Redbooks
ibm.com/redbooks

Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE

Sam Amsavelu

Neeraj Arora

Nikhil Kumar Bayawat

Victoria Coates

Gary Evans

Yuki Ishimori

Pankaj Kapoor

Fumiaki Nakamura

Alex Osadchyy

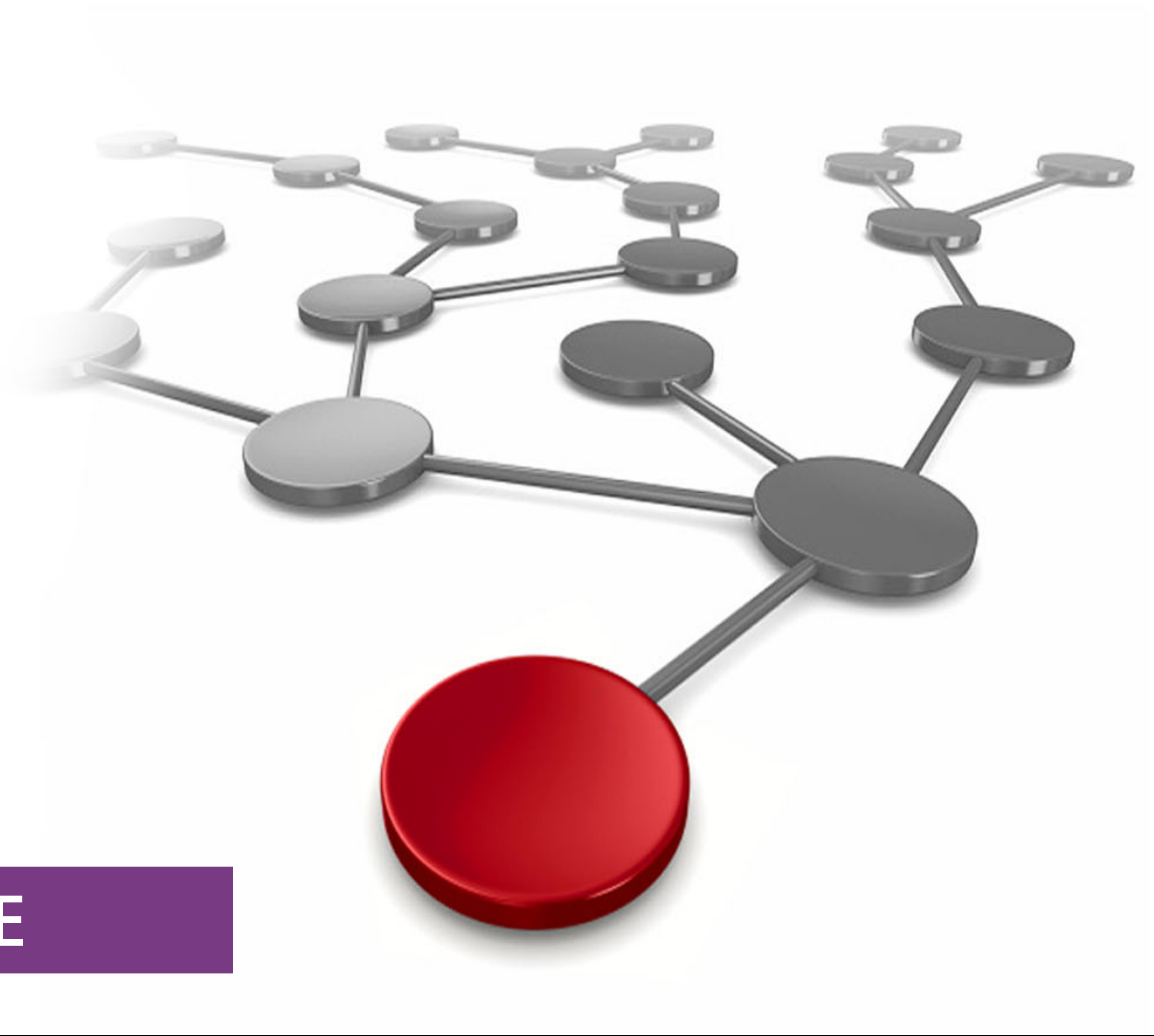
Varun Narula

Zeus Ng

Vaishnavi Prabakaran

Anand Subramanian

Jin Yang



LinuxONE

IBM

Redbooks

IBM Redbooks

Data Serving with FUJITSU Enterprise Postgres on IBM LinuxONE

July 2021

SG24-8499-00

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

First Edition (July 2021)

This edition applies to FUJITSU Enterprise Postgres 12 Advanced Edition on IBM LinuxONE.

Contents

Notices

Trademarks

Preface

Enterprises require support and agility to work with big data repositories and relational databases. FUJITSU Enterprise Postgres is one of the leading relational database management systems (RDBMSs), and it is designed to work with large data sets.

As more companies transform their infrastructures with hybrid cloud services, they require environments that protect the safety of their data and business rules.

At IBM®, we believe that your data is yours and yours alone.[1] The insights and advantages that come from your data are yours to use in the pursuit of your business objectives. IBM is dedicated to this mission, and the IBM LinuxONE platform is designed around this core statement.

IBM LinuxONE is a secure and scalable data serving and computing platform that is made for today's critical workloads. IBM LinuxONE is an all-Linux enterprise platform for open innovation that combines the best of Linux and open technology with the best of enterprise computing in one system.

Combining FUJITSU Enterprise Postgres, which is a robust Relational Database Management System (RDBMS) that provides strong query performance and high availability (HA), with IBM LinuxONE can transform your application and data portfolio by providing innovative data privacy, security, and cyber resiliency capabilities, which are all delivered with minimal downtime.

This IBM Redbooks® publication describes data serving with FUJITSU Enterprise Postgres 12 that is deployed on IBM LinuxONE, which provides the scalability, business-critical availability, and security that your enterprise requires. This publication is useful to IT architects, system administrators, and others who are interested in understanding the significance of using FUJITSU Enterprise Postgres on IBM LinuxONE.

This publication is written for those who are familiar with IBM LinuxONE and have some experience in the use of PostgreSQL.

In this publication, you see references to the FUJITSU Enterprise Postgres 12 on IBM LinuxONE manuals. These manuals are included with the `.iso` file for FUJITSU Enterprise Postgres. You can also find these manuals at *FUJITSU Enterprise Postgres 12 on IBM LinuxONE*.

You will find all the manuals that are mentioned in this publication under the words *Manual Set* and listed by the keywords that are used in this publication.

[1] `https://www.ibm.com/watson/data-privacy/`

Authors

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

Sam Amsevelu has worked for IBM for the past 25 years advising customers about how to build HA, reliable, and secure infrastructure architectures for their databases. They are 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 TCO possible. They are a 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), and high availability and disaster recovery (HADR) solutions (Oracle RAC, Data Guard, GoldenGate, replication, and sharding).

Neeraj Arora is a Consulting Architect with the Fujitsu Global Consulting Services team based out of Australia. He has more than 28 years of experience and holds a Bachelor of Engineering degree in Electronic Engineering from Birla Institute of Technology and Sciences (Pilani), India. His areas of expertise are in cloud technologies and cloud-native architectures. He has contributed to the container database and its function in creating container platforms for stateful sets.

Nikhil Kumar Bayawat is a Senior Consultant working with Fujitsu Australia Software Technology (FAST) in Sydney. He holds a bachelor's degree in Electronics and Communication engineering from Netaji Subhas University of Technology (formerly Netaji Subhas Institute of Technology), New Delhi, India. Nikhil has 18+ years of experience working across different IT domains covering Project Management, Consulting, Organizational Change Management, IT Operations, and IT Services. He has extensively worked on IBM Mainframes and IBM LinuxONE, and he has written on database performance, modernizing application development on Mainframes, and HA architectures that use FUJITSU Enterprise PostgresSQL on IBM LinuxONE. He worked with the Royal Bank of Scotland, IBM, Cognizant Technology Solutions, and Tata Consultancy Services before joining Fujitsu Australia. Over the years, Nikhil has acquired professional credentials and holds a PMI PMP, Prince2 Practitioner, and Agile Scrum Master in project management. He also holds AWS Solution Architect Associate, AWS Big Data Specialty, and Google Cloud Professional Data Engineer technical credentials, in addition to being certified for Db2 for z/OS® Database Administrator. He works with international, culturally diverse teams and excels at problem solving and managing critical situations. Nikhil presents regularly at customer workshops to provide consultancy in solving data management challenges.

Victoria Coates is a Z Client Technical Specialist in the IBM Americas distribution market. She has been with IBM since 2018, and holds a Bachelor's of Science degree from Northwestern University. Since joining IBM, her focus includes MongoDB on IBM LinuxONE, IBM Blockchain Platform, and Hyper Protect services.

Gary Evans is a Customer Success and Technical Services Manager at Fujitsu Limited, Software Business Unit. Gary has over 28 years of experience in software development and data technologies, and has worked in 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.

Yuki Ishimori is a Database Software Engineer for FUJITSU Enterprise PostgreSQL in Japan. He has 10 years of experience developing various database features and designing and deploying databases in the field. His area of expertise includes database operational design and performance tuning. He has written extensively on HA, Pgpool-II, and backup and recovery.

Pankaj Kapoor is a Senior Project Manager working for Fujitsu. He holds a master's degree in computer application from the Birla Institute of Technology-India, and he is certified as a PMP and scrum master. In nearly 17 years of his experience, he has worked in various domains, including telecommunications, web applications, security, databases, and the cloud technical stack. Pankaj has managed various releases of FUJITSU Enterprise Postgres on x86, s390x, and the Red Hat OpenShift environment. He has lead multiple IBM feature integrations with FUJITSU Enterprise Postgres, including CrytoExpress adapter integration with FUJITSU Enterprise Postgres.

Fumiaki Nakamura works in pre-sales for FUJITSU Enterprise Postgres in Japan. He has 5 years of experience in the PostgreSQL field. His areas of expertise include data analytics.

Alex Osadchyy is a Senior Technical Solution Architect at IBM Systems, US. Alex works with key IBM Z® independent software vendors (ISVs) to guide their journey to and growth of hybrid cloud and data solutions that are based on the enterprise platform for mission-critical applications, that is, IBM Z and IBM LinuxONE. He is a solution architect, engineering manager, and software developer with over 17 years of experience in creating software and solutions from low-level network drivers to multi-cloud platforms. With business insights, doctoral backgrounds, and research and development experience, he leverages a leadership and scientific approach for strategic solutions.

Varun Narula is a Customer Consultant in Australia. He has 12 years of experience in designing and implementing database solutions. He holds a bachelor's degree in Computer Science and Engineering. His areas of expertise include database application design, performance tuning, and database monitoring. He has written extensively on FUJITSU Enterprise Postgres application development, backup and recovery, and FUJITSU Enterprise Postgres Database monitoring.

Zeus Ng is a Principal Solution Architect working with FAST. He has 25+ years of experience in information technology. He holds a degree in Information Technology from City University of Hong Kong S.A.R. of the PRC. His areas of expertise include Cloud Architecture and Migration, Digital Transformation, and IT Security Governance. He has extensively written about FUJITSU Enterprise Postgres on Red Hat OpenShift.

Vaishnavi Prabakaran is a Senior Technical Specialist working on the FUJITSU Enterprise PostgreSQL database product. She has over 13 years of IT experience, including design and development of various database features, applications, and tools. Her areas of expertise include database security, performance, and administration. She holds a master's degree in Computer Applications from SRM University.

Anand Subramanian is the Technical Leader for IBM LinuxONE for Confidential Computing and Secure Data Serving in the Australia and New Zealand region. He is a Technology Thought Leader with 24 years of experience in the information technology field who has worked 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 from the University of Western Sydney and a Certificate in Machine Learning from Caltech. He is a Certified IT Architect. He has been employed by IBM for over 10 years.

Jin Yang is a Consulting System Service Representative in China. He joined IBM in 1999 to support IBM Z products maintenance for clients in China. He works in the Technical Support Group to provide second-level support for IBM Z and IBM LinuxONE clients as a country Top Gun since 2009. His areas of expertise include IBM Z and IBM LinuxONE hardware, channel connectivity, z/VM®, and Linux on IBM Z.

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

Bill Lamastro and Tom Ambrosio
IBM CPO

Rajni Baliyan, Marcelo Hauschild, Aya Hoshino, Junji Kawai, Niki Kennedy, Kazuhiko Inoue, Masaki Nagao, Kazuhiro Taniguchi, Nobuaki Takahashi, Yasuko Sato, Keiko Teranishi, Keiko Teranishi, Daisuke Ninomiya, Ryohei Takahashi, Prashant Verma
Fujitsu

Vance Morris, Koos Du Pleis, Abilio Oliveira, Nilton dos Santos
IBM

Now you can become a published author, too!

Comments welcome

- 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

- Look for us on LinkedIn:

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

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

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

- Stay current on recent Redbooks publications with RSS Feeds:

  `http://www.redbooks.ibm.com/rss.html`

1

Customer value

FUJITSU Enterprise Postgres is an exceptionally reliable and robust relational database that was created for organizations that require security, high availability (HA), scalability, and strong performance. It has the flexibility to host databases on bare metal servers and in the cloud. It is based on the world-renowned open source object relational database management system (RDBMS) that is called PostgreSQL, with more enterprise-grade features for better performance and enhanced security integration with IBM LinuxONE, which makes it a secure enterprise Postgres on IBM LinuxONE.

IBM LinuxONE is an enterprise-class, purpose-built data serving platform for mission-critical Linux business workloads that mandate the highest levels of security, scalability, performance, and resilience. The platform supports traditional n-tier Linux applications and databases, and cloud-ready containerized workloads and microservices, which deliver a secure hybrid cloud platform for all organizations.

Designed to consolidate thousands of cores of traditional n-tier and cloud-native Linux workloads into standard 19” rack sized enterprise servers, IBM LinuxONE delivers unparalleled savings in operating costs, energy, data center floor space, networking, and administration.

FUJITSU Enterprise Postgres uses PostgreSQL, which is a feature-rich open source RDBMS that is used by millions of users globally. It enables integration with a wide range of software bundles, information utilization systems, development tools, and application runtime environments. By further enhancing key features, Fujitsu delivers a flexible solution that achieves high reliability and asset protection that is aligned with an enterprise data management strategy.

FUJITSU Enterprise Postgres offers easy setup and smart recovery features that ensure seamless integration into any existing business system to provide hassle-free installation with an easy and intuitive operation for the user. System managers no longer must meet the challenge of allocating specific database expertise.

Improved high reliability and high performance along with Fujitsu long-term support creates substantial benefits to further complement intelligent business data systems for enterprises.

This chapter describes this solution in detail and the customer value of FUJITSU Enterprise Postgres features on IBM LinuxONE.

This chapter covers the following topics:

- Open standards that are enhanced with Fujitsu and IBM innovation
- Compatibility with open source PostgreSQL
- Cloud-native containerized FUJITSU Enterprise Postgres
- Data security
- Data integrity
- IBM LinuxONE performance and scalability
- Business continuity
- Disaster preparedness
- Hybrid Transactional Analytical Processing: Merging Online Transaction Processing and Online Analytical Processing
- Faster data consumption and memory optimization
- FUJITSU Enterprise Postgres editions for IBM LinuxONE
- Licensing

1.1 Open standards that are enhanced with Fujitsu and IBM innovation

Enterprises are competing on speed to market by leveraging open source products that offer both agility and flexibility to offer many ways to meet technology requirements and solve business problems. When data security is paramount for enterprises, commercial open source products provide cost-effective superior data security solutions while competing at par with proprietary software counterparts. PostgreSQL is one such database product that has gained enormous traction among community developers and enterprises over the last three decades.

1.2 Compatibility with open source PostgreSQL

PostgreSQL is the most sophisticated open source object RDBMS. It is respected for its community's dedication in developing the database software for stability, scalability, security, and extensibility, which are some of the reasons that Fujitsu chose PostgreSQL when it decided to build its most effective database solution.

Fujitsu has been committed to the PostgreSQL culture since 2004, and Fujitsu frequently contributes ideas to PostgreSQL growth that is based on Fujitsu comprehensive business experience. In turn, the improvement of PostgreSQL explicitly influences the value of FUJITSU Enterprise Postgres. CSV logging, table space, save points, 64-bit Windows support, COPY view FROM, viewing `pg_hba_file`, partitioning performance improvement, and improvement in table access methods are a few of the notable contributions among numerous others that Fujitsu has added since open source PostgreSQL V8.

Figure 1-1 on page 3 shows the latest enterprise features that Fujitsu developed and integrated with the PostgreSQL. Fujitsu has also packaged various extensions with FUJITSU Enterprise Postgres while also providing the services and support to create a complete database management solution.

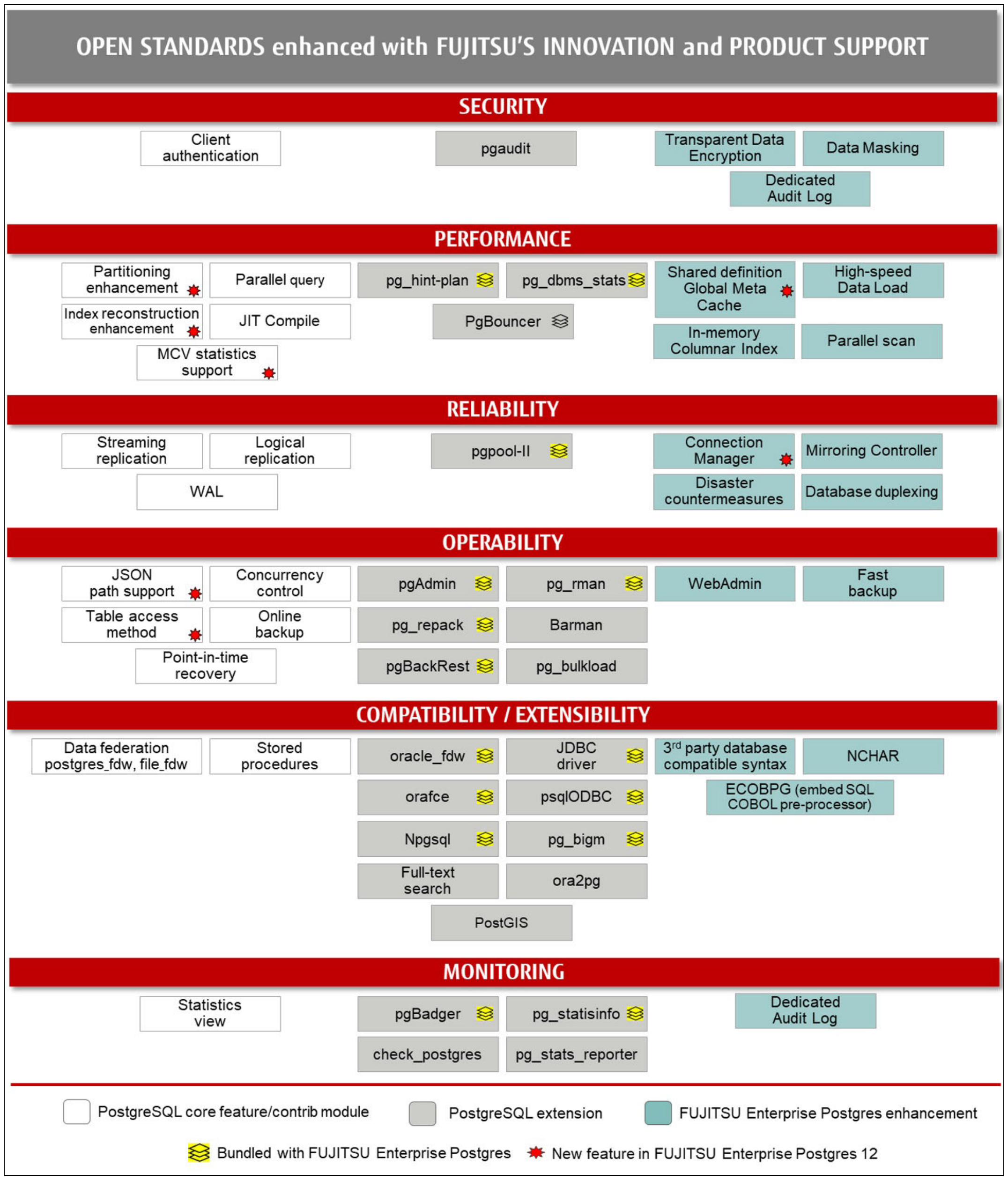


Figure 1-1 Open standards that are supported by FUJITSU

1.2.1 The most secure enterprise Postgres on IBM LinuxONE

IBM LinuxONE makes it easy for organizations to safeguard their most valuable data assets against internal and external threats by limiting potential attack vectors, simplifies the task of ensuring compliance with the industry standards and regulations, and improves the ability of the organization to gain and retain customer trust.

Security is designed into IBM LinuxONE at both the hardware and software levels. As an example, the platform's Pervasive Encryption capability can encrypt all data that is associated with an application, database, or cloud service, whether at rest or in flight.

This level of protection is achieved through the integrated hardware accelerated encryption of data that is delivered with near-zero overhead by the on-chip Central Processor Assist for Cryptographic Function (CPACF) and the Crypto Express adapter, which are the only commercially available PCIe hardware security module (HSM) and cryptographic coprocessor with the highest security rating (FIPS 140-2 Level 4).

