overview” on page 180) or a shadow set of VMs (5.2.3, “Shadow overview” on page 180).
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).

5.2.1 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
  ```

5.2.2 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 5.4.5, "Shadow instance deployment" on page 193.

The Ansible configuration consists of a playbook and a set of tasks that is repeated for each of the VMs that are being provisioned.

5.2.3 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 5-1 provides a list of the pros and cons of each option.

Table 5-1 Comparison table of the three options:

Mechanism	Pros	Cons
Reuse VMs.	► Single set of VMs. ► No networking changes are required.	► 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.	► Time-consuming (< 5 minutes) but reasonable given longer RTO windows.
Shadowing.	► A clean state that is clear of malware. ► Quicker than spinning up a fresh VM because starting the instances is near instantaneous.	► Networking addresses or names need changing to match what the Mongo data expects.

5.3 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.

5.3.1 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 5-1.

Example 5-1 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 5.4.6, “Hosts file” on page 195. The file is named after the replica set. This file is used to create shadows, terminate the replica set, and destroy the replica set.

5.3.2 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 (5.3, “MongoDB deployment overview” on page 181) 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 5-2 to create `mongodb_proc_net.te`.

Example 5-2 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 5-3.

Example 5-3 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 5-4 is generated by a script that is invoked from the playbook and used to initiate the replica set.

Example 5-4 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 5-5 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 5-5 admin.js

```
try {
    rs.secondaryOk()
    db = connect('admin')
    adminUser = db.system.users.find({user:'admin'}).count()
    if (adminUser == 0) {
        db.createUser( {
                user: "admin",
                pwd: "xxxxxxxx",
                roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
            }
        )
    }
} catch {
}
```

Other playbooks

Here are some other relevant playbooks:

- Quiesce

  The quiesce playbook (5.5.3, “Quiescing” on page 203) locks the database to prevent it from being updated. You are prompted for the admin password.

- Resume

  The resume playbook (5.5.4, “Resuming” on page 204) unlocks the database to allow updates to proceed. You are prompted for the admin password.

- Shutdown

  The shutdown playbook (5.6, “Terminating” on page 204) uses `systemd` to gracefully shut down a replica set.

5.3.3 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 5.6, “Terminating” on page 204.

5.4 Playbooks

This section provides an annotated description of the playbooks that are used to create images and deploy MongoDB replica sets.

5.4.1 Base deployment

This playbook creates a base image for use by other deployment processes. Example 5-6 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 5-6 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 5-6 on page 185 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.

5.4.2 Parameters

The JSON file that is shown in Example 5-7 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 5-7 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_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 5-7 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.

5.4.3 Replica set virtual machine instantiation

A replica set consists of three or more VMs. They are created by using a master playbook (Example 5-8) that invokes an image and a task playbook (Example 5-9 on page 190) 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 5-10 on page 191.

Example 5-8 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 5-8 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
}}:{{ 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 5-8 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 5-9 contains the tasks that are required to provision a single VM and data volume.

Example 5-9 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 5-9 on page 190 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 5-10 provides the variables that are required by the `deploy-hosts.yml` playbook and `deploy-image.yml` tasks.

Example 5-10 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 5-10 on page 191 and their descriptions:

- [001]: Name of the nodes and hostnames.
- [002]: Flavor of image that will 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.

5.4.4 OpenStack support

The file that is shown in Example 5-11 is used by the OpenStack module to authenticate against the IBM CIC host.

Example 5-11 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].crt
[009]     interface: public
[010]     identity_api_version: 3
[011]     domain_name: default
```

Here are the line numbers from Example 5-11 on page 192 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.

5.4.5 Shadow instance deployment

If you choose to use a shadowing backup and recovery mechanism, the playbooks that are shown in Example 5-12 and Example 5-13 on page 194 may be used to create “shadows” of a specified MongoDB replica set (Example 5-12) by using the hosts file (Example 5-13 on page 194) that is created when the replica set was created.