Combined, the hardware accelerated encryption and key management capabilities of the IBM LinuxONE CPACF and HSM make it easier for applications to meet regulations such as HIPAA and PCI DSS by delivering the following items:

- One hundred percent data protection
- One hundred percent key protection with the highest level of crypto certification
- One hundred percent audit ready with data protection compliance
- One hundred percent isolation between logical partitions (LPARs) for multitenancy with privacy, which is rated at Common Criteria Evaluation Assurance Level (EAL) 5+

The culmination of open-source software (OSS) PostgreSQL features, FUJITSU Enterprise Postgres enterprise features and its integration with the advanced IBM LinuxONE security capabilities makes FUJITSU Enterprise Postgres the world's most secure Postgres on IBM LinuxONE. Figure 1-2 shows that flexible computing and highly performing I/O that is secured by HSM-based encryption makes IBM LinuxONE and FUJITSU Enterprise Postgres one of the most cyber resilient, scalable, extensible, and performant database-platform combinations that are backed by 100% compatibility with open source PostgreSQL.

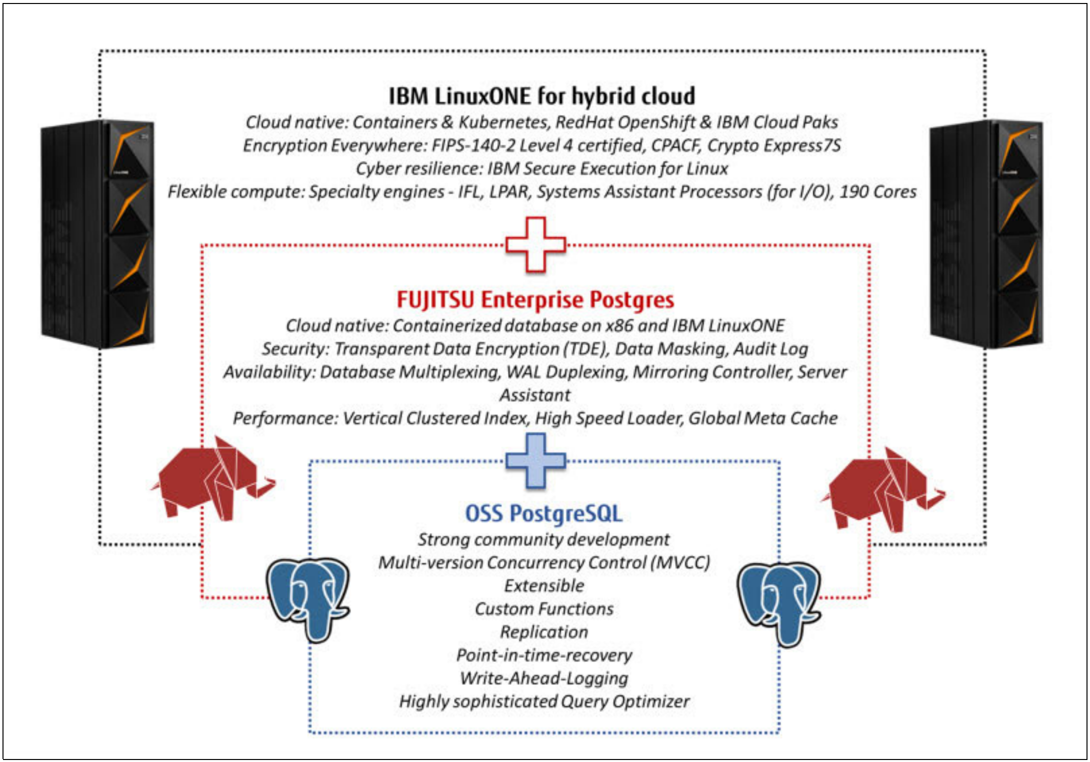

IBM LinuxONE for hybrid cloud
Cloud native: Containers & Kubernetes, RedHat OpenShift & IBM Cloud Paks
Encryption Everywhere: FIPS-140-2 Level 4 certified, CPACF, Crypto Express7S
Cyber resilience: IBM Secure Execution for Linux
Flexible compute: Specialty engines - IFL, LPAR, Systems Assistant Processors (for I/O), 190 Cores
FUJITSU Enterprise Postgres
Cloud native: Containerized database on x86 and IBM LinuxONE
Security: Transparent Data Encryption (TDE), Data Masking, Audit Log
Availability: Database Multiplexing, WAL Duplexing, Mirroring Controller, Server Assistant
Performance: Vertical Clustered Index, High Speed Loader, Global Meta Cache
OSS PostgreSQL
Strong community development
Multi-version Concurrency Control (MVCC)
Extensible
Custom Functions
Replication
Point-in-time-recovery
Write-Ahead-Logging
Highly sophisticated Query Optimizer

Figure 1-2 FUJITSU Enterprise Postgres enterprise features on IBM LinuxONE

1.2.2 Fujitsu support for the PostgreSQL community

Fujitsu is a major sponsor of the PostgreSQL community. The title of *Major Sponsor* was offered by the Sponsor Committee for the significant and lasting contributions that Fujitsu has made to PostgreSQL over the years. The improvements that Fujitsu has been contributing to the open source PostgreSQL inherently improves the customer value of FUJITSU Enterprise Postgres with all the enterprise-grade features that are developed by Fujitsu for enterprise customers. Figure 1-3 shows the Fujitsu enterprise features over and above the community developed PostgreSQL features.

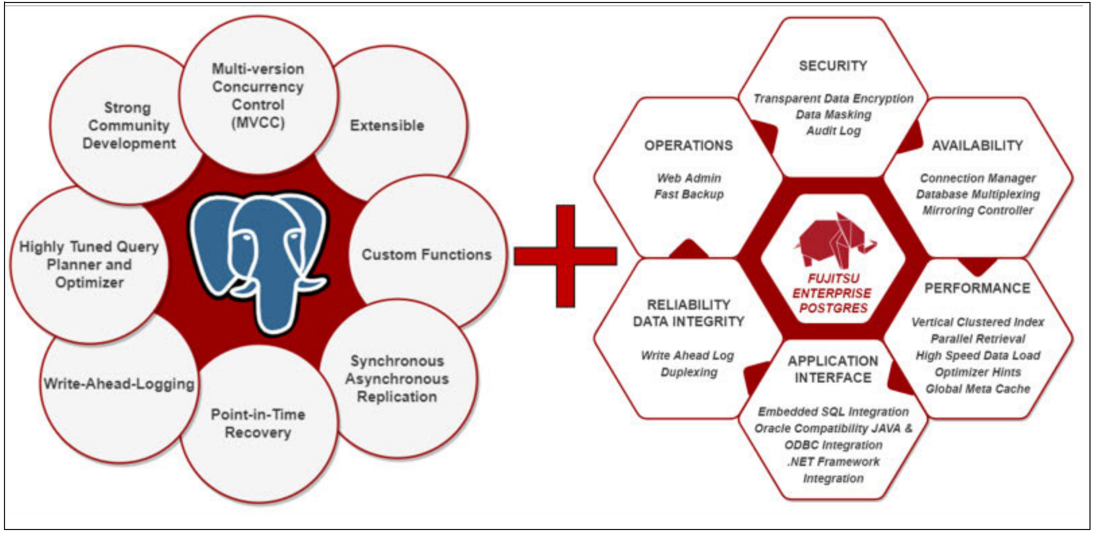


Figure 1-3 PostgreSQL enhanced with FUJITSU Enterprise Postgres features

Fujitsu also supports more than 10 open source PostgreSQL peripheral devices that cover various database management tasks, such as administration, migration, and operations. This support provides customers with a single shop destination for the RDBMS and the associated peripheral software bundled together. Whether it is about API connectivity, SQL extensions like ORAFCE, or HA through Pgpool-II, various open source peripheral devices make a complete database management tools ecosystem around FUJITSU Enterprise Postgres. The biggest value that customers receive from the software that is bundled open source peripherals is that these tools are validated, tested, and supported by Fujitsu, which reduces any validation effort by the customers when building their highly secure database management solution by using FUJITSU Enterprise Postgres.

Table 1-1 shows the open source tools that are packaged with FUJITSU Enterprise Postgres and supported by Fujitsu.

Table 1-1 Open-source tools packaged with FUJITSU Enterprise Postgres and supported by Fujitsu

Application development and database management use cases	**OSS peripheral tools**
Oracle compatibility in FUJITSU Enterprise Postgres	`orafce`
Failover, connection pooling, and load balancing	`Pgpool-II`
Easy and efficient access to Oracle databases by using a foreign data wrapper	`oracle_fdw`
Collection and accumulation of statistics for performance monitoring and health checks	`pg_statsinfo`

Application development and database management use cases	**OSS peripheral tools**
Stable SQL performance for consistent application responses	`pg_hint_plan and pg_dbms_stats`
Space management through table and index reorganization	`pg_repack`
Backup and recovery management	`pg_rman`
Log analysis for monitoring	`pgBadger`
Full-text search (multibyte)	`pg_bigm`
JDBC driver	PostgreSQL JDBC driver
Open Database Connectivity (ODBC) driver	`psqlODBC`

1.2.3 Customer first support model: Free from penalties and prohibitive obligations

Fujitsu has been helping its customers modernize database solutions while providing data management solutions for both open source PostgreSQL and with the enhanced enterprise-grade FUJITSU Enterprise Postgres. Thus, unlike proprietary database management software, Fujitsu customers can always move from open source PostgreSQL to FUJITSU Enterprise Postgres and vice-versa because Fujitsu does not lock its customers to restrictive contracts. Figure 1-4 on page 7 shows that FUJITSU Enterprise Postgres has the OSS standards of PostgreSQL that are further fortified with Fujitsu enterprise-grade features, which are supported by a 24x7 product support model.

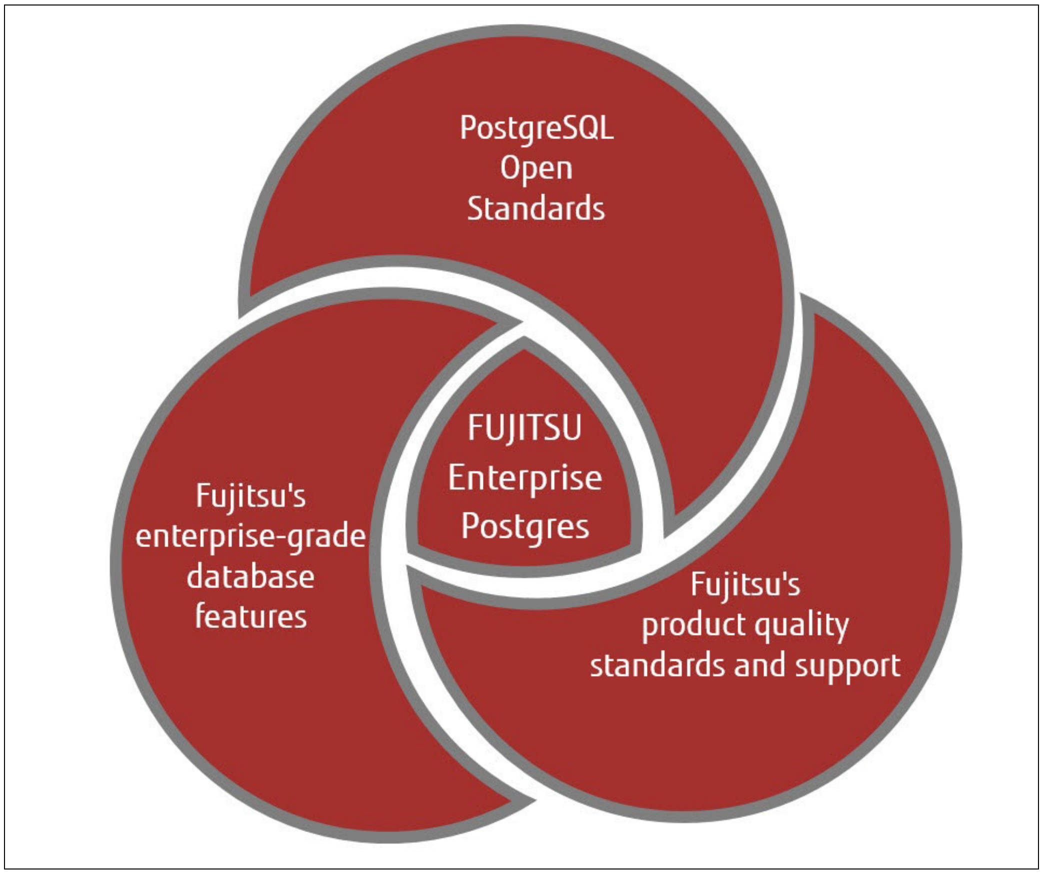


Figure 1-4 Open standards that are enhanced with enterprise features with 24x7x365 product support

Because FUJITSU Enterprise Postgres is based on open source PostgreSQL, customers always have compatibility and migration if they chose to opt for open source PostgreSQL over security enhanced FUJITSU Enterprise Postgres because there are no penalties or prohibitive obligations.

1.2.4 Beyond PostgreSQL: Fujitsu support for open source

Fujitsu has 15+ years of experience contributing to open source software development. Fujitsu contributes to PostgreSQL and other open source projects. As one of the founding members of the Linux Foundation starting in 2010, Fujitsu has been contributing to the development of the Linux operating system (OS) since 2005. Apart from the Linux Foundation, Fujitsu is also part of other open source projects, such as Cloud Foundry, the Open Container Project, the OpenStack Foundation, Open Network OS, and Open Daylight.

1.2.5 IBM and open source

IBM has been on a 20-year journey in collaboration with the open source community, from offering subscriptions and support for all major distributions on Linux across its IBM Z, IBM Power Systems, and IBM Cloud® offerings, to its recent acquisition of Red Hat. IBM has a track record of delivering support with a 99% fix rate for both commercial enterprise open source software, such as from Red Hat and SUSE, and community open source software, with specialized advice on deploying and optimizing software packages on IBM systems and IBM hybrid cloud platforms.

Open Hybrid Cloud

Open hybrid cloud with IBM LinuxONE is a combination of benefits from the IBM Z server platform, which has its roots in IBM systems that are designed for security, resilience, and scalability, with the openness of the Linux OS and enterprise-scale Kubernetes from Red Hat. Seeking to gain a competitive advantage in digital transformation, organizations want flexibility, simplicity, and a better economic equation for their applications in the cloud. IBM hardware and software that is combined with Red Hat platform middleware offers a true enterprise-class hybrid multicloud solution that is highly scalable and open.

- IBM LinuxONE offers flexible compute and horizontal scalability that can grow to millions of containers.
- Hardware that is designed for security, resilience, and scalability.
- Containerized applications offer great portability across hybrid cloud environments, including on-premises and public, and architectures like Intel x86, IBM Power Systems, and IBM LinuxONE.
- Red Hat OpenShift Container Platform 4 (RHOCP) is a supported Kubernetes distribution that is configured by leading enterprise practices, and security and performance.

1.3 Cloud-native containerized FUJITSU Enterprise Postgres

As organizations continue to focus on their digital transformation, they must change the way that they manage their existing data assets. Migration to cloud is seen as a fundamental step in an organization's digital transformation, and as applications migrate to the cloud databases naturally follow them into cloud environments such as AWS, Azure, and IBM Cloud. The proliferation in the number of cloud-hosted applications with a tremendous increase in the scale of the data that is generated and consumed by such applications is creating an unprecedented demand for cloud-hosted databases.

Containers and container orchestration have matured to the point that they are now positioned at the core of digital transformation by using cloud-native initiatives. Databases are increasingly popular candidates for containerization as they become an on-demand utility as part of the shift toward microservices and serverless architectures, which are becoming a critical component in the software stack of these new digital applications.

FUJITSU Enterprise Postgres 12 Operator is available on Red Hat OpenShift Container Platform (RHOCP). Figure 1-5 on page 9 shows the features of RHOCP. Figure 1-6 on page 10 shows the HA database implementation that uses FUJITSU Enterprise Postgres 12 Operator on RHOCP 4 - Kubernetes platform. For more information about FUJITSU Enterprise Postgres on RHOCP, see Chapter 10, “FUJITSU Enterprise Postgres cluster on Red Hat OpenShift Container Platform” on page 373.

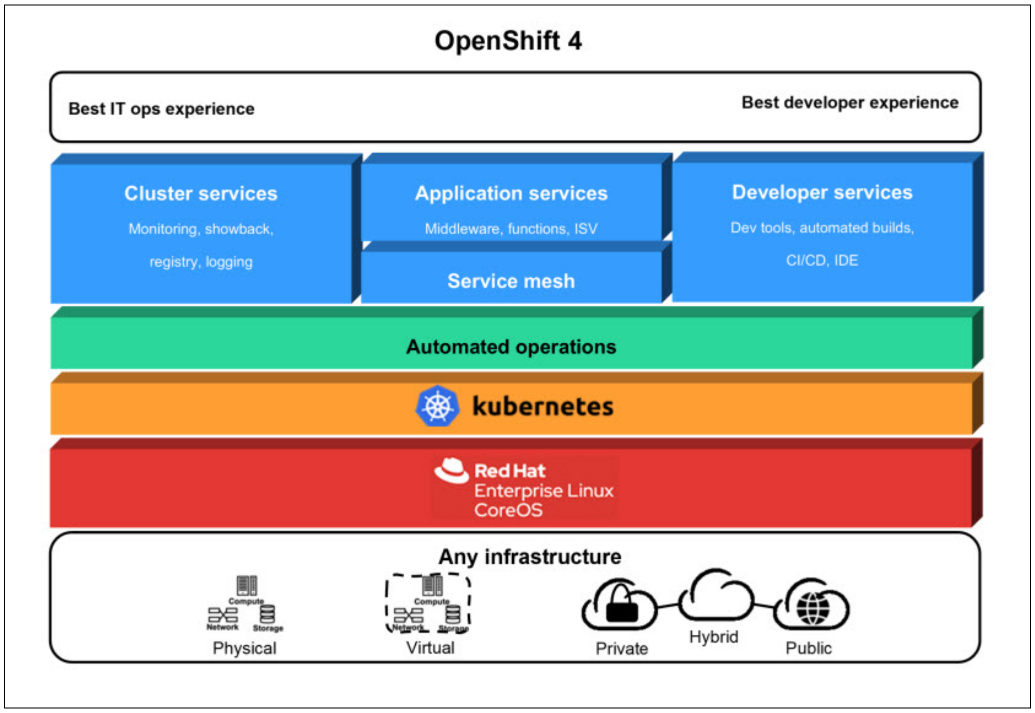


Figure 1-5 Red Hat OpenShift 4: Kubernetes platform

An installation that is composed of an enterprise-grade database such as FUJITSU Enterprise Postgres that is running in a container environment can provide many advantages over a standard cloud database. With the transition of applications to a microservices architecture, container deployment of RDBMS is moving to the mainstream, and PostgreSQL is the third most popular technology to run in containers on a Kubernetes platform that manages applications at massive scale.

Containerized databases have emerged as building blocks for shifting large, monolithic applications to workloads based on microservices and serverless architectures. However, containerizing a database is not as straightforward as containerizing an application. Fujitsu has leveraged its Enterprise PostgreSQL knowledge and experience and created a Managed Postgres Cluster offering to organizations to use as part of their cloud-native container journey.

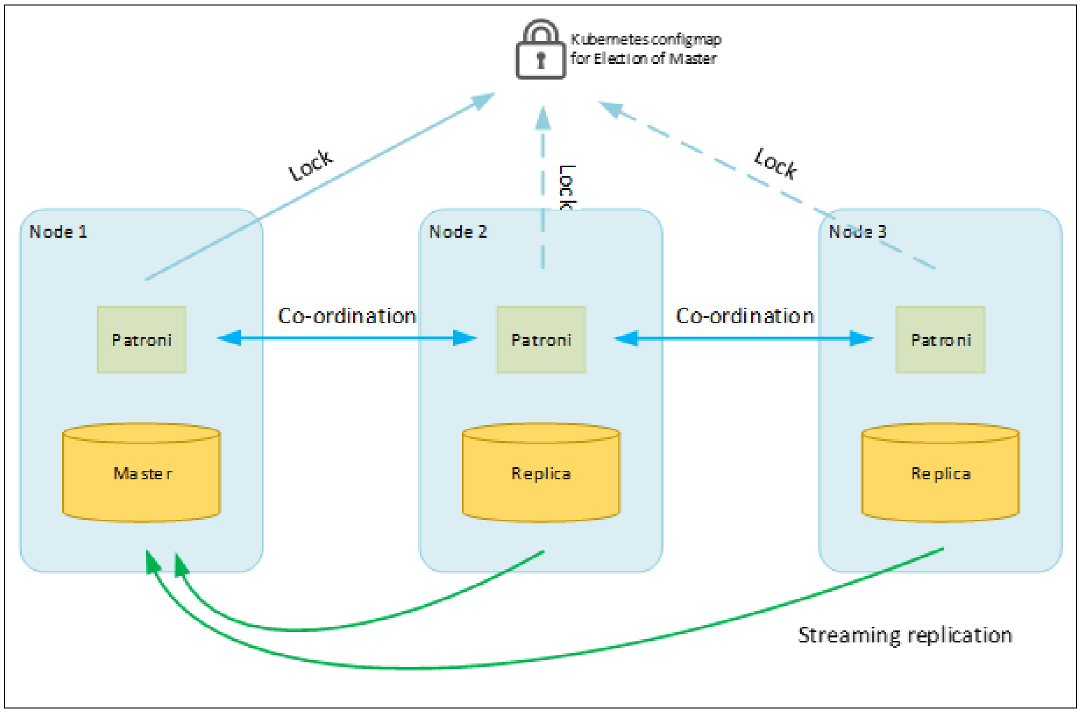


Figure 1-6 High availability FUJITSU Enterprise Postgres cluster with Leader Election

1.3.1 Containerized FUJITSU Enterprise Postgres key benefits

A FUJITSU Enterprise Postgres Database can be set up in a container platform to provide an enterprise-level database that is comparable to the cloud-managed Amazon RDS DBaaS without the disadvantages of it, such as data breaches, data loss, and inflexibility. A FUJITSU Enterprise Postgres Database in a managed containerized environment provides the following benefits:

- On-demand utility: A microservices architecture of smaller container applications can have their own dedicated database that is configured on as needed basis.
- Capacity planning and provisioning: Using separate storage nodes, Kubernetes Storage Class, and Persistent Volume features, storage performance and capacity can be scaled independently of compute resources, which provide more flexibility in upfront database capacity and makes changes easier to implement.
- High velocity DevOps: Software-defined containerized databases provide a crucial missing link in high-velocity DevOps cycles, allowing development and operations teams to collaborate seamlessly.
- High-throughput and low-latency networking: The emergence of container orchestration such as Red Hat OpenShift provides networking and data storage and network resource isolation that is necessary to achieve the required performance.
- Managing numerous tuning parameters: Custom database configurations parameters that use ConfigMaps and Custom Resource Definitions (CRDs) within a Kubernetes cluster can be passed into the container at run time to provide easier version control and dynamic control of DB management.
- HA and failover management: Using automated scripted deployment of containerized databases and an orchestration framework of Red Hat OpenShift provides a built-in HA for failover scenarios without needing to maintain a failover cluster replica. This configuration saves resources that are idle much of the time.

- Hybrid cloud / multi-cloud deployments: FUJITSU Enterprise Postgres running on Red Hat OpenShift based Kubernetes platform provides a cloud-neutral RDBMS solution that can be deployed across any cloud environment-supporting hybrid / multi-cloud strategy without needing to change the application code.
- Software upgrades: Leveraging the Red Hat OpenShift Operator framework, it is easier to plan and run a database software upgrade with or without a schema change. The software upgrade can be managed to minimize business disruptions, including the option to skip upgrades to maintain compatibility with other applications, which removes a critical dependency on Amazon RDS. Providing this level of control interferes with the automatic environment upgrade process.

Note: Fujitsu provides a demonstration video for FUJITSU Enterprise Postgres Operator at FUJITSU Enterprise Postgres Videos.

1.4 Data security

In the field of computer security, Anderson's Rule in the domain of database security states that if you plan to design a database at scale with functions and security, but you also want ease of access, then it is highly likely that your database will be insecure.[1] Paradoxically, Anderson's Rule also states that if you tightly secure your database, then it is likely that the database will be impossible to use.

For these reasons, balancing database security and its ease of use is at the center of modern database security developments. Concurrently, malicious hackers around the world are devising sophisticated attack methods and security threats by using vulnerabilities to cause data breaches that have left organizations with lowered reputations, low customer confidence, financial impact due to non-compliance, and loss of future business prospects. It is estimated that 1.7 MB of data is created every second for a single person in 2020. Additionally, 90% of the world's data was created in the last two years. This astonishing rate of data creation makes data security even more paramount for enterprises. Fixing the vulnerabilities at every layer that lead to the data assets in the database management system is no longer a “once fixed, forever fixed” scenario.

This section describes some methods of security risk mitigation and options.

1.4.1 Vulnerability management

The most effective way to fix a vulnerability is through proactive vulnerability management. Vulnerabilities are not limited to the database management system as a product but also might originate from processes that are defined around the use of the database management system. SQL injections are the most common and popular method that hackers use to get unlimited access to sensitive information by embedding malicious code into applications that are integrated with database management systems. Another common vulnerability is weak or a lack of encryption for the database and a complete lack of encryption for backed-up copies of the database. If there are holes in the network of the enterprises, then lack of encryption of the databases can easily expose the sensitive data to unethical hackers. Any unchecked vulnerability leaves the doors open for new security threats.

[1] https://en.wikipedia.org/wiki/Anderson%27s_rule_(computer_science)

1.4.2 Threat management

A security threat is described as a threat to the confidentiality, integrity, and availability of information assets. The threat encompasses technological risks, but does not entail physical damage, such as accessing a database. An unprotected environment can lead to unauthorized access of the database. In general, a security threat is a culmination of the source of the threat, assets that must be secured, procedures, processes, and the type of unauthorized access. A typical example might be a system administrator accessing the business data that is stored in the database through unauthorized access using the database vulnerability, and eventually being successful in viewing or updating the business data.

Hardening data and database security is a collaborative effort that requires an enterprise approach for standardization of security policies and procedures. With a collaborative approach, an enterprise is better prepared for minimizing data breach risks and exposures even in non-production environments. Collaborative efforts are even more effective when the enterprises use a proactive top-down approach to engage IT administrators, data stewards, data officers, IT compliance, auditors, and legal teams. Through collaboration, classifying the risk to the data becomes easier.

Collaboration provides a holistic view of security risks that are associated from the creation to the archiving of the data. When we are talking about database management systems and their security, it is not just the security of the database management system but also the layers above the database management system that are part of the overall security landscape and architecture. Whether it is about security at the network layer or the OS layer, a security threat might penetrate through all the layers before it reaches information assets that are stored in a database management system. For this reason, it is imperative to understand the various security threats, and so collaboration is crucial to scope security measures to maintain the integrity, confidentiality, and availability of the database.

1.4.3 Transparent Data Encryption by using IBM Crypto Express adapters

FUJITSU Enterprise Postgres on IBM LinuxONE provides various enterprise-grade data security features covering encryption, data obfuscation, and proactive auditing of the database logs. Because FUJITSU Enterprise Postgres is an enhancement of the OSS community PostgreSQL, all its security features are 100% PostgreSQL-compatible. Depending on your organizations' needs, the data at rest can be encrypted by using Transparent Data Encryption (TDE). TDE is integrated with IBM LinuxONE Crypto Express adapters (see Figure 1-7 on page 13), which enable an organization to encrypt the data by using HSM. No one has access to the Master Encryption Key (MEK) because it is HSM-managed, which makes the underlying data tamper-proof. In addition, IBM CEX6S is validated to meet FIPS 140-2 Level 4 certification, which is the highest level of security.

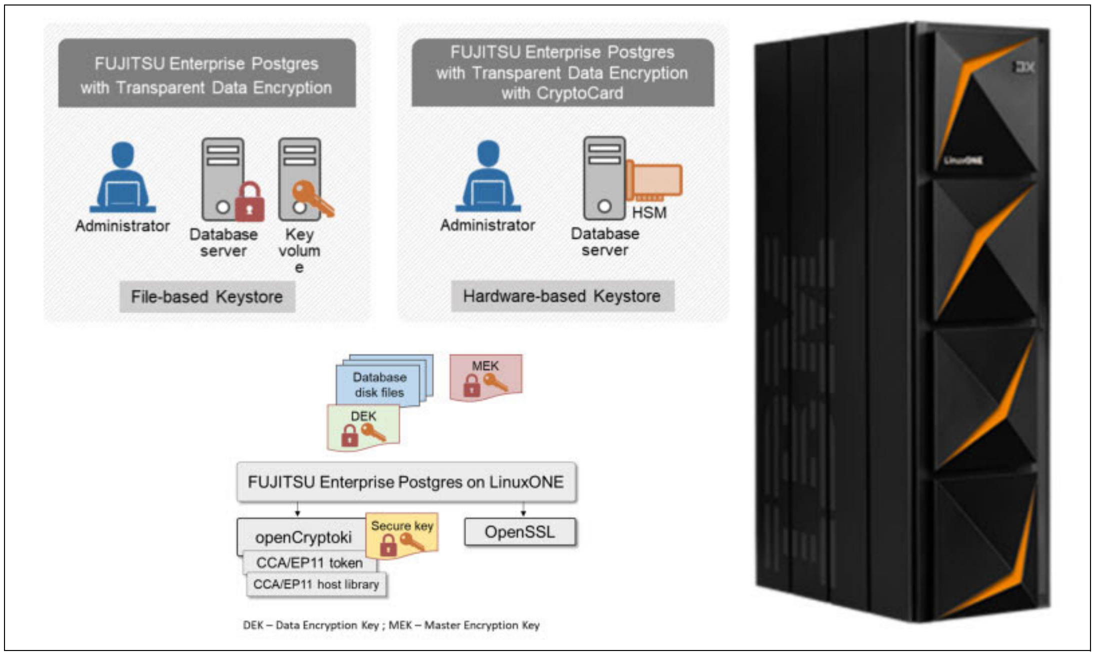

FUJITSU Enterprise Postgres with Transparent Data Encryption
Administrator
Database server
Key volume
File-based Keystore
FUJITSU Enterprise Postgres with Transparent Data Encryption with CryptoCard
Administrator
HSM
Database server
Hardware-based Keystore
Database disk files
MEK
DEK
FUJITSU Enterprise Postgres on LinuxONE
openCryptoki
Secure key
CCA/EP11 token
CCA/EP11 host library
OpenSSL
DEK – Data Encryption Key ; MEK – Master Encryption Key

Figure 1-7 FUJITSU Transparent Data Encryption by using openCryptoki and IBM LinuxONE Crypto Express adapters

Transparent Data Encryption features

Here are some of the features of TDE:

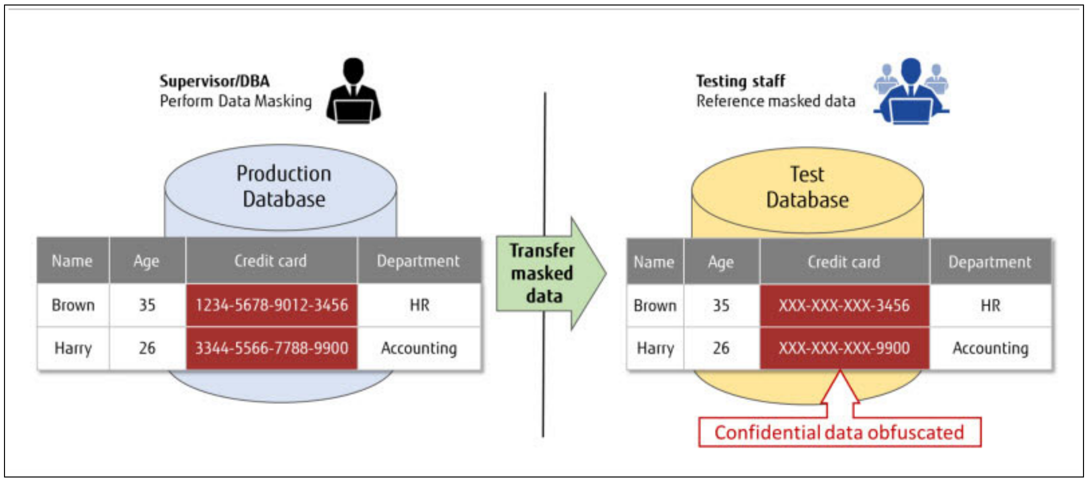
- TDE uses the Advanced Encryption Standard (AES) as its encryption algorithm. AES was adopted as a standard in 2002 by the United States Federal Government and is used throughout the world.
- TDE provides encryption at the file level (AES-128 and AES-256), essentially solving the issue of protecting data at rest.
- TDE fulfills requirements for the Payment Card Industry Data Security Standard (PCI DSS) and allows confidential information such as credit card numbers to be made unrecognizable on disk.
- Data is automatically encrypted and decrypted when it is written and read; manual key management is not required. Even if an attacker gets through all the access controls and connects to the server, they cannot access the data because the OS file is encrypted.
- Encryption is extended to *Write-Ahead-Logs* (WALs) (also known as transaction logs), backups, temporary tables, and temporary indexes, thus providing comprehensive security.
- The encryption algorithm does not alter the size of the object that is being encrypted, so there is no storage overhead.
- Flexibility to encrypt a subset of the data per organizations data classification rules and policies.
- Both file-based and hardware-based keystore management mechanisms are supported.
- Flexibility of using multiple Data Encryption Keys (DEKs), which are again encrypted by the MEK.

- On IBM LinuxONE, MEK is further secured with an IBM LinuxONE CryptoCard HSM managed Secure key.
- Take advantage of the CPACF in the IBM Z processor to minimize encryption and decryption overhead so that even in situations where previously the minimum encryption target was selected as a tradeoff between performance and security, it is now possible to encrypt all the data of an application.

1.4.4 Data masking: Policy-based dynamic data obfuscation technique

Although your organization decided to encrypt the data, there might be scenarios where data obfuscation or data masking is essential. Whether you want to share your data with market researchers or share medical records with medical researchers, sharing personally identifiable data or payment card information without obfuscation might lead to regulatory fines. This situation might result in many problems depending upon the number of governing bodies investigating the data breach case. Therefore, data masking is as important as data encryption.

As shown in Figure 1-8, FUJITSU Enterprise Postgres uses a dynamic data masking technique that is built on a policy-based data obfuscation process.



Name	Age	Credit card	Department
Brown	35	1234-5678-9012-3456	HR
Harry	26	3344-5566-7788-9900	Accounting

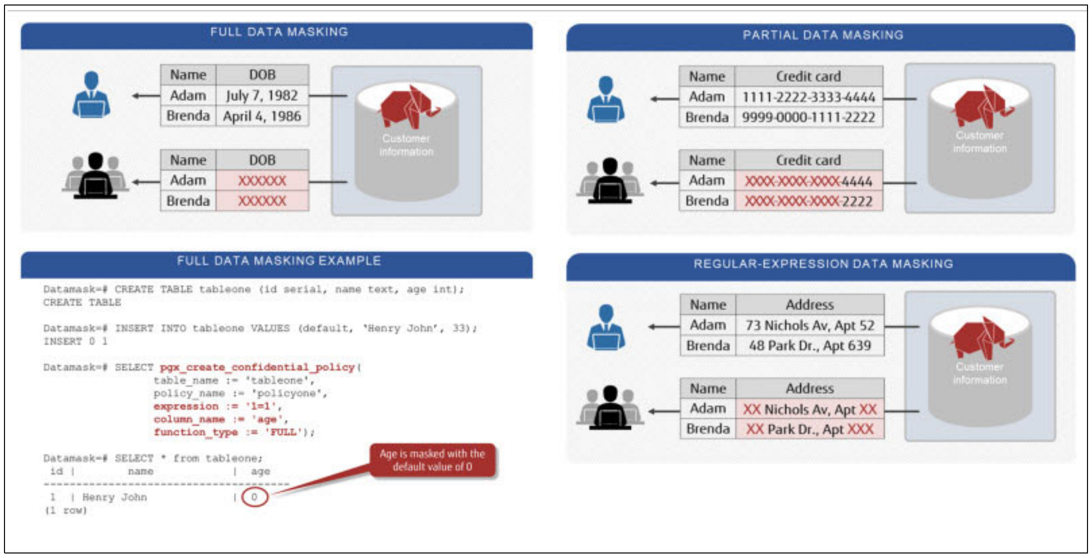
Name	Age	Credit card	Department
Brown	35	XXX-XXX-XXX-3456	HR
Harry	26	XXX-XXX-XXX-9900	Accounting

Figure 1-8 Data masking for test data management

The biggest advantage of FUJITSU Enterprise Postgres Data Masking is that the existing SQLs require no change, but the results are presented to the user depending on their role and the operational data masking policy on the underlying FUJITSU Enterprise Postgres Database objects. This advantage presents a huge value to the application teams and to the organization because this data masking technique removes the overhead of changing the application to use Fujitsu policy-based dynamic Data Masking feature.

An effective implementation of data masking ensures that the only data that is visible is that which does not expose personally identifiable information, health data, or financial data. There are various types of data masking techniques, and they can be categorized as *persistent data masking* and *dynamic data masking* techniques. Persistent data masking techniques irreversibly and permanently change the data, and dynamic data masking techniques simply alters the appearance of the data without changing the real data in the database management system.

The three types of data masking are shown in Figure 1-9 on page 15.


FULL DATA MASKING
Customer information
PARTIAL DATA MASKING
Customer information
FULL DATA MASKING EXAMPLE
REGULAR-EXPRESSION DATA MASKING
Customer information
Age is masked with the default value of 0

Name	DOB
Adam	July 7, 1982
Brenda	April 4, 1986

Name	DOB
Adam	XXXXXX
Brenda	XXXXXX

Name	Credit card
Adam	1111-2222-3333-4444
Brenda	9999-0000-1111-2222

Name	Credit card
Adam	XXXX-XXXX-XXXX-4444
Brenda	XXXX-XXXX-XXXX-2222

Name	Address
Adam	73 Nichols Av, Apt 52
Brenda	48 Park Dr., Apt 639

Name	Address
Adam	XX Nichols Av, Apt XX
Brenda	XX Park Dr., Apt XXX

```
Datamask=# CREATE TABLE tableone (id serial, name text, age int);
CREATE TABLE

Datamask=# INSERT INTO tableone VALUES (default, 'Henry John', 33);
INSERT 0 1

Datamask=# SELECT pgx_create_confidential_policy(
                table_name := 'tableone',
                policy_name := 'policyone',
                expression := '1=1',
                column_name := 'age',
                function_type := 'FULL');

Datamask=# SELECT * from tableone;
 id |          name            |  age
-------------------------------------
 1  | Henry John               |  0
(1 row)
```

Figure 1-9 Types of data masking: Full, partial, and expression-based

FUJITSU Data Masking features

Here is a list of FUJITSU Data Masking features:

- Flexible: After policy-based data masking is applied, the policies can be disabled or enabled as required without having to remove or reapply them.
- Customizable: Policy-based data masking allows development of data-sensitive masking policies for different data classifications. Data masking policies can be applied to tables for different columns that come under one data classification or another.
- Irreversible and tamperproof: The FUJITSU Data Masking feature is irreversible because the relationship between original data and masked data is severed during the masking policy implementation, which de-risks any reverse engineering for re-creating the production data.
- Dynamic data masking: A masking policy is applied at the time that data is accessed, so queried data is modified according to the masking policy before results are returned in the query chain.
- Role-based: Data masking policies can be configured and applied for specific roles / conditions so that the original data cannot be viewed without appropriate privileges.
- Data integrity maintained: Referential integrity is maintained in the source database as the data masking policy is applied during the data access process.
- Reusability: Masking policies generate data that can be same every time that masked data is accessed, which maintains test data consistency across multiple iterations of accessing original data.
- Policy metadata: Availability of catalog tables for querying data masking policy information to assess the current sensitive data policy state of a database.

1.4.5 Dedicated Audit Logs

Although a data encryption and data masking implementation proactively mitigate the risks arising from application development and test data management process vulnerabilities, there always is a mandatory requirement to review the user activity to ensure that any new threats are managed by the information security division of your organization. Auditing the database logs periodically while also maintaining regulatory compliance, policies, and standards is one of the non-negotiable processes.

Log auditing: OSS PostgreSQL and FUJITSU Enterprise Postgres comparison

Database log auditing is not new, although over the years it has become more exhaustive and sophisticated. FUJITSU Enterprise Postgres has a unique implementation of separating the system and audit logs. Fujitsu implemented a dedicated audit log capability that is a differentiator from open source PostgreSQL. Although open source PostgreSQL logs session and object records in the same server log, it creates an unavoidable overhead and complexity for log analysis. FUJITSU Enterprise Postgres solves this problem with the feature known as *Dedicated Audit Logs*. The Dedicated Audit Logs feature saves the log records in different files based on their purpose. Server logs are saved in system messages, and obtaining audit information is simplified through the audit logs, which saves the audit trail in a separate file other than the server log.

Figure 1-10 shows a comparison between the audit log feature of OSS PostgreSQL and FUJITSU Enterprise Postgres.

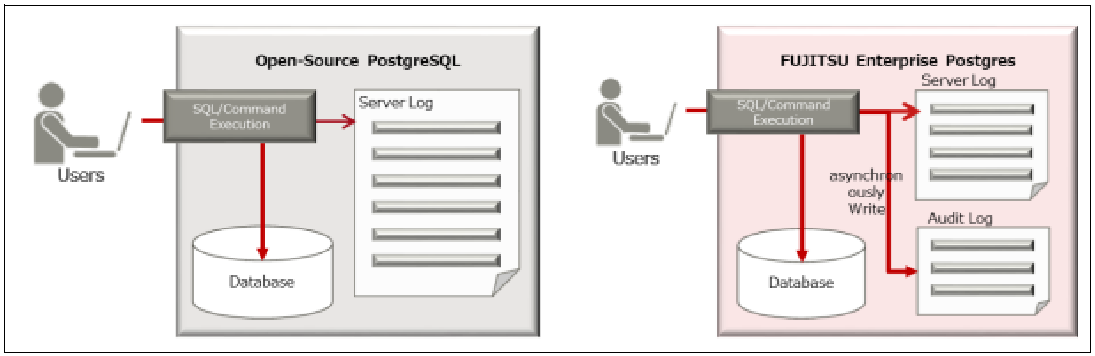


Figure 1-10 Audit Log feature comparison between OSS and FUJITSU Enterprise Postgres PostgreSQL

Audit log features

FUJITSU Enterprise Postgres Audit logging enables organizations to trace and audit the usage of sensitive data and connection attempts to the database. Audit logging provides a clear picture of data access by logging following details:

- What data is accessed.
- When the data is accessed.
- How the data is accessed and who accessed the data.