Example 5-12 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
}}:{{ 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 5-13 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}} mkvolumegroup -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
}}' } ]"
```

5.4.6 Hosts file

The file that is shown in Example 5-14 is created by the deployment playbooks and contains all the information that is needed for a successful deployment.

Example 5-14 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 5-14 on page 195, 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.

5.5 MongoDB deployment

MongoDB deployment is performed by using a playbook (5.5.1, “Controller” on page 196) that invokes another playbook for each member of the replica set. In this section, we describe each of those playbooks.

5.5.1 Controller

The controller playbook (shown in Example 5-15) is a sort of master playbook that is used to invoke the Tasks playbook (5.5.2, “Tasks” on page 197 playbook).

Example 5-15 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 5-15 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.

5.5.2 Tasks

Tasks let you configure and save frequently run jobs so you can later run them with one click.

Example 5-16 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]   firewalld:
[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 5-16 on page 197 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.

5.5.3 Quiescing

Use the **`mongo`** command to lock the database, as shown in Example 5-17.

Example 5-17 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 5-17 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.

5.5.4 Resuming

Use the **mongo** command to unlock the database, as shown in Example 5-18.

Example 5-18 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 5-18 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.

5.6 Terminating

The termination of a replica set is performed by using a controller playbook (5.6.1, “Controller” on page 204) and an embedded task.

5.6.1 Controller

The controller playbook that is shown in Example 5-19 includes the `terminate-host.yml` task that shuts down replica set VMs.

Example 5-19 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 5-19 on page 204 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.

5.6.2 Embedded task

The playbook that is shown in Example 5-20 shuts down a VM if it is a shadow MongoDB server or if we force the shutdown of a *real* MongoDB server.

Example 5-20 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 5-20 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.

5.7 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.

5.7.1 Master playbook

The `destroy-hosts.yml` playbook, which is shown in Example 5-21, controls the decommissioning process.

Example 5-21 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 5-21 on page 205 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.

5.7.2 Image and volume tasks

The `deploy-image.yml` file that is shown in Example 5-22 contains the tasks that are required to provision a single VM and data volume.

Example 5-22 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 5-22 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.

5.7.3 Deploying or destroying the variables file

The file that is shown in Example 5-23 provides the variables that are required by the `deploy-hosts.yml` playbook and `deploy-image.yml` tasks.

Example 5-23 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 5-23 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 `volumegroup`.
- [013]: Name of the volume group.
- [014]: Port to SSH for SAN Volume Controller.

A

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 A-1.

Example: A-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 A-2.

Example: A-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 A-3. This error occurs because omitting the **FROM** clause is not allowed in FUJITSU Enterprise Postgres.

Example: A-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 A-4. In SQL2, the second **INSERT** statement inserts a zero-length string ('') into column COL_2 of table TBL.

Example: A-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 A-5.

Example: A-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 A-6 because FUJITSU Enterprise Postgres treats that zero-length string as a different value from NULL.

Example: A-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 A-1and Table A-2.

Table A-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 A-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 A-7 and Example A-8 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example A-9 and Example A-10.

Example: A-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: A-8 [Oracle-Result] MINUS operator

```
COL_1
----------
AAA
CCC
```

Example: A-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: A-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 A-11 and Example A-12 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example A-13 and Example A-14 on page 216.

Example: A-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: A-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: A-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: A-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 A-15 and Example A-16 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example A-17 and Example A-18.

Example: A-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: A-16 [Oracle-Result] Correlation name of subquery

```
COL_1 COL_2
----- -----
xxxxx XXXXX
yyyyy YYYYY
```

Example: A-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: A-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 A-19 and Example A-20 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example A-21 and Example A-22.

Example: A-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: A-20 [Oracle-Result] DELETE statements