The information that is provided by audit logs can be used to counter security threats such as:

- Spoofing
- Unauthorized database access
- Privilege misuses

Regardless of the nature of your organization's business, robust data and database security help your organization to reduce risks, drive business growth, and add competitive advantage. FUJITSU Enterprise Postgres on IBM LinuxONE is a combination that enterprises are looking for. As the volume of data that is stored in an IT system increases, so does the risk of data leakage. All data in FUJITSU Enterprise Postgres can be protected by using TDE, which uses the same algorithms that are used by the US government, which provides data owners with the highest levels of security. FUJITSU Enterprise Postgres enhances security at many levels, including data encryption, log encryption, and internal communication channel encryption.

1.5 Data integrity

Data integrity and operational continuity are paramount for any organization. Data loss is a significant risk in the event of a physical disk failure. FUJITSU Enterprise Postgres mitigates this risk through data redundancy across physical hardware and reliably coordinating this distributed data during automated fail-over and recovery processes.

1.5.1 Data integrity as a state versus data integrity as a process

Although it is necessary to promote a culture of integrity to reduce data integrity risks, it is through the people, processes, and technology together that organizations can truly maintain data integrity through the entire lifecycle of data entities. Data integrity is both a state and a process. When data integrity is seen as a state, it signifies the accuracy, validity, maintainability, and recoverability of the data assets. Data integrity as a process signifies the steps that are implemented to maintain the reliability and efficacy of the data entity during the stages of data lifecycle. There are various stages throughout the data lifecycle that a data entity moves across and is transformed. Broadly, these stages might be categorized as:

- Data creation
- Data collection
- Data processing
- Data distribution
- Data discovery
- Data analysis
- Data reprocessing
- Data archival

As the data entities move through the data life-cycle stages, the data is stored in various formats and operated on, so the people, processes, and tools for managing the data storage and operation are highly critical to maintain the data integrity throughout the data lifecycle.

1.5.2 Data integrity and data security: The fine differences

Although data integrity and data security might seem similar, they are often confused. Data security is one of the aspects of achieving data integrity, so they are not synonymous. Incidents that lead to data loss have been increasing over time. Whether it is a ransomware attack, natural disaster, disk failure, network failure, or the lack of a robust backup and recovery strategy, any or all of these incidents might be a reason for creating data integrity issues and a major financial, commercial, and regulatory setback for the enterprise. Furthermore, it is even more important from a technology perspective that the underlying database management system provides enough ways and means to recover the data assets to an acceptable level of integrity that maintains the accuracy and reliability of the data.

One one way to achieve higher probability of successful recovery is to create a redundancy of the features that might become a single point of failure. Transaction logs (WALs) are one such critical function of OSS PostgreSQL that might become a single point of failure. FUJITSU Enterprise Postgres solves the issue of single point of failure by creating redundancy in transaction logging through the enterprise-grade feature that is known as Transaction Log Mirroring or WAL duplexing.

1.5.3 Transaction log mirroring or WAL duplexing

Transaction logs are the most trusted way to ensure data integrity in almost all RDBMSs. In PostgreSQL, the concept of transaction logging is implemented as WAL. As the name suggests, the idea is to write the transactions to a transaction log or WAL, and then to the disk. In this way, PostgreSQL ensures that, in the event of system failure, data integrity is not compromised. Fujitsu has developed and integrated enterprise-grade features that are known collectively as WAL duplexing, which enables mirroring the database transaction logs, and therefore prevents data loss that can occur, for example, because of disk failure. Although OSS PostgreSQL has a feature to copy logs to another disk, it would not protect the enterprises in a scenario where the source disk fails before the transaction logs are copied, which creates a serious issue of non-recoverability and data integrity issues.

Figure 1-11 shows the process flow of OSS PostgreSQL single WAL versus FUJITSU Enterprise Postgres WAL duplexing feature.

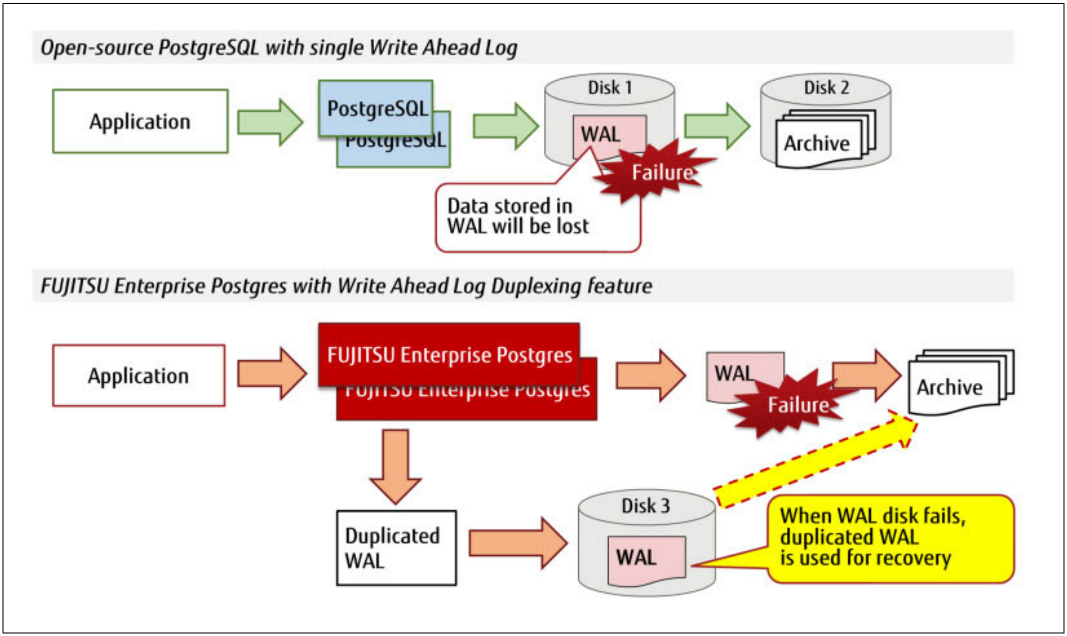

Open-source PostgreSQL with single Write Ahead Log
Application
PostgreSQL
Disk 1
WAL
Disk 2
Archive
Failure
Data stored in WAL will be lost
FUJITSU Enterprise Postgres with Write Ahead Log Duplexing feature
Application
FUJITSU Enterprise Postgres
WAL
Failure
Archive
Duplicated WAL
Disk 3
WAL
When WAL disk fails, duplicated WAL is used for recovery

Figure 1-11 FUJITSU Enterprise Postgres Write-Ahead-Log duplexing feature

To protect your business from a disk failure, FUJITSU Enterprise Postgres has a mechanism to write the transaction logs (or WAL) to two disks concurrently in parallel. Even if one disk fails, there always will be a copy of the transaction logs to recover and maintain the data integrity of the business transactions.

1.6 IBM LinuxONE performance and scalability

IBM LinuxONE is a highly scalable platform that can keep pace with your business growth cost effectively. Unique to the platform is a capability to scale both vertically and horizontally to suit any workload by leveraging the following key features:

- EAL5+ level isolation of LPARs, for example, production, staging, and dev/test can be deployed on the same hardware.
- Multiple levels of virtualization with zVM/KVM and containerization (see Figure 1-12).
- High granularity of hardware resource sharing (<1%).
- Capacity on Demand (CoD) options.
- Upgrade physical resources without system downtime.
- Single-system image (SSI) clustering: Provision virtual servers in seconds.
- Scalability of up to 1000s of virtual servers.

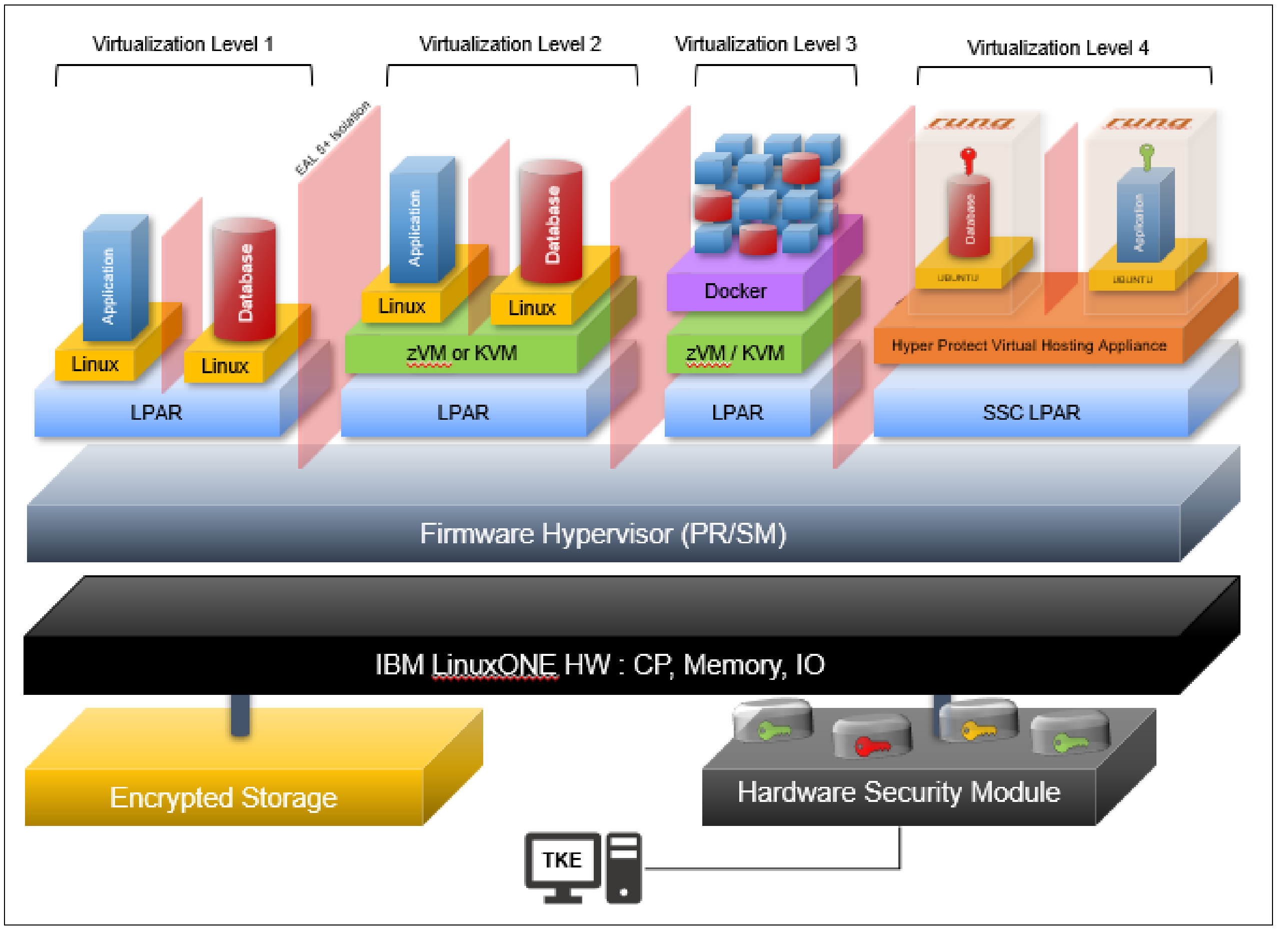


Figure 1-12 IBM LinuxONE virtualization levels

IBM LinuxONE is designed for extreme performance to effectively run the heaviest workloads at the lowest cost of any enterprise platform. With its impressive number of processors, clock-speed, I/O bandwidth, and more, IBM LinuxONE is designed to operate at near 100% utilization, delivering maximum return on investment (ROI) in the shortest time possible. Key performance enhancing features include:

- Extremely fast commercial processor (5.2 GHz) that enables consolidation of more than 1,000 x86 cores per frame.
- Up to 190 cores in four racks.
- Up to 70% reduction in software licenses.
- More than 2x higher CPU clock frequency than other commercially available CPUs.
- More than 2x better performance per core than other commercially available CPUs.
- The only commercially available processor designed for sustained utilization of 80 - 90%.
- Dedicated engines to offload I/O and network processing and HiperSockets for in-memory communication across LPARs.
- Workload prioritization: Use z/VM to meet tiered service-level agreement (SLA) requirements through dynamic and fine-grained prioritization. Utilization up to 100%.

1.7 Business continuity

Business continuity is built on the capacity of an organization to be resilient. It takes a lot more than just strategy, leadership, and vision to cultivate the culture of resiliency in an organization. Enterprise resiliency means that in the event of a disaster or disruption, the organization has built the capacity to maintain the business as usual and the capability to capitalize on the ever-changing business landscape.

1.7.1 Culture of resiliency through people, processes, and technology

While it is true that not every eventuality can be accurately predicted in order for a business continuity plan to be built to account for it, it is even more critical to instill a culture of resiliency across the people, processes, and technology dimensions of the organization.

Simply put, people, processes, and tools are *not* free from flaws. The people aspect of resiliency is a reflection of trust among peers and leadership of the organization. It is the same trust that transforms into reliability of processes and technology that the people of an organization design, maintain, and operate for a robust business continuity.

Figure 1-13 on page 21 shows the importance of synergy between people, processes, and technology to fully use and optimize HA, replication, and recovery.

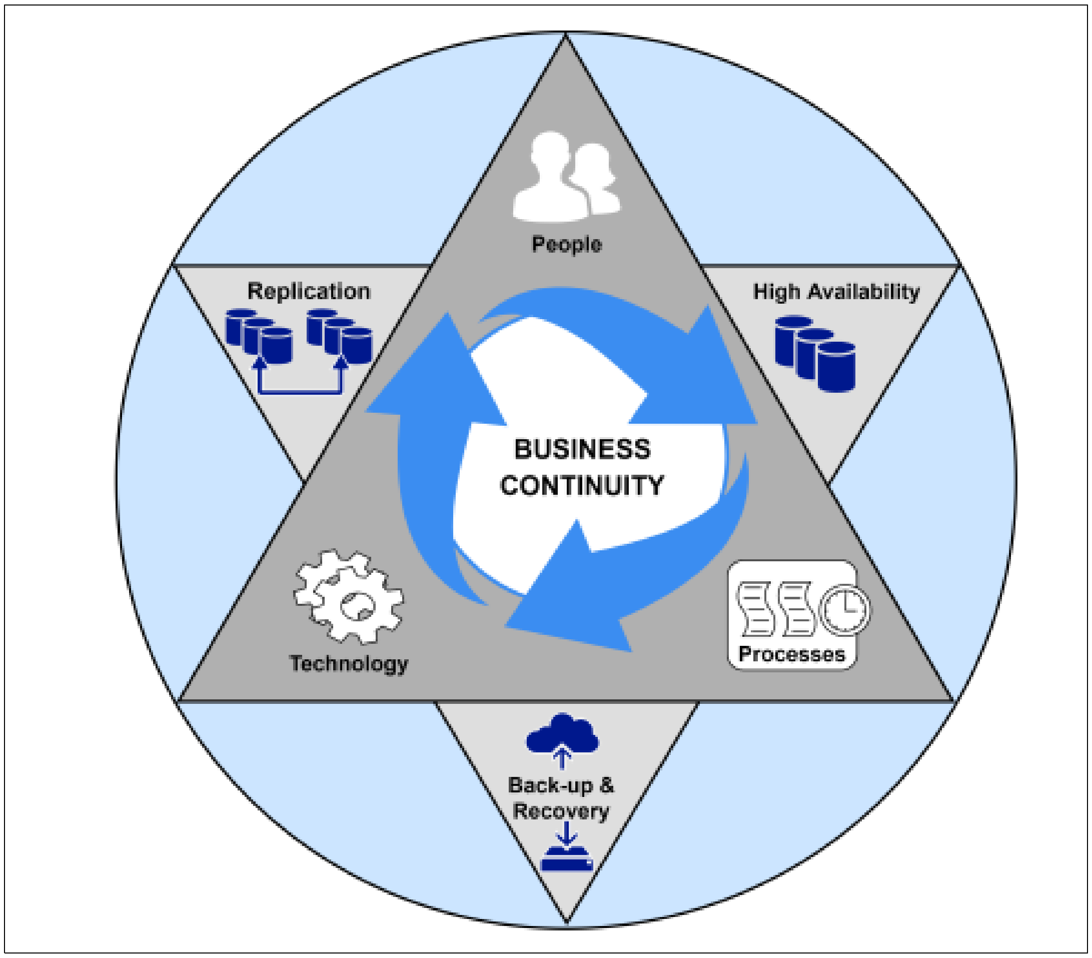


Figure 1-13 People, processes, technology, and business continuity

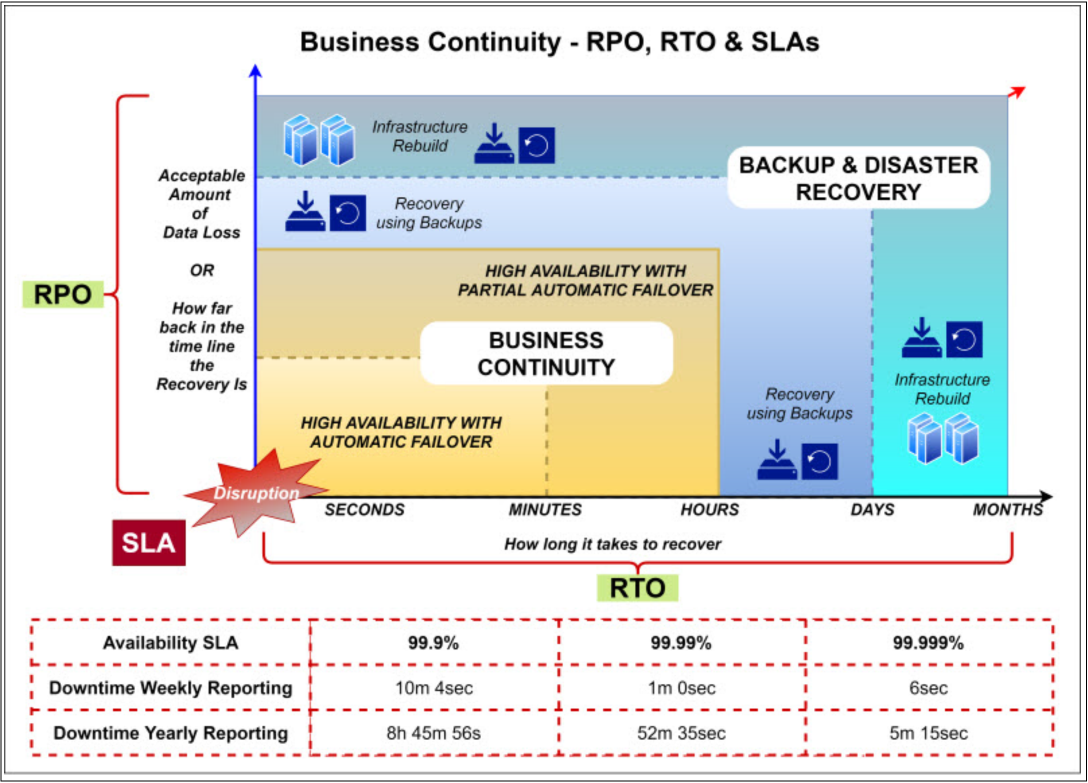
Although people and processes play a major role in maintaining resilient business continuity, the role of technology is to achieve operational excellence through automation. Often, business continuity is interchangeably used with HA and disaster recovery (DR). However, there are key differences between the three terms:

- Business continuity ensures that mission-critical services operate with minimal or no impact during a disaster through the effective implementation of an optimized business continuity plan.
- HA and backup and recovery are components of business continuity. Planning and implementation of HA IT architectures that complement seamless backup and recovery procedures ensures ROI of the technology. Furthermore, ROI is linked in a way to recovery time objective (RTO) and recovery point objective (RPO) that your organization would set for your IT systems.

1.7.2 RTO and RPO

From the database perspective, RPO means the amount of data that can be lost before the databases again become operational. Similarly, RTO implies the duration for which the database layer can be inoperative or unavailable to the business. Your organization might define RPO and RTO based on various factors, including the acceptable financial loss to the business or the acceptable outage duration that the business can manage without considerable financial loss. Depending on these factors, the RPO and RTO govern the SLAs of the database services.

Figure 1-14 shows that, for critical business services, the RTO for database recovery should be within a few seconds to minutes. If you have the availability SLA as 999.99% for your application, then in the entire year of reporting, the application cannot have more than 5 minutes and 15 seconds of acceptable downtime. As the availability SLA becomes more stringent, it requires architects to build more redundancy into the technology stack. From the application layer to the system layer, there must be redundancy and recoverability that are built into each layer.


Business Continuity - RPO, RTO & SLAs
Infrastructure Rebuild
Recovery using Backups
BACKUP & DISASTER RECOVERY
Acceptable Amount of Data Loss
OR
How far back in the time line the Recovery Is
RPO
HIGH AVAILABILITY WITH PARTIAL AUTOMATIC FAILOVER
BUSINESS CONTINUITY
HIGH AVAILABILITY WITH AUTOMATIC FAILOVER
Recovery using Backups
Infrastructure Rebuild
Disruption
SECONDS
MINUTES
HOURS
DAYS
MONTHS
SLA
How long it takes to recover
RTO

Availability SLA	99.9%	99.99%	99.999%
Downtime Weekly Reporting	10m 4sec	1m 0sec	6sec
Downtime Yearly Reporting	8h 45m 56s	52m 35sec	5m 15sec

Figure 1-14 Business continuity, high availability, and disaster preparedness

1.7.3 High availability and scalability with FUJITSU Enterprise Postgres

When it comes to the database layer, FUJITSU Enterprise Postgres comes with HA components that are built in. Depending on your database infrastructure SLAs, RPOs, and RTOs, you gain the advantage to create your own HA architectures. Whether you want to have primary and secondary database instances with automatic failover or your architecture demands distributing the application load across multiple read replicas, these goals can be achieved by using FUJITSU Enterprise Postgres Database Multiplexing, which uses PostgreSQ streaming replication technology to enable highly reliable database operation.

Furthermore, FUJITSU Enterprise Postgres also provides automated switchover and standby disconnection operations through the FUJITSU Enterprise Postgres Mirroring Controller (MC) and Server Assistant components.

Automatic failover by using FUJITSU Mirroring Controller

MC can automatically switch from a primary server to a standby server when the former is affected by disruption. Server Assistant prevents both the primary or the standby server from taking the role of primary server concurrently when there is a disconnection between the primary and the standby server.

Figure 1-15 shows the operation of MC for a situation where the primary database server operation is disrupted and the application switches over from the primary server to the standby server.

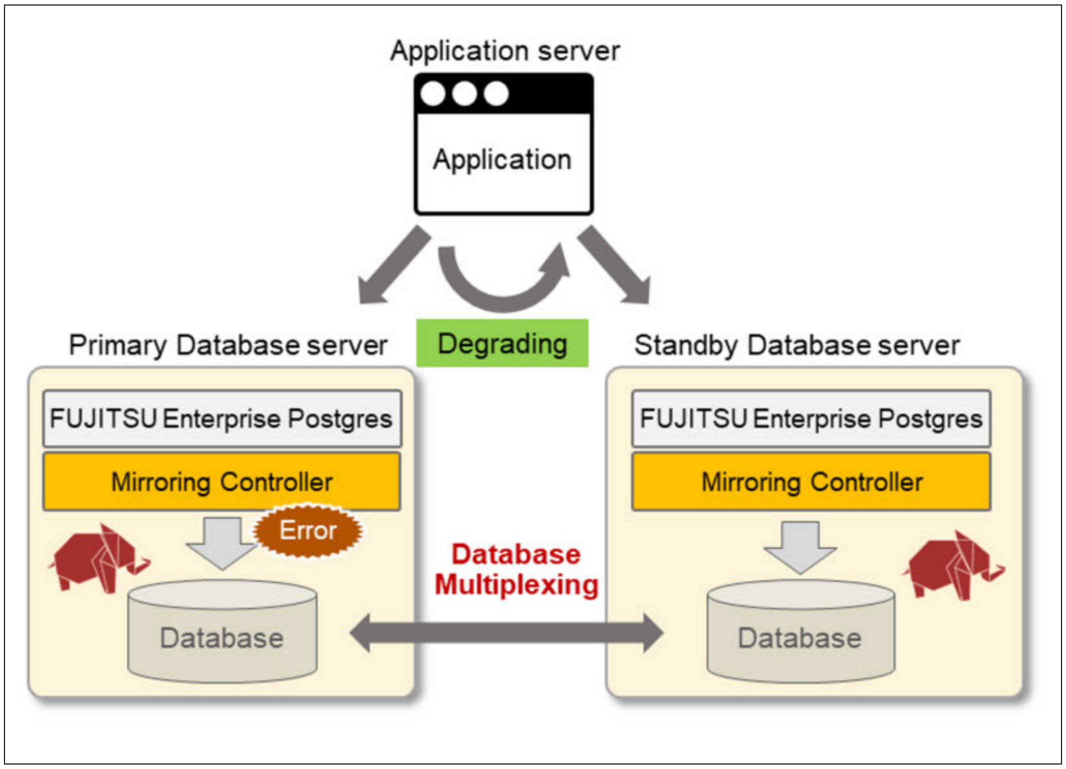


Figure 1-15 Mirroring Controller operation for high availability

Avoiding split-brain syndrome by using FUJITSU Server Assistant

Figure 1-16 shows a situation when MC might not be able to determine the status of the two servers if there is a network issue between the two database servers or one of the servers is not in a stable state of operation. In this situation, which is also known as *split-brain*, both servers might start acting as the primary database server, which can create data integrity issues because the application update operations become active on both database servers. This situation is handled by FUJITSU Enterprise Postgres Server Assistant software, which is installed on the arbitration server. Server Assistant ensures quorum for the cluster by acting as a witness server between the primary and standby FUJITSU Enterprise Postgres servers.

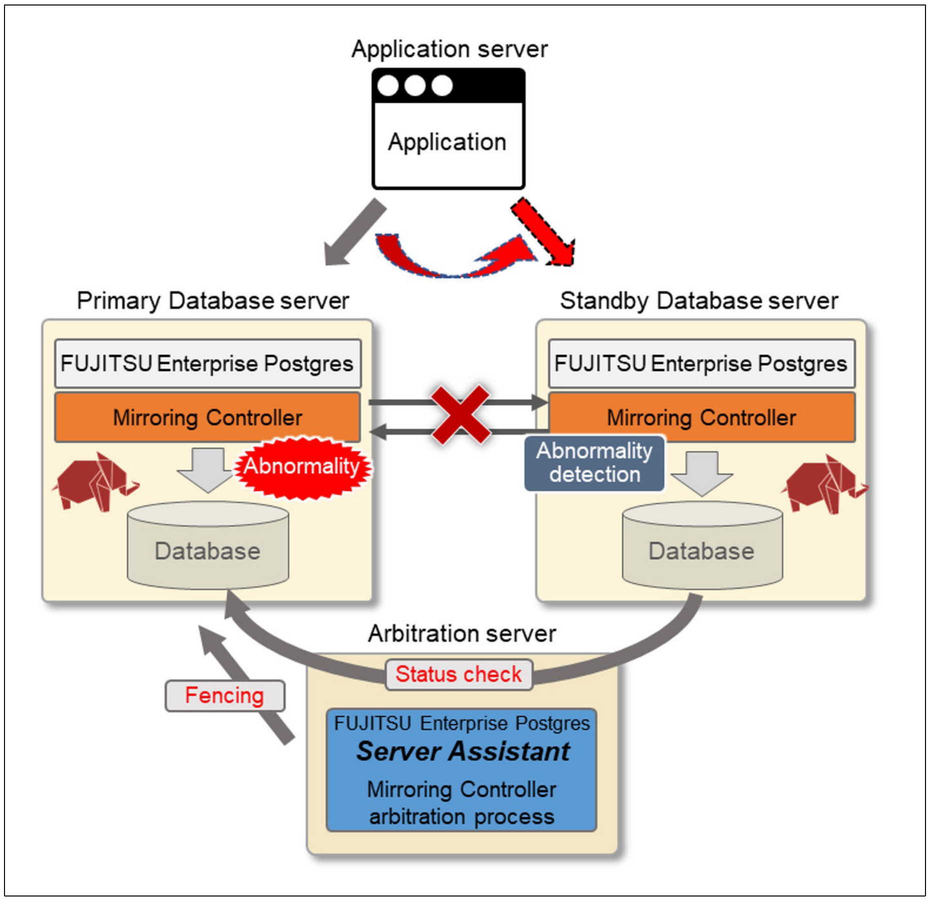


Figure 1-16 Server Assistant operation for high availability

1.7.4 Connection pooling, load balancing, and using Pgpool-II

Because your organization wants to achieve maximum availability for its applications and database instances, it must make sure that the HA does not become overwhelming for the database server processes. With the increasing number of client connections, you might need double-digit or a greater number of processor cores and memory to match the demand.

PostgreSQL is designed so that every client connection is treated as a process that consumes processor and memory resources, which means that with an increasing number of client connections, the OS starts saturating the processor resources and eventually impacting performance.

Although there are many solutions to balance the application workload on the database and the database performance, the most effective solution that is available is by introducing connection pooling software as a middleware between the application layer and the database layer. Pgpool-II and PgBouncer are two widely used connection pooling software with features such as connection pooling, connection switching, and load balancing in HA architectures. With strict RPO and RTO, it is important to have redundancy that is built in to every layer and include HA of the connection pooling software and avoid any layer becoming a single point of failure.

Figure 1-17 shows a typical HA architecture along with Pgpool-II as middleware between an application and the database server.

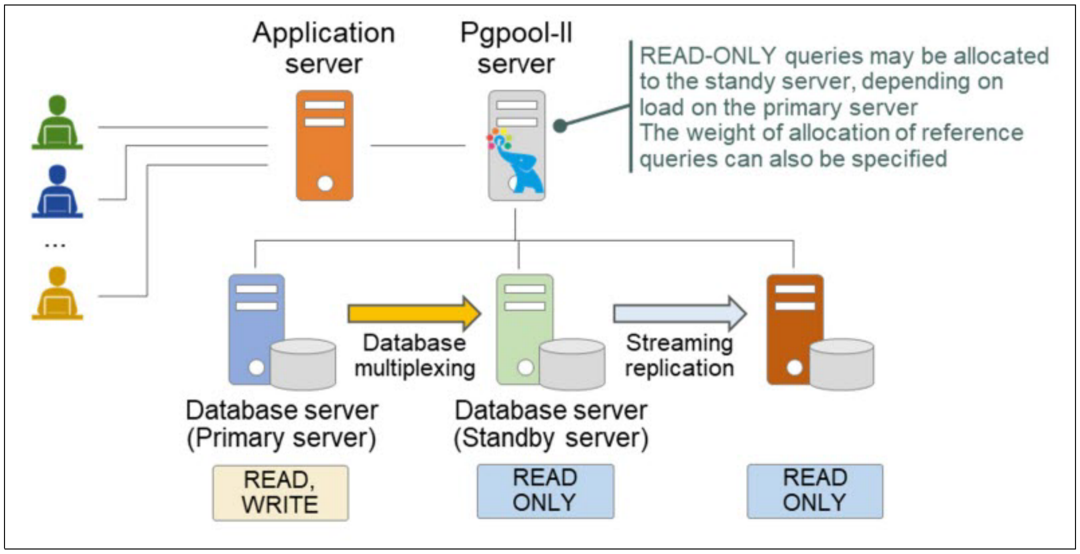

Application server
Pgpool-II server
READ-ONLY queries may be allocated to the standy server, depending on load on the primary server
The weight of allocation of reference queries can also be specified
...
Database multiplexing
Streaming replication
Database server (Primary server)
Database server (Standby server)
READ, WRITE
READ ONLY
READ ONLY

Figure 1-17 High availability and connection pooling

In Chapter 6, “Connection pooling and load balancing with Pgpool-II” on page 233, we describe steps to achieve HA and reliability by using Pgpool-II open source connection pooling software with FUJITSU Enterprise Postgres. Pgpool-II comes packaged with FUJITSU Enterprise Postgres and is fully supported by Fujitsu for fixing any bugs.

Transparent application switching and using Connection Manager

When the primary database server fails and an automatic switchover is initiated to promote the standby server to accept a read/write operation, FUJITSU Enterprise Postgres Connection Manager provides heartbeat monitoring and transparent connection support features. The Connection Manager monitors the client/server and FUJITSU Enterprise Postgres instances running on the database server. In the event of physical server failure and an inter-server network link being down, the Connection Manager notifies the client and the database instance. Figure 1-18 shows the operation of the Connection Manager.

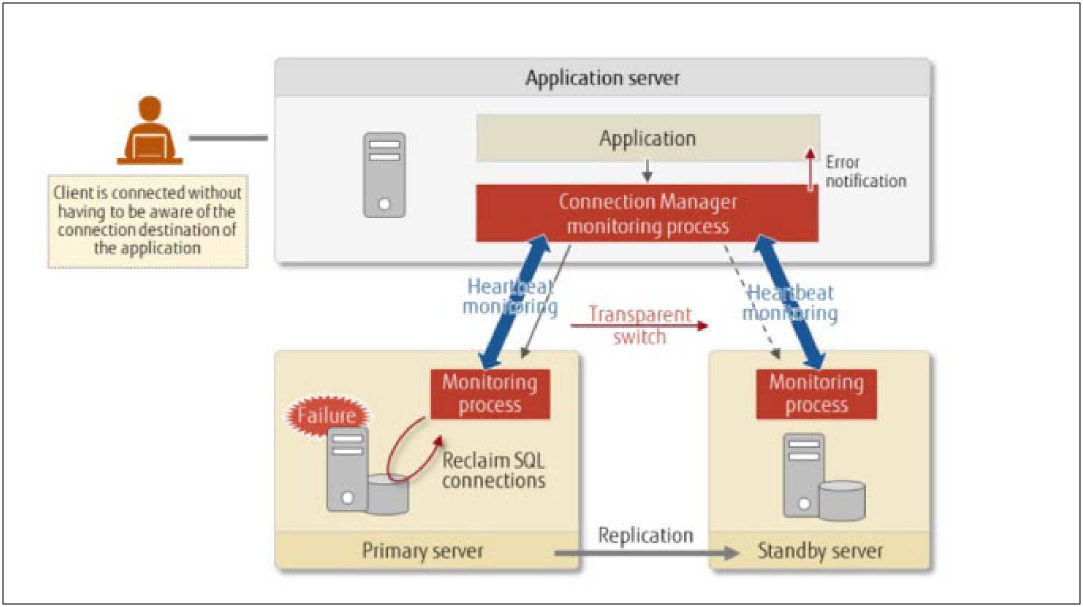


Figure 1-18 FUJITSU Enterprise Postgres Connection Manager showing application switching process

The Connection Manager transparent connection feature allows the application to connect to an instance from a set of instances that are configured for replication. The connection to an underlying database instance is transparent to the application. FUJITSU Enterprise Postgres Connection Manager is available by default with FUJITSU Enterprise Postgres V12. Connection Manager is described in detail in 7.2, “Connection Manager” on page 263.

There are various HA modes of operation that are possible when using FUJITSU Enterprise Postgres. PostgreSQL replication is the foundation of all HA and disaster preparedness architectures. We describe these modes of operation in detail in this publication. These modes of operation are:

- Cold standby
- Warm standby
- Hot standby
- Single primary with multiple subordinates, which can either be in a cascaded operation or a parallel operation that is connected to a single primary that uses native streaming replication or logical replication

It is usually the RPO, RTO, and SLAs that govern your decision to use any of these or a combination of these HA architectures for your mission-critical applications.

1.8 Disaster preparedness

Preparing for the worst situation is to minimize the effects of a disaster. Any disaster, small or large, can disturb the operational efficacy of a system and bring loss to your business, and materials and lives. A disaster is also a test of the resiliency of the people, processes, and technology of an organization, country, and region.

With systematic planning, foolproof vulnerability assessment, installing HA monitoring and warning systems, and well-coordinated disaster drills, an organization can control the extent of the damage, and in some cases altogether avoid any major damage. The classic example is Japan's skyscrapers, which are built to withstand many earthquakes throughout the year, small or large on the Richter scale. Although natural disasters might be the worst situations impacting your business, there are various other reasons that your organization might need means and measures to protect the data so that it always can be recovered and your organization can continue its business as usual.

There are various best practices that an organization should follow for backups that can be used for recovery. FUJITSU Enterprise Postgres supports many backup and recovery methods, which are available through a GUI or a command-line interface (CLI) (also called command-line user interface (CUI)). FUJITSU WebAdmin provides a one-click backup and recovery feature through its GUI, and `pgx_dmpall` and `pgx_rcvall` are run by using the CLI for more granular scope and automation backup and recovery capabilities.

1.8.1 One-click backup and recovery by using WebAdmin

FUJITSU Enterprise Postgres provides intuitive backup and recovery solutions through the WebAdmin software, which comes packaged with FUJITSU Enterprise Postgres and provides one click back-up and recovery options.

FUJITSU Enterprise Postgres provides three recovery features:

- Media recovery.
- Point-in-time-recovery (PITR).
- Backup and recovery through external copy technologies that are integrated with FUJITSU Enterprise Postgres through user exits.

Recovery by using WebAdmin requires less time and effort because WebAdmin automatically determines the recovery operation scope.

Figure 1-19 shows, in the event of either of the disk failing, that the data always can be recovered to the most recent PiT just before the disk failure. This recovery is achieved by recovering from the most consistent backup that is available and then replaying the WAL logs (or update logs) that are mirrored or duplicated between the primary data storage destination and the backup data storage destination. The data can be recovered to a consistent state for resuming business as usual regardless of disk failure. FUJITSU Enterprise Postgres provides flexibility to DBAs to run the recovery by using the WebAdmin software or through server commands on the CLI.

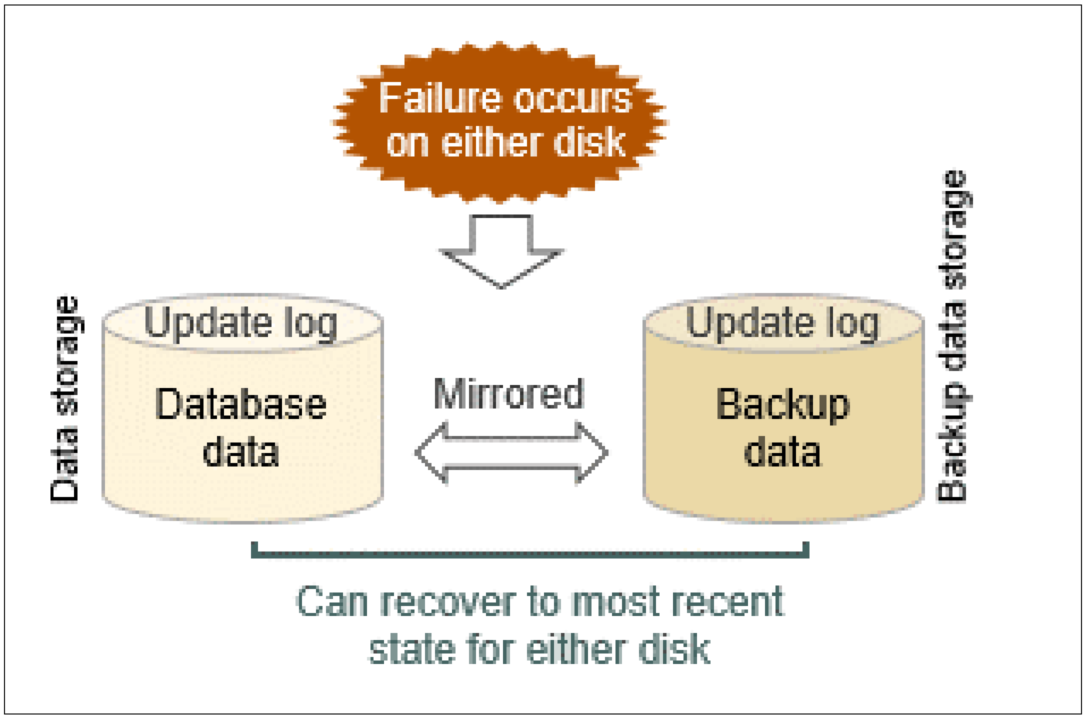


Figure 1-19 Backup and recovery by using WAL mirroring and replay

1.8.2 Backup and recovery by using IBM Spectrum Protect

IBM Spectrum® Protect is a centralized, automated data protection platform that helps to reduce data loss and manage compliance with data retention and availability requirements. It provides scalable data protection for physical file servers, applications, and virtual environments.

IBM Spectrum Protect is a complete solution. It enables advanced data protection for current and next-generation environments, including cloud, virtualized and software-defined environments; core applications; and remote facilities. In addition, applications that can write to S3 targets can also use the automated and flexible policy management of IBM Spectrum Protect. Data that is managed by IBM Spectrum Protect is easily replicated to off-site recovery facilities for safekeeping.

FUJITSU Enterprise Postgres integration with IBM Spectrum Protect allows administrators to manage database backups by using a centralized data protection platform. The additional S3 API support for S3 client application facilitates the use of existing backup tools and enables fast and reliable integration between FUJITSU Enterprise Postgres and IBM Spectrum Protect.

IBM LinuxONE: A resilient platform for secure data serving

IBM LinuxONE is designed to deliver 99.999% availability and above, with built-in redundancy for all critical internal components, and is the only commercially available platform with an estimated mean time between failures (MBTF) of nearly 50 years.

- Hardware designed for no single points of failure.
- Redundant Array of Independent Memory (RAIM) memory, core sparing, no single point of failure, and detects and recovers from memory failures.
- 99.99% availability per frame, and it can be designed for to 5 - 7 nines availability.
- Can be designed for < 60 minute and zero data loss for DR.
- Unique Capacity BackUp (CBU) uses On/Off CoD to provide off-site DR with drastically reduced DR software licensing requirements.

IBM LinuxONE can be deployed with IBM Geographically Dispersed Parallel Sysplex® (IBM GDPS®) to deliver near continuous availability for mission-critical workloads, enabling clients to achieve an RPO of zero and an RTO of zero to less than an hour. For more information or assistance with implementing GDPS, contact your IBM customer representative.

1.9 Hybrid Transactional Analytical Processing: Merging Online Transaction Processing and Online Analytical Processing

Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) are the two traditional workload characterizations that are associated with most data layer architectures across almost all organizations that use relational database management software. Although OLTP provides a snapshot of the live business processes, OLAP shows a multi-dimensional current and historical perspective of the business process lifecycle. Because OLTP provides an instant snapshot of the live business processes, OLTP systems are governed by design principles that favor high-speed and high volume transaction processing.

From hardware to logical database design, the goal is to complete as many successful transactions as possible. The system should be capable of processing as many transactions as possible with the condition that each transaction should be fast enough to commit the data without consuming too many computing resources and provide the data for further processing without latency.

OLAP systems are the backbone of business analytics and data mining with complex aggregations run on the consolidated data that is received from the OLTP systems. The Extract, Transform, and Load (ETL) layer connects the OLTP and OLAP systems. With the tremendous growth of data over the last two decades, the ETL pipelines have become overly complex, which increases the time gap between data ingestion and drawing business insights from the consolidated data in OLAP systems.