```
COL_1
-----
yyyyy
```

Example: A-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: A-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 A-23 with the result in Example A-24) with the **TRUNCATE** statement in FUJITSU Enterprise Postgres (Example A-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 A-23 with the result in Example A-24) with the **DELETE** statement in FUJITSU Enterprise Postgres (Example A-26 on page 219 with the result in Example A-27 on page 219). 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: A-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: A-24 [Oracle-Result] TRUNCATE statements for a partition

```
CO COL_2
-- -----
A2 22222
B1 33333
B2 44444
```

Example: A-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: A-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: A-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 A-28 with its result in Example A-29 on page 220) is not supported on FUJITSU Enterprise Postgres. If `ROWNUM` is specified in the **`SELECT`** statement list, replace it with the `ROW_NUMBER` function (Example A-30 on page 220 with its result in Example A-31 on page 220) when migrating to FUJITSU Enterprise Postgres.

Example: A-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: A-29 [Oracle-Result] ROWNUM specified in the SELECT list

```
    ROWNUM COL
---------- ---
         1 DDD
         2 CCC
         3 BBB
         4 AAA
```

Example: A-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: A-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 A-32 and Example A-33 for Oracle Database to the ones for FUJITSU Enterprise Postgres in Example A-34 on page 221 and Example A-35 on page 221.

Example: A-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: A-33 [Oracle-Result] ROWNUM specified in the WHERE clause

```
COL
---
DDD
CCC
```

Example: A-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: A-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 A-36 shows a sequence definition when using an Oracle Database with the results shown in Example A-37. Example A-38 shows a sequence definition when using FUJITSU Enterprise Postgres with the results shown in Example A-39.

Example: A-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: A-37 [Oracle-Result] Sequence definition

```
Sequence created.
Sequence created.
```

Example: A-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: A-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 A-40 on page 223 and Example A-41 on page 223 show the use of sequence pseudocolumns in the Oracle Database, and Example A-42 on page 223 and Example A-43 on page 223 show the use of sequence pseudocolumns in FUJITSU Enterprise Postgres.

Example: A-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: A-41 [Oracle-Result] Sequence pseudocolumns

```
     COL_1 COL_2
---------- -----
         1 AAAAA
         1 aaaaa
         2 BBBBB
         2 bbbbb
```

Example: A-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: A-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 A-44 and Example A-45) with their use in FUJITSU Enterprise Postgres (Example A-46 and Example A-47).

Example: A-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', 'bbbbb');
INSERT INTO TBL VALUES('CCCCC', 'ccccc');
SELECT * FROM TBL WHERE COL_2 ^= 'bbbbb'
ORDER BY COL_1;
```

Example: A-45 [Oracle-Result] Inequality operator

```
COL_1 COL_2
----- -----
AAAAA aaaaa
CCCCC ccccc
```

Example: A-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', 'bbbbb');
INSERT INTO TBL VALUES('CCCCC', 'ccccc');
SELECT * FROM TBL WHERE COL_2 != 'bbbbb'
ORDER BY COL_1;
```

Example: A-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 A-48 and Example A-49 on page 225 to the use of **REGEXP_LIKE** in FUJITSU Enterprise Postgres in Example A-50 on page 225 and Example A-51 on page 225.

Example: A-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: A-49 [Oracle-Result] REGEXP_LIKE

```
     COL_1 COL_2
---------- ----------
         2 Abcde
         3 abcde
```

Example: A-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: A-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 A-52 and Example A-53 on page 226) with FUJITSU Enterprise Postgres (Example A-54 on page 226 and Example A-55 on page 226).

Example: A-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: A-53 [Oracle-Result] SYSDATE

```
COL_1      COL_2
---------- -----
2021-10-12 XXXXX
```

Example: A-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: A-55 [FUJITSU Enterprise Postgres-Result] SYSDATE