Almost all modern applications that rely on real-time analytics have challenged database management product designers to design RDBMSs so that businesses can draw real-time insights from changing data rather than waiting for the ETL to complete, which usually occurs with overnight batch processing. To characterize such a requirement, Gartner coined the term *HTAP* at the early part of this decade. HTAP is the newest workload characterization after OLTP and OLAP.

Figure 1-20 shows Vertical Clustered Index (VCI), which is the Fujitsu analogue to the HTAP capability. VCI is built into FUJITSU Enterprise Postgres.

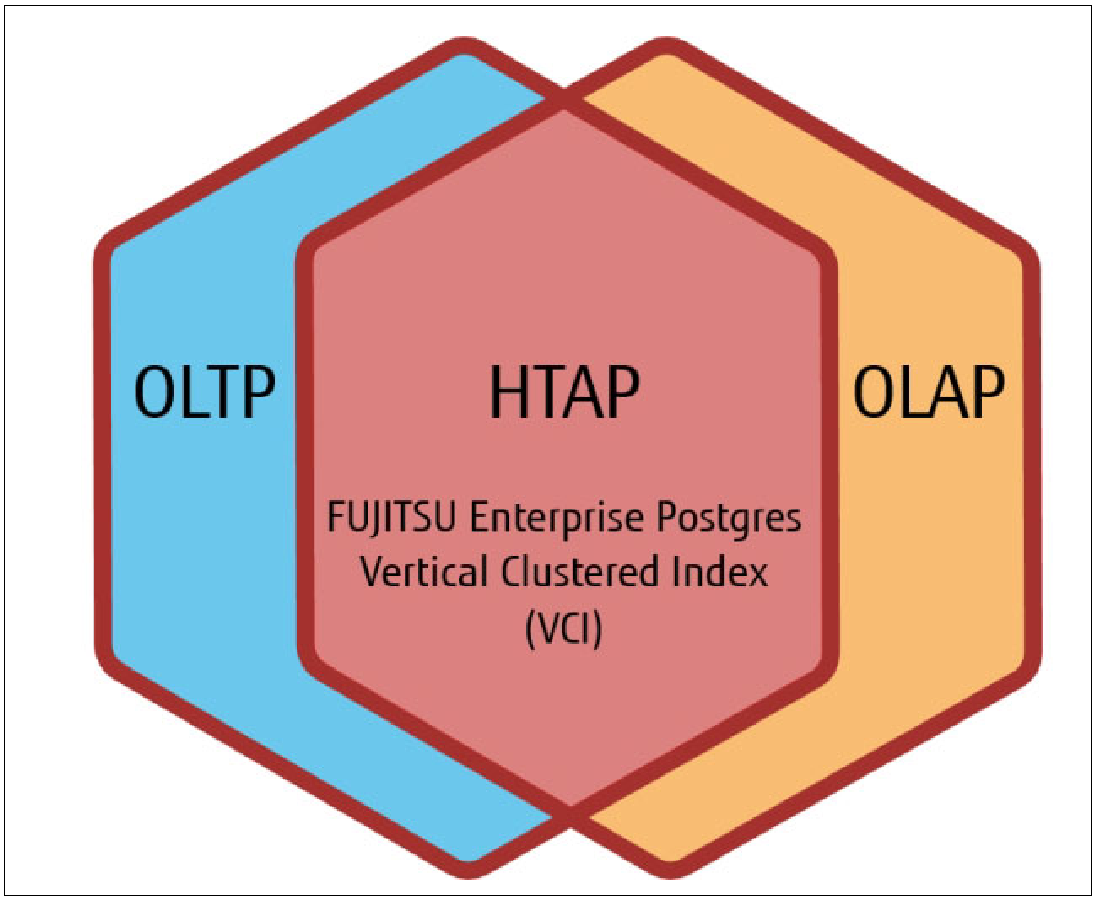


Figure 1-20 HTAP and FUJITSU Enterprise Postgres Vertical Clustered Index

Vertical Clustered Index

There are various implementations of HTAP that are available, which are either in the form of high-speed, in-memory processing database features, columnar storage, or in-memory and NoSQL features combined as an offering. Although many of the HTAP solutions are either not ACID-compliant or require a new hardware appliance, FUJITSU VCI is both ACID-compliant and does not depend on any special type of hardware appliance. The design thinking behind VCI is to enable the businesses to perform real-time analytics on changing data sets without the need to perform ETL on data from OLTP to OLAP systems and any additional cost.

Unlike many other HTAP features in other RDBMSs, VCI comes packaged with FUJITSU Enterprise Postgres without any additional cost. VCI is the Fujitsu implementation of in-memory columnar storage (IMCS) that is designed to improve the performance of analytical queries running on existing OLTP systems instead of OLAP systems to enable real-time analytics on transient data.

Figure 1-21 on page 31 shows how the VCI technology synchronizes the conversion between row format table data into VCI columnar storage by using write-optimized and read-optimized in-memory storage conversion technology to maintain the integrity of the business transactions.

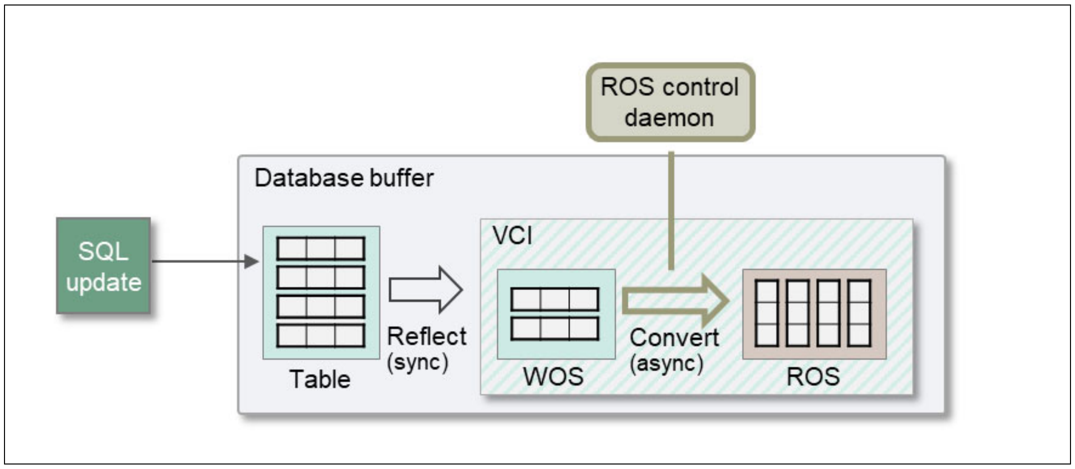


Figure 1-21 Vertical Clustered Index

Furthermore, VCI helps decrease the dependency on ETL while improving business efficiency with improved TCO through faster business reporting. With VCI enabled on candidate columns, many of the time-consuming queries can be sped up while also offloading the OLAP processing to VCI indexes. VCI is crash-safe because the columnar data is always stored on the disk, and in the event of a server restart, the columnar data can be cached immediately to ensure stable performance.

1.10 Faster data consumption and memory optimization

This section describes how to achieve faster data loading and reduce memory usage when using FUJITSU High-Speed Data Load and Global Meta Cache (GMC).

1.10.1 Faster data load when using FUJITSU High-Speed Data Load

To fully use the latest advancements in computing technology, it is critical that the data load utility is not limited by the database technology. The FUJITSU High-Speed Data Load utility is an answer to optimally using multi-core technology for bulk loading data in to FUJITSU Enterprise Postgres. It is a performance enhancement feature that extends the OSS PostgreSQL base COPY function by adding parallelism to data load processes.

Figure 1-22 shows that the FUJITSU High-Speed Data Load utility achieves performance by spreading the data from an external file into many parallel workers to simultaneous run data conversion, table creation, and index creation, and loading the data, which all reduces the load time.

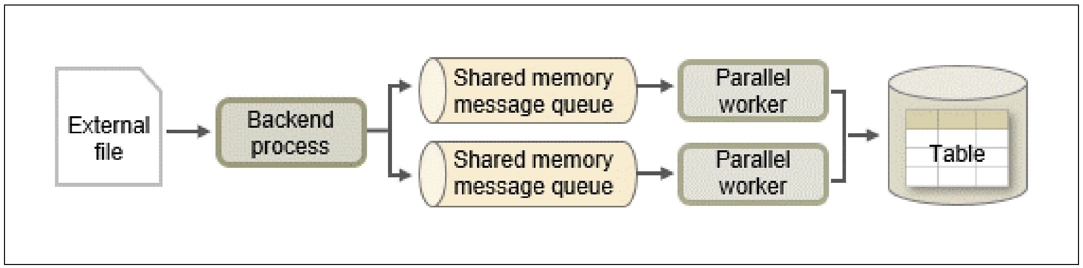


Figure 1-22 FUJITSU Enterprise Postgres High-Speed Data Load utility for faster data loading

A degree of parallelism is maintained and controlled by the FUJITSU High-Speed Data Load utility dynamically at run time. Several benchmarks were jointly conducted by Fujitsu and IBM on the latest IBM LinuxONE servers, which have shown tremendous performance differences between the OSS PostgreSQL data load utilities and FUJITSU High-Speed Data Load utility. For more information. see 2.3.2, “FUJITSU High-Speed Data Load: pgx_loader” on page 64.

1.10.2 Reducing memory usage with FUJITSU Global Meta Cache

GMC is the Fujitsu enhancement to OSS PostgreSQL that reduces memory usage by deploying a meta cache in the GMC area of shared memory and deploying only the process management information to each process. Figure 1-23 shows the comparison of meta cache operations with and without the FUJITSU GMC feature.

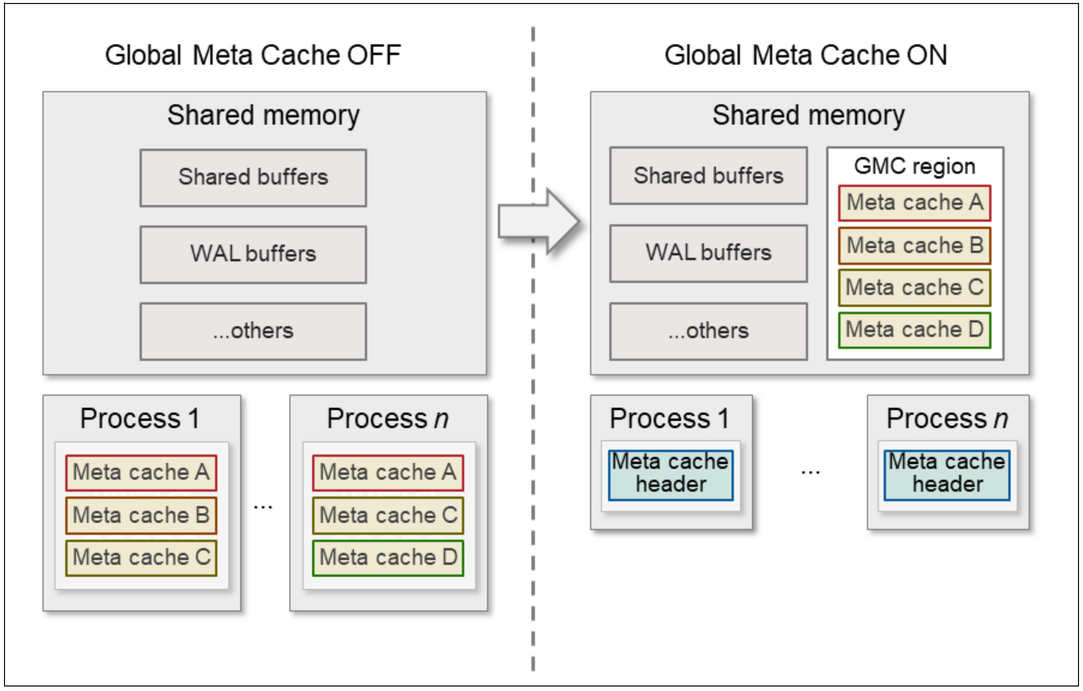


Figure 1-23 FUJITSU Enterprise Postgres Global Meta Cache and OSS Cache management

All RDBMSs have system catalog tables that store schema metadata about all the objects that are in the database. When an SQL query runs, the processing of the SQL goes through several steps from query parsing to creation of the query plan and its execution to fetch the data from the tables. For query processing, PostgreSQL depends on the metadata of the user-defined tables and indexes that is maintained in the system catalog tables. During this processing, PostgreSQL caches this metadata into the memory, so this cached metadata is called *meta cache*.

PostgreSQL maintains this meta cache per process or per connection in the shared memory area of the installed system RAM. As the number of connections increases, the meta cache usage also increases. Moreover, the meta cache usage also increases in proportion to the number of user-defined tables and other database objects being accessed. For OLTP or Mixed Workload applications, where the number of connections often become overwhelming, the total meta cache size might outgrow and exceed the total installed memory, which adversely impacts the memory allocations to the database cache or the OS file cache.

FUJITSU GMC alleviates system memory usage by caching the metadata per process in the GMC area instead of caching in the per process memory area, thus sharing the meta cache on the shared memory. For more information about FUJITSU Enterprise Postgres GMC, see 2.3.3, “Global Meta Cache” on page 68.

1.10.3 Comparing FUJITSU Enterprise Postgres 12 with open source PostgresSQL on IBM LinuxONE

Table 1-2 shows the feature comparison of OSS PostgreSQL to FUJITSU Enterprise Postgres on IBM LinuxONE.

Table 1-2 Comparing FUJITSU Enterprise Postgres 12 to OSS PostgreSQL 12

OSS PostgreSQL 12	FUJITSU Enterprise Postgres on IBM LinuxONE	Feature	Feature description
✗	✓	Integration with IBM LinuxONE CryptoCard HSM	The CryptoCard is an HSM that protects your digital keys by storing them in separate hardware. The HSM is FIPS 140-2 Level 4 certified.
✗	✓	TDE (256-bit PCI DSS-compliant)	256-bit encryption is one of the most secure encryption methods available because it uses a 256-bit key to encrypt/decrypt data.
✗	✓	Data masking	Policy-based data masking allows you to retain the structure of the data when sharing a database with sensitive customer information beyond the permitted production environment, such as for development and testing.
✗	✓	Database transaction log mirroring	Mirrors transaction logs at different storage locations to improve recovery reliability in disk failure scenarios.
✓	✓	IBM zEnterprise® Data Compression (IBM zEDC) based data compression	Fujitsu backup and recovery commands are equipped to maximize compression and recompression by using IBM zEDC. Reduces storage occupancy and increases efficiency by using the IBM zEDC compression acceleration capability by using hardware-based compression and decompression for backup and recovery operations.
✗	✓	MC	A unique feature of FUJITSU Enterprise Postgres that continually monitors your system and seamlessly switches OLTP to an alternative server to ensure business continuity when an abnormality is detected.
✗	✓	Server Assistant	A unique feature of FUJITSU Enterprise Postgres that continually monitors your system and provides quorum service between the primary and standby database servers and seamlessly switches OLTP to an alternative server to ensure business continuity when an abnormality is detected.

✗	✓	Database duplexing	Two copies of a single database on different server instances, often in different physical locations to ensure HA.
✗	✓	Dedicated audit log	A unique feature of FUJITSU Enterprise Postgres to deliver in the key areas of data accountability, traceability, and the ability to audit. It is also compliant with Payment Card Industry Data Security Standard (PCI DSS).
✗	✓	Connection Manager	A unique feature of FUJITSU Enterprise Postgres that allows application access to continue without being aware of the connection destination of the applications.
✗	✓	GMC	The GMC feature enables sharing of meta caches on shared memory, thus reducing overall system memory usage.
✗	✓	Easy setup (installer)	PostgreSQL has offered an installer system since Version 8.0 to make the process easier and faster.
✗	✓	WebAdmin	WebAdmin in FUJITSU Enterprise Postgres helps you easily manage your database and its contents. An Enhanced GUI management tool makes the setup and cluster management simpler. Save time and cost by setting up the database quickly and easily.
✗	✓	High-speed backup/recovery	GUI-based, one-click backup and recovery improves efficiency and mean time to recovery. CLI-based, one command for all backup and recovery operations.
✗	✓	FUJITSU High-Speed Data Load	Data loading is the process of copying and loading data or data sets from a source file in a database. The faster the better.
✗	✓	In-memory columnar index (VCI)	HTAP capability by using VCI. VCI is the Fujitsu proprietary implementation of in-memory columnar index that uses a parallel processing engine to instantly update column-oriented data in response to changes in row-oriented data, and processes column-oriented data quickly.
✓	✓	Parallel query	This Fujitsu enhancement allows systems to allocate more workers during query execution (if there are workers available at the time). This improves query performance, which might otherwise starve CPU if there were fewer or no workers when the query started.
✗	✓	Support for COBOL applications	COBOL is a business-based programming language that is used by many organizations around the world.
✗	✓	Support for Embedded SQL National Character Data in COBOL	National character strings are synonyms for character strings or graphic strings that can be a sequence of bytes that represent character data in UTF-8 or UTF-16BE encoding in a Unicode database.

☒	☑	Support for variable format source code in COBOL programs	Source code is the set of instructions and statements that are written by a programmer to be converted into machine language by a compiler. The converted code is the object code. FUJITSU Enterprise Postgres supports a variable version of source code in COBOL applications.
☒	☑	Smart setup (Create Master/Standby/Standalone)	Use the smart setup features to set up your database quickly and efficiently to save time and money, and ensure a correct setup from the start by using WebAdmin.

1.11 FUJITSU Enterprise Postgres editions for IBM LinuxONE

FUJITSU Enterprise Postgres on IBM LinuxONE is available in only one version that is called FUJITSU Enterprise Postgres 12 Advanced Edition. FUJITSU Enterprise Postgres 12 SP1 Advanced Edition is the latest version that is available on IBM LinuxONE and IBM Z servers. For this publication, we used FUJITSU Enterprise Postgres 12 Advanced Edition.

FUJITSU Enterprise Postgres 12 SP1 Advanced Edition on IBM LinuxONE is based on OSS PostgreSQL 12.1.

FUJITSU Enterprise Postgres certified operating system on IBM LinuxONE

One of the OSs shown below is required to use FUJITSU Enterprise Postgres 12 SP1 Advanced Edition on IBM LinuxONE. We suggest you check and use a version that is certified and supported by Red Hat or SUSE for the target IBM Z / IBM LinuxONE hardware.

- Red Hat Enterprise Linux 7.7 or later.
- Red Hat Enterprise Linux 8.1 or later.
- SUSE Linux Enterprise Server 12 SP5 or later for FUJITSU Enterprise Postgres 12 SP1 Advanced Edition.
- SUSE Linux Enterprise Server 12 SP4 or later for FUJITSU Enterprise Postgres 12 Advanced Edition. This edition is used in our lab environment for this publication.

1.12 Licensing

Fujitsu provides two types of subscription licenses for FUJITSU Enterprise Postgres 12 on IBM LinuxONE:

- Trial subscription license
- Annual subscription license

Trial subscription license

Fujitsu and IBM provide a FUJITSU Enterprise Postgres 12 on IBM LinuxONE^{TM} trial version for 90 days. You can download the trail version from FUJITSU Enterprise Postgres trial version.

With the trial version, you also get access to all the enterprise features of FUJITSU Enterprise Postgres 12 for 90 days.

Annual subscription license

Different annual subscription licenses are available depending on how you use FUJITSU Enterprise Postgres 12 on IBM LinuxONE. For production use, you may license the lower of the maximum number of virtual Integrated Facility for Linux (IFL) processors that are available to FUJITSU Enterprise Postgres in the virtual machines (VMs) or to the maximum number of IFL processors in the IBM LinuxONE server that are available to FUJITSU Enterprise Postgres. For a non-production environment, a single license is used up to four IFL processors.

License considerations for high availability architectures

If you are considering implementing a HA architecture, then you require more subscription licenses only for an active-standby setup. For a passive-standby or cold-standby setup, you do not require more subscription licenses.

Licensing model when using a partial number of IFL processors

If you are using a partial number of IFL processors across a number of VMs where FUJITSU Enterprise Postgres is installed, then the number of IFL processors for which the annual subscription is to be licensed is rounded up to the nearest number of logical IFL processors. For more information about subcapacity licensing calculations, contact your IBM or Fujitsu customer service representative.

2

Introducing FUJITSU Enterprise Postgres features on IBM LinuxONE

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

Figure 2-1 provides a view of all the enterprise features covering development, operations, and management of FUJITSU Enterprise Postgres.

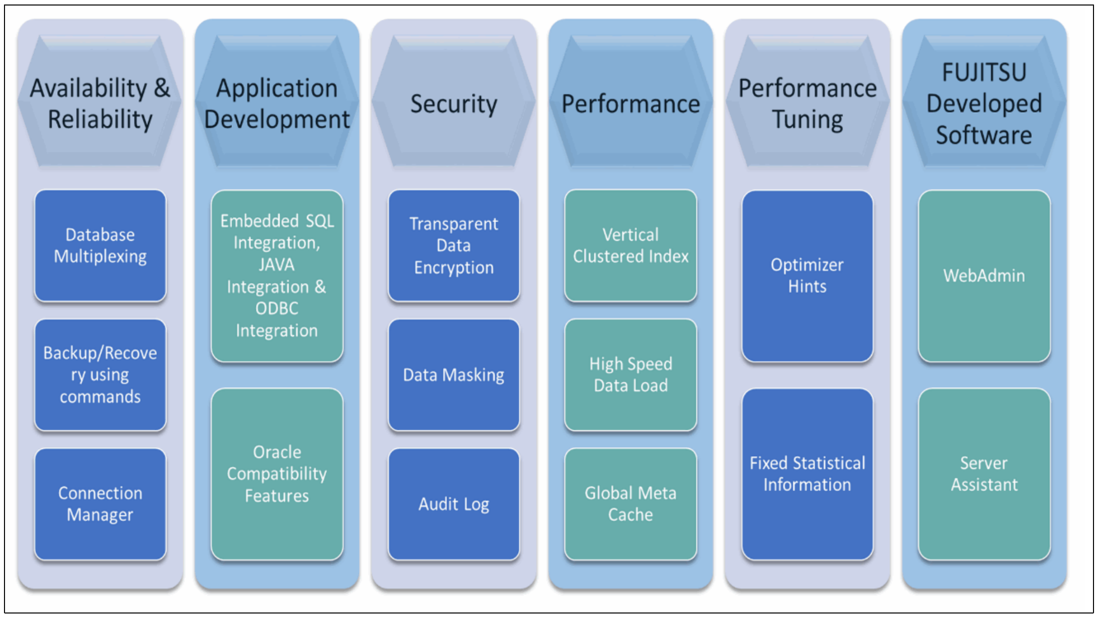

Availability & Reliability
Database Multiplexing
Backup/Recovery using commands
Connection Manager
Application Development
Embedded SQL Integration, JAVA Integration & ODBC Integration
Oracle Compatibility Features
Security
Transparent Data Encryption
Data Masking
Audit Log
Performance
Vertical Clustered Index
High Speed Data Load
Global Meta Cache
Performance Tuning
Optimizer Hints
Fixed Statistical Information
FUJITSU Developed Software
WebAdmin
Server Assistant

Figure 2-1 FUJITSU Enterprise Postgres features

This chapter covers the following topics:

- Availability and reliability features
- Security
- Performance
- Fujitsu developed database management software
- Application development features
- FUJITSU Enterprise Postgres supported open-source software peripheral devices
- FUJITSU Enterprise Postgres data sheet

2.1 Availability and reliability features

There are different types of HA modes that can be achieved with FUJITSU Enterprise Postgres Advanced Edition. As we described in Chapter 1, “Customer value” on page 1, in general, a complete HA solution for databases should be highly resilient and have instant availability through automatic failover for business continuity. There are four major requirements for which a robust and fault-tolerant HA implementation is required:

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

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

2.1.1 Database multiplexing

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

Figure 2-2 shows a high-level architecture with switchover and failover capabilities that uses an MC.

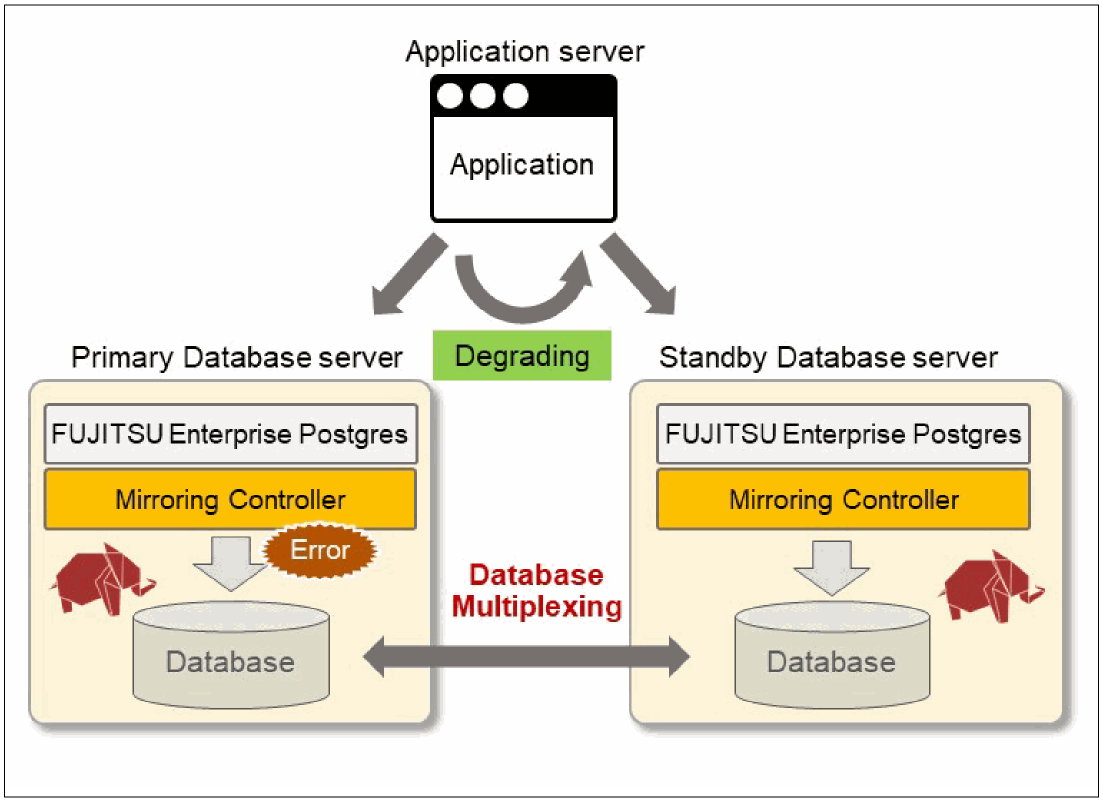


Figure 2-2 FUJITSU Enterprise Postgres Database Multiplexing

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

- Constantly monitor the operating system (OS), server, disk, network, and database, and notify you if something is amiss.
- If the primary server fails, MC performs an automatic switch/disconnect to the standby server to ensure operational continuity, and the primary server is disconnected.
- Detects streaming replication issues (log transfer network and Write-Ahead-Log (WAL) send/receive processes) by periodically accessing the PostgreSQL system views.
- Comes with no additional cost over the annual subscription license.

- Provides HA with minimal disruption (in seconds).
- Packaged as part of FUJITSU Enterprise Postgres Advanced Edition.

Note: For more information, see the “Database Multiplexing Mode” section in the *FUJITSU Enterprise Postgres Cluster Operation Guide*.

Chapter 5, “High availability and high reliability architectures” on page 165, describes the implementation of HA architecture that shows the steps to configure the MC and achieve database multiplexing in a hot-standby mode of operating for FUJITSU Enterprise Postgres Advanced Edition.

The MC is effective in performing failover and controlled switchover when any issues occur in the primary database server. To resolve a split-brain scenario in data multiplexing mode, FUJITSU Enterprise Postgres uses the Server Assistant program, which is described in 2.1.2, “Server Assistant: Avoiding split-brain scenarios” on page 40.

2.1.2 Server Assistant: Avoiding split-brain scenarios

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

Figure 2-3 on page 41 shows a high-level architecture with Server Assistant providing the quorum service (also known as arbitration service) between the primary and standby server, and fencing the primary database server when the heartbeat fails between the primary and the standby database servers.

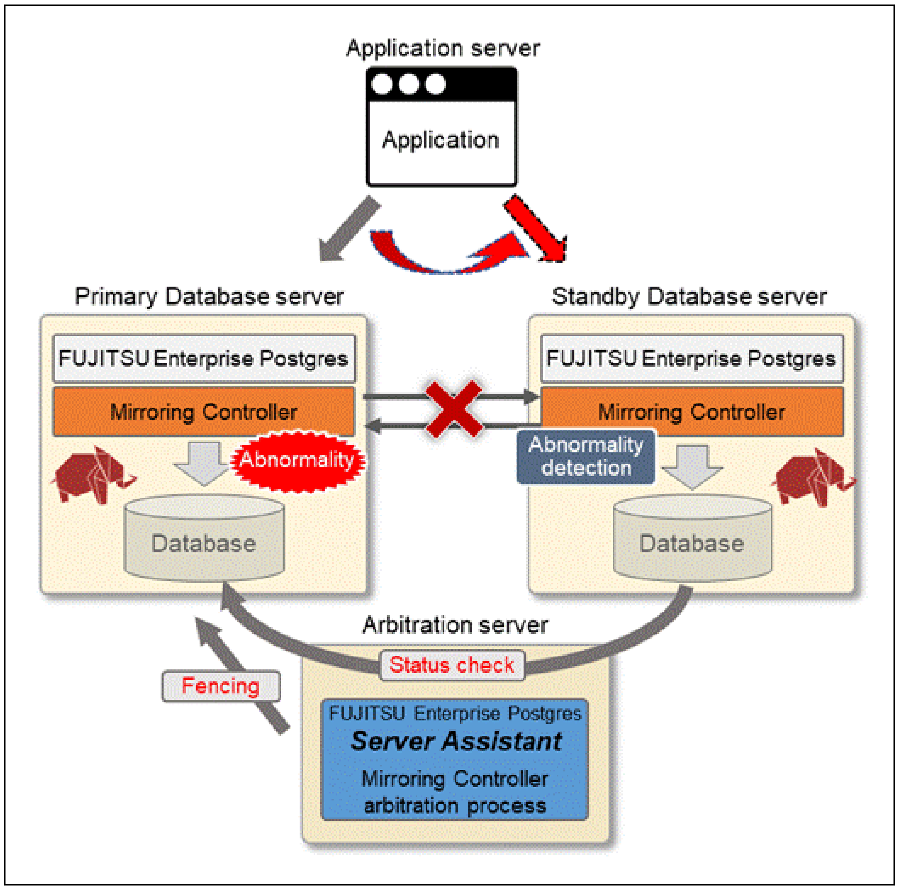


Figure 2-3 FUJITSU Enterprise Postgres Server Assistant arbitration process

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

Note: Although Server Assistant can be installed on the same server as the database server, it is a best practice that Server Assistant is installed on an independent server.

On IBM LinuxONE, it can be installed on a virtual guest (VM) that can be on a logical partition (LPAR) other than the LPARs on which the primary and standby FUJITSU Enterprise Postgres instances are configured.

In our lab environment, Server Assistant is installed and configured on LPAR 3, and the primary and standby database servers are installed on LPAR 1 and LPAR 2. The lab environment that is used in this publication is described in Chapter 6, “Connection pooling and load balancing with Pgpool-II” on page 233. LPAR 3 is on IBM LinuxONE B, and LPAR 1 and LPAR 2 are on IBM LinuxONE A.

In 5.4.6, “Simulating automatic failover” on page 217, we describe automatic failover from the primary database server to the standby database server and the recovery of primary database server.

In Chapter 6, “Connection pooling and load balancing with Pgpool-II” on page 233, we further describe adding Pgpool-II to provide connection pooling and load balancing in the database multiplexing mode of operation of FUJITSU Enterprise Postgres.

2.1.3 Backup and recovery by using commands

Many factors, including hardware failure, cyberattack, natural disasters, and human error can put data at risk, and as a result impact both business opportunities and the reputation of an organization. You must be prepared by implementing a robust backup/recovery plan to ensure that your business is back up and running with minimal revenue and data loss should one of these problems occur.

FUJITSU Enterprise Postgres supports many backup and recovery methods that are available through the GUI and command-line interface (CLI) (also called command-line user interface (CUI)). Although the FUJITSU WebAdmin GUI tool provides a one-click backup and recovery feature, the CLI utilities **`pgx_dmpall`** and **`pgx_rcvall`** provide a more granular and automated backup and recovery capability.

Backup customization flexibility

Whether the task is full and incremental or physical and logical backups, FUJITSU Enterprise Postgres extends the PostgreSQL backup/recovery utilities by allowing automation of these tasks through scripts, where you can use the copy technology of your choice.

Encryption

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

Types of backup methods that are supported

Figure 2-4 on page 43 shows the many utilities that FUJITSU Enterprise Postgres supports.

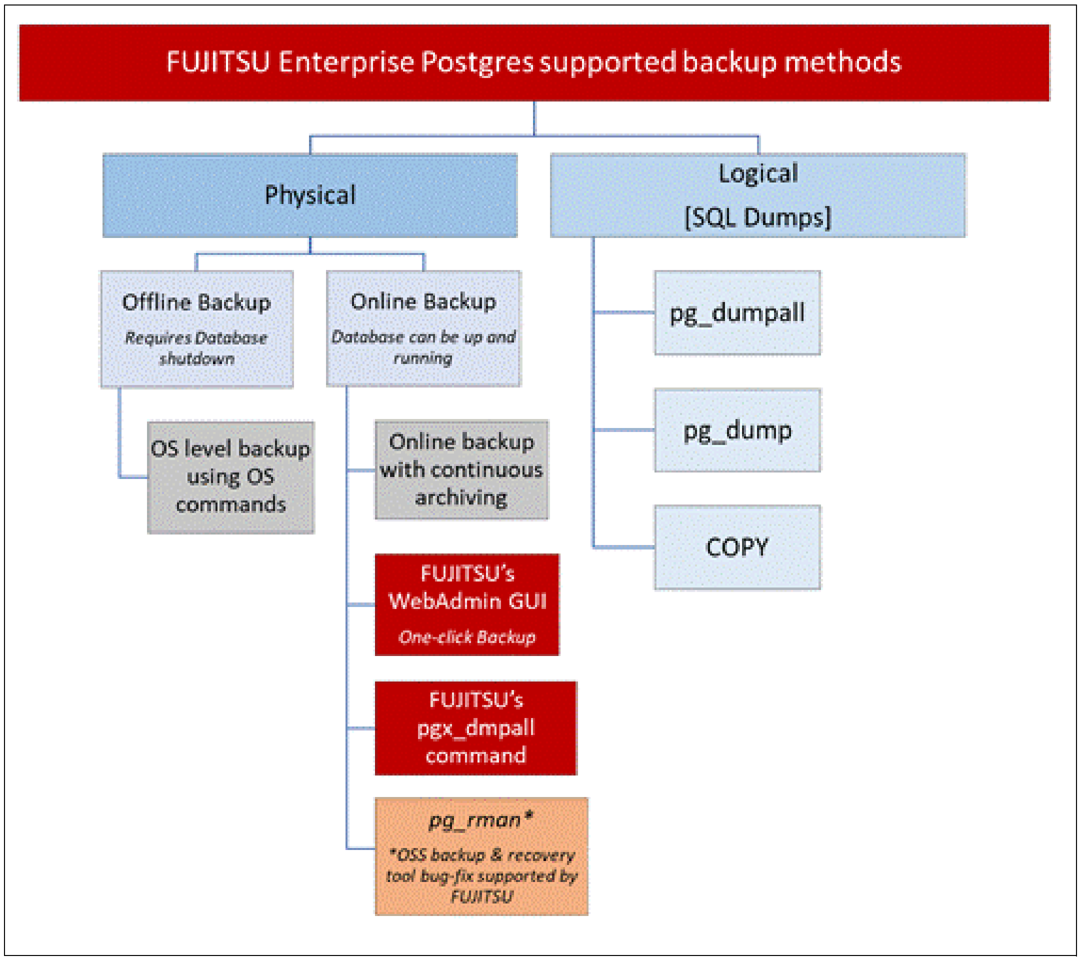

FUJITSU Enterprise Postgres supported backup methods
Physical
Logical [SQL Dumps]
Offline Backup
Requires Database shutdown
Online Backup
Database can be up and running
OS level backup using OS commands
Online backup with continuous archiving
FUJITSU's WebAdmin GUI
One-click Backup
FUJITSU's pgx_dmpall command
pg_rman*
*OSS backup & recovery tool bug-fix supported by FUJITSU
pg_dumpall
pg_dump
COPY

Figure 2-4 FUJITSU Enterprise Postgres supported backup methods

FUJITSU pgx_dmpall backup and pgx_rcvall recovery commands

By using FUJITSU Enterprise Postgres backup and restore commands, you can customize the ensuing behavior by calling customized hook scripts, which simplifies and standardizes your backup and recovery operations across applications in line with your business recovery point objective (RPO).

Figure 2-5 on page 45 shows the phases that the **`pgx_dmpall`** and **`pgx_rcvall`** utilities undergo during a backup and recovery operation:

- **`pgx_dmpall`** phases (also called modes):
  - Prepare mode:
    - Determine the database resources to be backed up.
    - Prepare a backup area.
    - Create the copy command.
  - Backup mode:
    - Performs a backup on the backup area that was determined by the prepare mode.

  - Finalize mode:
    - Writes information relating to the destination of the current backup to the backup information file.
    - Enables the prepare mode to check the destination of the previous backup during the next backup.
- **pgx_rcvall** phase
  - Restore mode:
    - The recovery process starts the copy command in restore mode before WAL is applied by the archive log files.
    - The restore mode recovers the file system by implementing a restore of all target data areas.
  - Metadata files are used for exchanging information between **pgx_dmpall** and **pgx_rcvall**:
    - Backup information file: A file that is used by **pgx*** commands that has the same lifecycle as the backup area where the backup is stored. The contents can be viewed by running the **pgx_rcvall** command.
    - Mode transfer file: A file that is used by the three modes in the **pgx_dmpall** command, and a lifecycle that is a temporary file that is exclusive to the **pgx_dmpall** command.

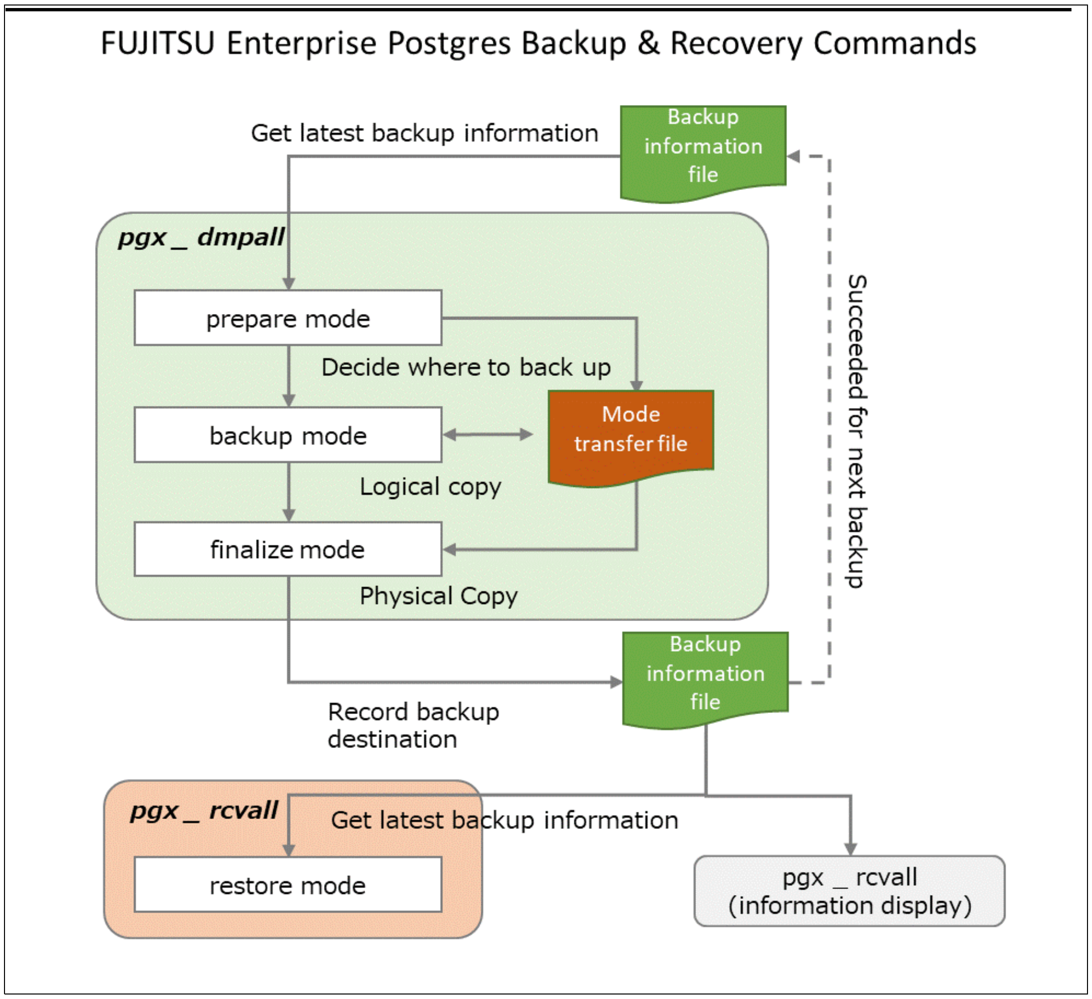

FUJITSU Enterprise Postgres Backup & Recovery Commands
Get latest backup information
Backup information file
pgx _ dmpall
prepare mode
Decide where to back up
backup mode
Mode transfer file
Logical copy
finalize mode
Physical Copy
Succeeded for next backup
Backup information file
Record backup destination
pgx _ rcvall
Get latest backup information
restore mode
pgx _ rcvall (information display)

Figure 2-5 Backup command pgx_dmpall and recovery command pgx_rcvall in action

Backup planning: When to use which backup method

Because there are various methods that are supported for backup for different scenarios, Table 2-1 briefly outlines the use case scenarios.

Table 2-1 Backup planning by considering recovery scenario

Recovery scenario	Backup method
Recover only the table at the time of the backup when data loss of the table occurs due to a customer's mistaken operation.	Because it is a table-based backup, select the logical backup **`pg_dump`** command.
Build a new system on another server, migrate all the data of the old system, and continue the same business.	Select the **`pg_dumpall`** command for logical backup because it is an entire system migration.

Recovery scenario	Backup method
Disk hardware failure occurs, and you want to recover to the latest point just before the failure occurred.	Physical recovery is supported by physical backup. To restore to the latest point, select online backup by continuous archive, the **pgx_dmpall** command, or use WebAdmin.
Recover to the time when the data was logically destroyed, and the database is not working properly.	Recovery of database destruction is supported by physical backup. Because you want to specify the recovery time, select online backup by continuous archive or the **pgx_dmpall** command.