```
   col_1    | col_2
------------+-------
 2021-10-12 | XXXXX
```

Migration pattern: SYS_CONNECT_BY_PATH

SYS_CONNCT_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 A-56 and Example A-57 on page 227 demonstrate how Oracle Database handles the **SYS_CONNCT_BY_PATH** function. Example A-58 on page 227 and Example A-59 on page 227 demonstrate how FUJITSU Enterprise Postgres handles the **SYS_CONNCT_BY_PATH** function.

Example: A-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: A-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: A-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: A-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 A-60 and Example A-61 provide examples when using Oracle Database, and Example A-62 on page 229 and Example A-63 on page 229 provide examples when using FUJITSU Enterprise Postgres.

Example: A-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: A-61 [Oracle-Result] Database object name

```
     COL_1 COL_2
---------- -----
         1 AAAAA
         2 BBBBB
```

Example: A-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: A-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 A-64 and Example A-65 provide examples when using an Oracle Database, and Example A-66 and Example A-67 on page 230 provide examples when using FUJITSU Enterprise Postgres.

Example: A-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: A-65 [Oracle-Result] Implicit conversion

```
COL_1
-----
AAAAA
BBBBB
```

Example: A-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: A-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 A-68 and Example A-69 provide examples when using an Oracle Database, and Example A-70 and Example A-71 provide examples when using FUJITSU Enterprise Postgres.

Example: A-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: A-69 [Oracle-Result] Zero-length string

```
COL_1 COL_2
----- -----
yyyyy
zzzzz
```

Example: A-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: A-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 A-72 and Example A-73 provide examples when using an Oracle Database, and Example A-74 on page 232 and Example A-75 on page 232 provide examples when using FUJITSU Enterprise Postgres.

Example: A-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: A-73 [Oracle-Result] Comparing a fixed-length character string and variable-length character string

```
COL_1
-----
AAA
CCC
```

Example: A-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: A-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 A-76 and Example A-77 on page 234 provide examples when using Oracle Database, and Example A-78 on page 234 and Example A-79 on page 235 provide examples when using FUJITSU Enterprise Postgres.

Example: A-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: A-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: A-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: A-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 A-80 and Example A-81 on page 237 provide examples when using Oracle Database, and Example A-82 on page 237 and Example A-83 on page 238 provide examples when using FUJITSU Enterprise Postgres.

Example: A-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: A-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: A-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: A-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 A-84 and Example A-85 provide examples when using Oracle Database, and Example A-86 and Example A-87 on page 240 provide examples when using FUJITSU Enterprise Postgres.

Example: A-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: A-85 [Oracle-Result] Cursor variables

```
id=1, val=data1
id=2, val=data2
PL/SQL procedure successfully completed.
```

Example: A-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: A-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 A-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 A-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 A-3 on page 240 when migrating to FUJITSU Enterprise Postgres. These examples enable Oracle compatibility features.

- CURSOR_ALREADY_OPEN: Shown in Example A-88 through Example A-91 on page 243.
- DUP_VAL_ON_INDEX: Shown in Example A-92 on page 243 through Example A-95 on page 244.
- INVALID_CURSOR: Shown in Example A-96 on page 244 through Example A-99 on page 245.
- INVALID_NUMBER: Shown in Example A-100 on page 245 through Example A-103 on page 246.
- ZERO_DIVIDE: Shown in Example A-104 on page 246 through Example A-107 on page 247.

CURSOR_ALREADY_OPEN

Example: A-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: A-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: A-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: A-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: A-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: A-93 [Oracle-Result] Predefined exceptions: DUP_VAL_ON_INDEX

```
INSERT: data1
DUP_VAL_ON_INDEX
PL/SQL procedure successfully completed.
```

Example: A-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: A-95 [FUJITSU Enterprise Postgres- Result] Predefined exceptions: DUP_VAL_ON_INDEX

```
INSERT: data1
DUP_VAL_ON_INDEX
DO
```

INVALID_CURSOR

Example: A-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: A-97 [Oracle-Result] Predefined exceptions: INVALID_CURSOR

```
Anonymous block: BEGIN
INVALID_CURSOR
PL/SQL procedure successfully completed.
```

Example: A-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: A-99 [FUJITSU Enterprise Postgres- Result] Predefined exceptions: INVALID_CURSOR

```
Anonymous block: BEGIN
INVALID_CURSOR
DO
```

INVALID_NUMBER

Example: A-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: A-101 [Oracle-Result] Predefined exceptions: INVALID_NUMBER

```
INVALID_NUMBER
PL/SQL procedure successfully completed.
```

Example: A-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: A-103 [FUJITSU Enterprise Postgres- Result] Predefined exceptions: INVALID_NUMBER

```
INVALID_NUMBER
DO
```

ZERO_DIVIDE

Example: A-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: A-105 [Oracle-Result] Predefined exceptions: ZERO_DIVIDE

```
ZERO_DIVIDE
PL/SQL procedure successfully completed.
```

Example: A-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: A-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 A-108 and Example A-109 from Oracle Database with the handling of *SQLSTATE* in Example A-110 and Example A-111 on page 249 with FUJITSU Enterprise Postgres.

Example: A-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: A-109 [Oracle-Result] SQLCODE

```
Error: code=-1, message=ORA-00001: unique constraint (TESTUSER01.SYS_C007646)
violated
PL/SQL procedure successfully completed.
```

Example: A-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;
$$ LANGUAGE plpgsql;
```

Example: A-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: A-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: A-113 [Oracle-Result] Stored functions

```
Tom execute create_message
NOTICE: sample message
PL/SQL procedure successfully completed.
```

Example: A-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: A-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 A-116, with the results shown in Example A-117 on page 252.
- FUJITSU Enterprise Postgres stored functions that are converted from the Oracle Database stored functions, as shown in Example A-118 on page 252, with the results shown in Example A-119 on page 252 without specifying IMMUTABLE.
- FUJITSU Enterprise Postgres stored functions that specify IMMUTABLE for Example A-118 on page 252 are shown in Example A-120 on page 252, with the results in Example A-121 on page 253.

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: A-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: A-117 [Oracle-Result] Stored functions

```
MESSAGE
--------------------------------------------------------------------------------
NOTICE: sample message
```

- Example of FUJITSU Enterprise Postgres simply converted

Example: A-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: A-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: A-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: A-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 A-122 - Example A-125 on page 255 show the basic conversion method of stored procedures. These examples of FUJITSU Enterprise Postgres enable Oracle compatibility features.

Example: A-122 [Oracle -PL/SQL] Stored procedures

```
ALTER SESSION SET NLS_TIMESTAMP_FORMAT = 'YYYY-MM-DD HH24:MI:SS.FF';
CREATE TABLE user_tbl(
  user_id NUMBER UNIQUE,
  name VARCHAR2(10)
);
INSERT INTO user_tbl 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: A-123 [Oracle-Result] Stored procedures

```
---- Login message ----
Hello Tom
Login Time: 2021-10-21 04:17:05.207654
PL/SQL procedure successfully completed.
```

Example: A-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: A-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 A-126 - Example A-129 on page 256 of FUJITSU Enterprise Postgres enable Oracle compatibility features.

Example: A-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: A-127 [Oracle-Result] Cursor FOR LOOP statements

```
id=1, val=data1
id=2, val=data2
PL/SQL procedure successfully completed.
```

Example: A-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: A-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 A-130 - Example A-133 on page 258).
- 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: A-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: A-131 [Oracle-Result] EXECUTE IMMEDIATE statement

```
   USER_ID NAME
---------- ----------
         1 Tom
         2 Mary
```

Example: A-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: A-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 A-134 - Example A-137 on page 259 of FUJITSU Enterprise Postgres enables Oracle compatibility features.

Example: A-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: A-135 [Oracle-Result] Exponentiation operator

```
2 ** 10 = 1024
PL/SQL procedure successfully completed.
```

Example: A-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: A-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 A-138 and Example A-139. Examples of a FUJITSU Enterprise Postgres **FOR LOOP** statement are in Example A-140 and Example A-141 on page 260.

Example: A-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: A-139 [Oracle-Result] FORALL statement

```
PL/SQL procedure successfully completed.
NAME
----------
Tom
Mary
John
Jane
Alex
```

Example: A-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: A-141 [FUJITSU Enterprise Postgres- Result] FOR LOOP statement

```
DO
 name
------
 Tom
 Mary
 John
 Jane
 Alex
```

B

Additional data source information

UK Ordnance Survey maps provide a rich mapping source. Limited 1:125,000 range maps are available at no charge at OS Data Hub. An alternative source for worldwide mapping is OpenStreetMap, which publishes geographically based maps at OpenStreetMap.

Climate models are constantly being updated because different modeling groups around the world incorporate higher spatial resolution, new physical processes, and biogeochemical cycles. These modeling groups coordinate their updates around the schedule of the Intergovernmental Panel on Climate Change (IPCC) assessment reports and release a set of model results (“runs”) in the lead-up to each report.

Note: CLIM26 provides future predictions of weather and climatic changes for the next 80 years, which are broken down into Global Climate Models (GCMs) and Shared Socioeconomic Pathways (SSPs) (four territories at the time of writing). This data is published and made publicly available as “CMIP6 multi-model ensemble” at World Climate Research Program.

Abbreviations and acronyms

App J	Appendix J
BA	Business Analytics
BI	Business Intelligence
CA	certificate authority
CDC	Change Data Capture
Ci/CD	Continuous Integration and Continuous Development
IBM CIC	IBM Cloud Infrastructure Center
CPACF	CP Assist for Cryptographic Functions
DBMS	database management system
DDL	Data Definition Language
DML	Data Manipulation Language
DSS	Direct Systems Support
DX	digital transformation
EAL5+	Evaluation Assurance Level 5+
ETL	extract, transform, and load
FDW	Foreign Data Wrapper
FFIEC	Federal Financial Institutions Examination Council
FQDN	fully qualified domain name
FTDC	Full Time Diagnostic Data Capture
GAP	Grafana, Alertmanager, and Prometheus
GCM	Global Climate Model
GCS	Geographic Coordinate System
GIS	geospatial information systems
HA	high availability or highly available
HADR	high availability and disaster recovery
HSM	hardware security module
IaaS	infrastructure as a service
IBM	International Business Machines Corporation
IBM SGC	IBM Safeguarded Copy
IFL	IBM Integrated Facility for Linux
IPCC	Intergovernmental Panel on Climate Change
LPAR	logical partition
LVM	Logical Volume Management
MTLS	Mutual Transport Layer Security
ORDBMS	object-relational database management system
OSC	Open Spatial Consortium
OS	operating system
PCI-DSS	Payment Card Industry Data Security Standard
PITR	point-in-time recovery
POC	proof-of-concept
QGIS	Quantum Geographic Information System
RAC	Real Application Cluster
RBAC	role-based access control
RHOCP	Red Hat OpenShift Container Platform
ROI	return on investment
RPO	recovery point objective
RTO	recovery time objective
SME	subject matter expert
SNA	Sine Nomine Associates
SRID	spatial reference identifier
SRS	spatial reference system
SSP	Shared Socioeconomic Pathway
TCO	total cost of ownership
TDE	Transparent Data Encryption
TOID	Topographic Object Identifier
TPC-C	TPC Benchmark C
UPRN	Unique Property Reference Number
USRN	Unique Street Reference Number
VM	virtual machine
VPN	virtual private network
WSC	Washington System Center

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 publication 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

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

This website is also relevant as a further information source:

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

Redbooks

Leveraging LinuxONE to Maximize Your Data Serving Capabilities

SG24-8518-00

ISBN 0738460486

IBM

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8518-00

ISBN 0738460486

Printed in U.S.A.