Backup utility selection decision tree

The decision to use a backup utility is governed by the backup target and the RPO. Figure 2-6 provides guidance about using OSS and Fujitsu backup utilities for different backup target and RPO scenarios.

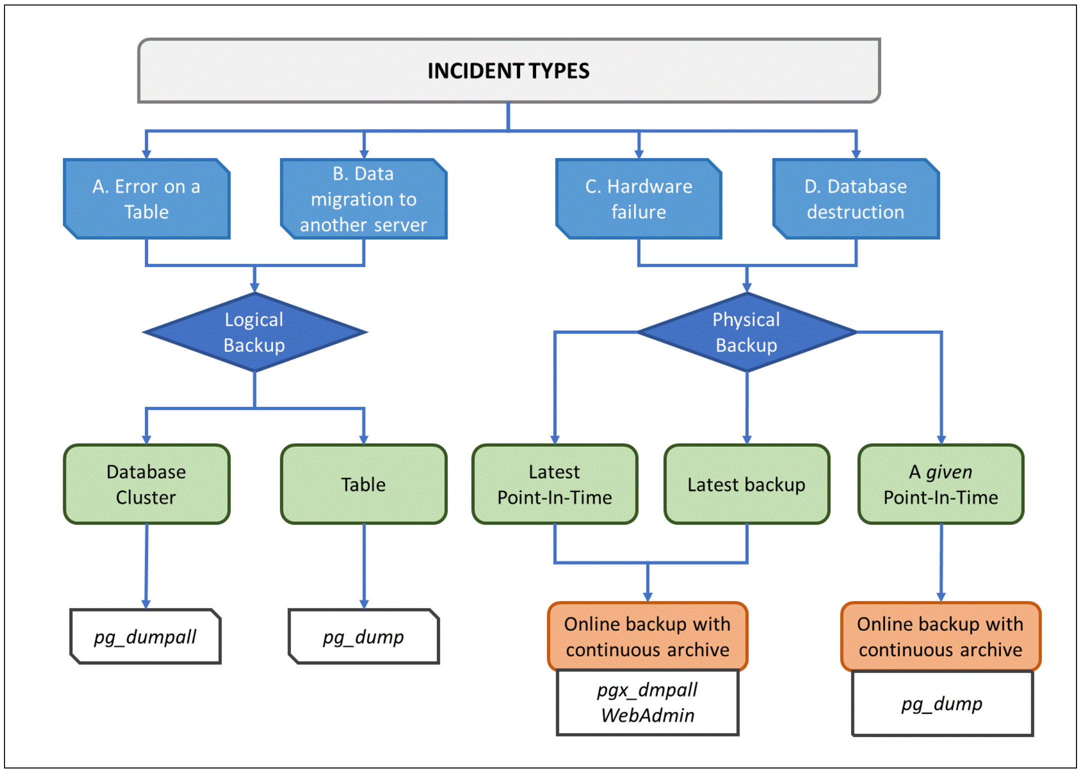


Figure 2-6 Decision tree for selecting backup methods

Note: In Chapter 9, "Backup, recovery, and monitoring" on page 311, we describe the implementation of use cases by using **pgx_dmpall**, **pg_rman** for backups, and point-in-time recovery (PITR) by using **pgx_rcvall** and a restore point. **pg_rman** provides the facility to do incremental backups and comes packaged with FUJITSU Enterprise Postgres.

2.1.4 Connection Manager

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

Figure 2-7 shows the operation of Connection Manager. Connection Manager monitors the application server and FUJITSU Enterprise Postgres Database instances running on the database servers, which are primary and standby database servers.

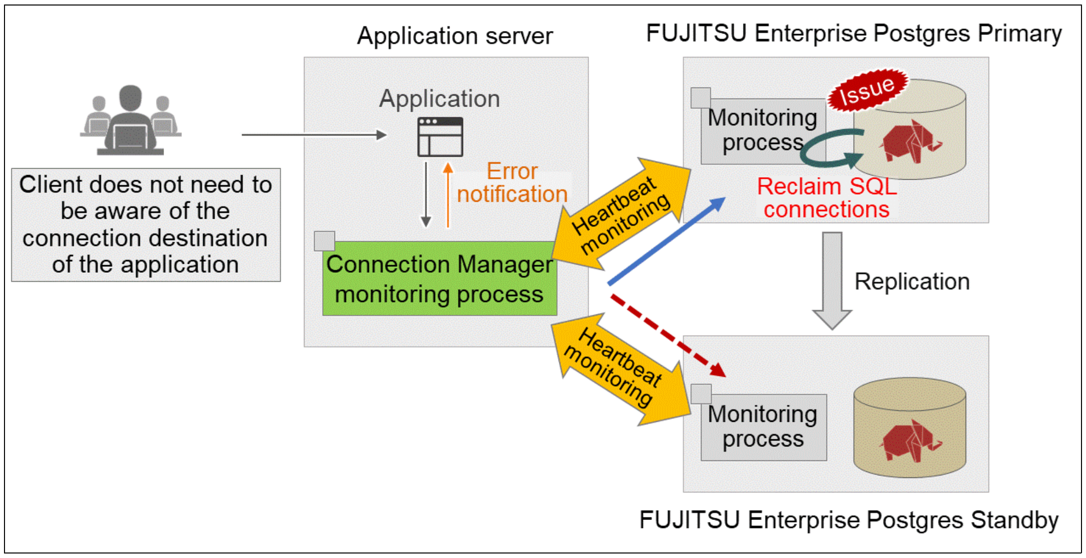

Application server
FUJITSU Enterprise Postgres Primary
Application
Issue
Monitoring process
Client does not need to be aware of the connection destination of the application
Error notification
Heartbeat monitoring
Reclaim SQL connections
Connection Manager monitoring process
Replication
Heartbeat monitoring
Monitoring process
FUJITSU Enterprise Postgres Standby

Figure 2-7 Connection Manager heartbeat monitoring and transparent connection support features

If there is a server failure and the inter-server network link is unavailable, Connection Manager alerts the client and the database instances. The Connection Manager transparent connection feature allows the application to connect to a FUJITSU Enterprise Postgres instance from a set of instances that is configured for replication. The connection to an underlying FUJITSU Enterprise Postgres instance is transparent to the application.

Connection Manager features

Connection Manager provides the following features. These features further improve business application availability.

Heartbeat monitoring feature

Heartbeat monitoring is a keep-alive equivalent timeout function that is implemented at the application layer. This feature detects kernel panics between the server running the application and the server running the FUJITSU Enterprise Postgres instance. It also detects and reports physical server failures and downed inter-server network links between the client and the database instance. The client is notified of an error event through the SQL connection, and the instance is alerted in the form of a force collection of SQL connections with clients that are out of service.

Figure 2-8 shows the internal workings of heartbeat monitoring through the *watchdog and terminator* process.

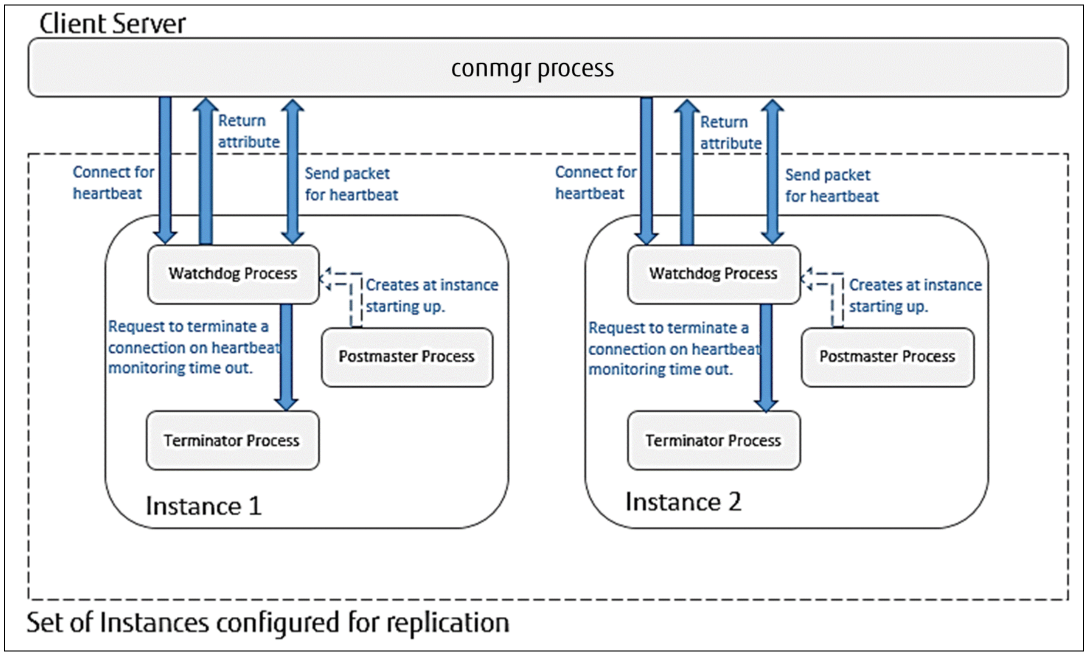

Client Server
conmgr process
Connect for heartbeat
Return attribute
Send packet for heartbeat
Watchdog Process
Creates at instance starting up.
Request to terminate a connection on heartbeat monitoring time out.
Postmaster Process
Terminator Process
Instance 1
Connect for heartbeat
Return attribute
Send packet for heartbeat
Watchdog Process
Creates at instance starting up.
Request to terminate a connection on heartbeat monitoring time out.
Postmaster Process
Terminator Process
Instance 2
Set of Instances configured for replication

Figure 2-8 Heartbeat monitoring process

Transparent connection support feature

As the name implies, the transparent connection support feature makes the connection between the application and the database instances transparent. When an application wants to connect to an instance of an attribute (Primary/Standby) configured with replication, it can connect without knowing which server the instance is running on. Although it might seem to be a single-step process, it involves two steps. In the first step, the application connects to the client driver (libpq), and in the next step, the client driver connects to the Connection Manager process known as the *conmgr* process. Thereafter, the client driver and the database instance send and receive the query execution data, which does not impact the performance of the SQL query execution.

Note: The available client drivers for Connection Manager are libpq (C language library) and ECPG (embedded SQL in C).

In Chapter 7, “Application development, compatibility features, and migration” on page 251, we describe the configuration and operation of Connection Manager.

Note: For more information about Connection Manager operations, see the *Connection Manager User Guide*.

2.2 Security

Data security is an important area of focus for organizations. Organizations must ensure that the confidentiality and safety of the data that is available in electronic and physical form. Organizations with even the most sophisticated technology have faced data breaches that have resulted in expensive consequences of a legal nature. More importantly, damage might be done to their reputation, which has a long-term impact on customer trust and diminished confidence and morale in doing business or using services.

Data security generally follows a layered framework that covers every touch point that is vulnerable to security threats. These layers include human, physical, application, databases, and other detection technologies (see Figure 2-9). The aim of a layered security framework is to make sure that a breach in one layer does not compromise the other layers, and in the worst case, the entire data security.

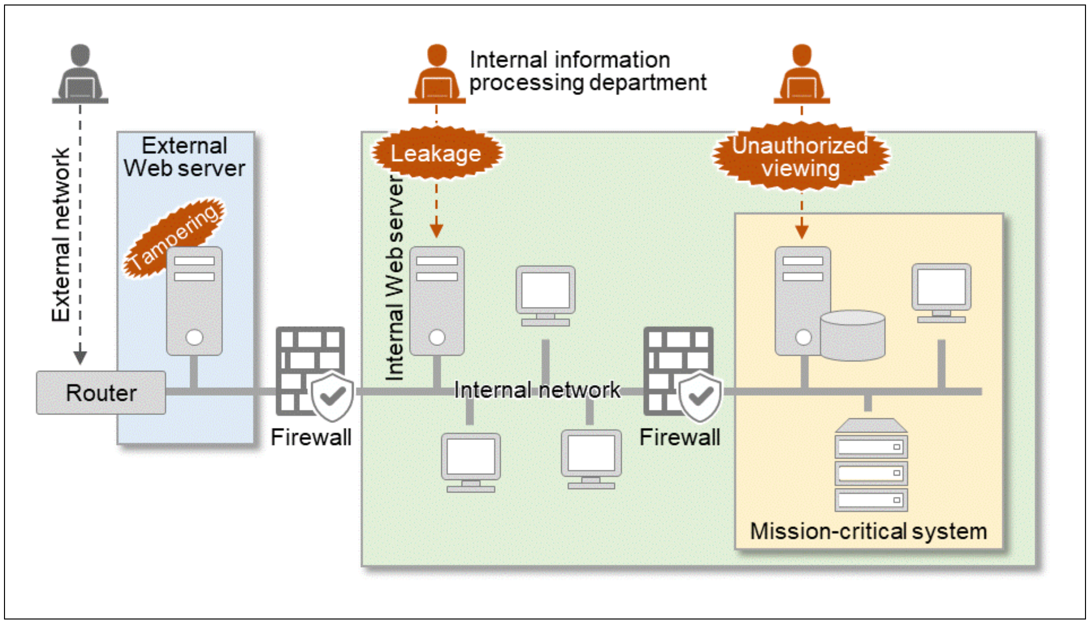


Figure 2-9 Layered security architecture and common vulnerable data security breach points

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

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

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

2.2.1 FUJITSU Transparent Data Encryption

Encryption is regarded as one of the best data protection techniques that are available. Through encryption, the user can scramble the data by using encryption keys, and then use decryption keys to decipher the data. TDE is an encryption method that is transparent to the user and the application, which means no change is required in the application or the existing access policies for using TDE.

FUJITSU Enterprise Postgres comes with TDE integrated. Unlike some other proprietary RDBMS products, you are not required to purchase extra products to gain this function. FUJITSU Enterprise Postgres supports both file-based keystore management and hardware-based keystore management.

Figure 2-10 shows FUJITSU Enterprise Postgres integrated to use CryptoCard based encryption and keystore management.

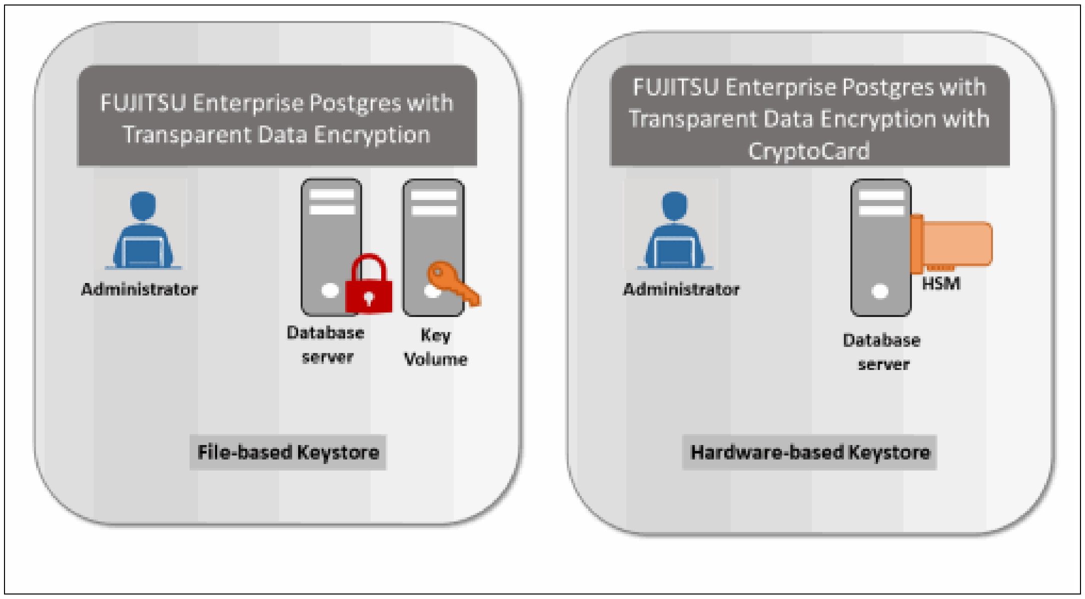

FUJITSU Enterprise Postgres with Transparent Data Encryption
Administrator
Database server
Key Volume
File-based Keystore
FUJITSU Enterprise Postgres with Transparent Data Encryption with CryptoCard
Administrator
HSM
Database server
Hardware-based Keystore

Figure 2-10 Transparent Data Encryption: File-based and hardware security module-based keystore management

TDE key features

Here are the key features of TDE:

- TDE uses the Advanced Encryption Standard (AES) as its encryption algorithm. AES was adopted as a standard in 2002 by the United States Federal Government and is used throughout the world.
- TDE provides encryption at the file level (AES-128 and AES-256), essentially solving the issue of protecting data at rest.
- TDE fulfills requirements for the Payment Card Industry Data Security Standard (PCI DSS), and allows confidential information such as credit card numbers to be made unrecognizable on disk.
- Data is automatically encrypted and decrypted when it is written and read, so manual key management is not required. Even if an attacker gets through all the access controls and connects to the server, they cannot access the data because the OS file is encrypted.
- Encryption is extended to WALs (also known as transaction logs), backups, temporary tables, and temporary indexes, thus providing comprehensive security.

- The encryption algorithm does not alter the size of the object that is being encrypted, so there is no storage overhead.
- Flexibility to encrypt subset of the data per an organization's data classification rules and policies.
- Both file-based and hardware-based keystore management mechanisms are supported.
- Flexibility of using multiple Data Encryption Keys (DEKs), which are encrypted by the Master Encryption Key (MEK).
- On IBM LinuxONE, MEK is further secured with an IBM LinuxONE CryptoCard hardware security module (HSM) managed secure key.
- Take advantage of the CPACF in the IBM Z processor to minimize encryption and decryption overhead so that even in situations where previously the minimum encryption target was selected as a tradeoff between performance and security, it is now possible to encrypt all the data of an application.

Figure 2-11 shows a high-level implementation of FUJITSU TDE with openCryptoki for storing master keys on the CryptoCard (also called HSM). openCryptoki allows FUJITSU Enterprise Postgres to transparently use Common Cryptographic Architecture (CCA) and EP11 (Enterprise PKCS #11) firmware in a CryptoCard.

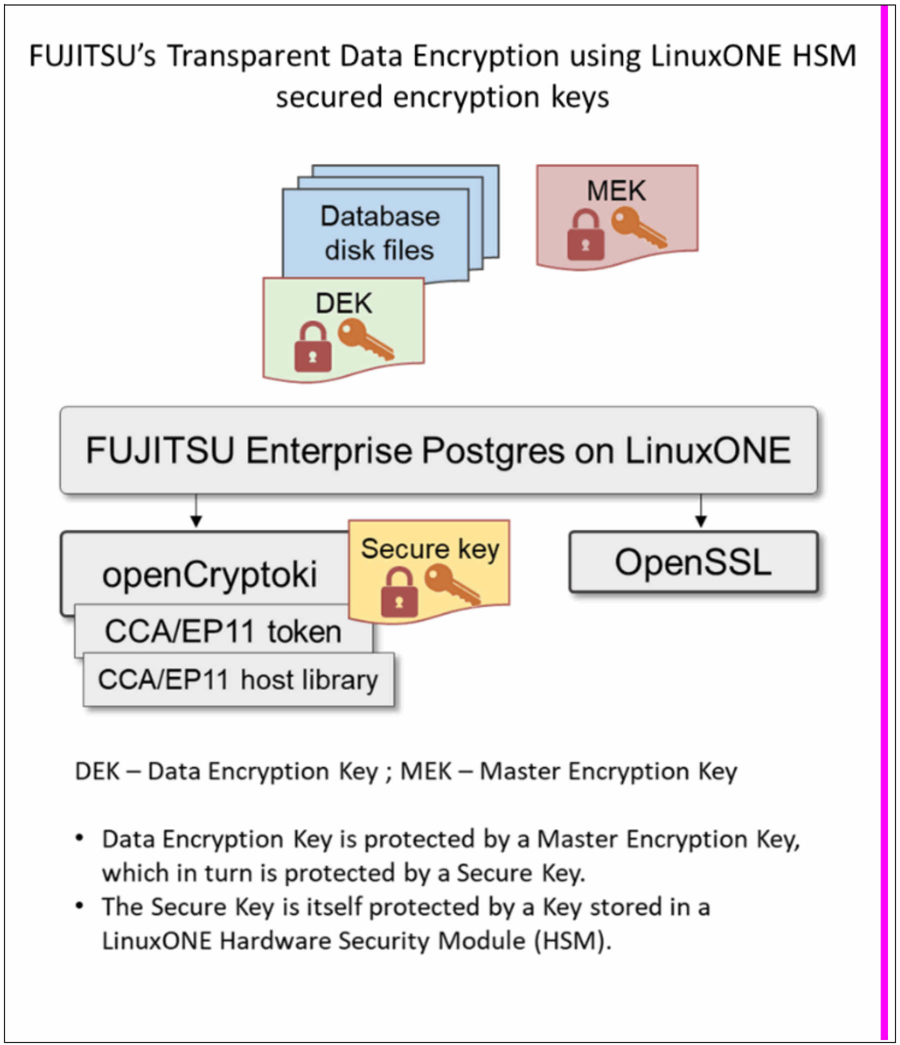

FUJITSU's Transparent Data Encryption using LinuxONE HSM secured encryption keys
Database disk files
MEK
DEK
FUJITSU Enterprise Postgres on LinuxONE
openCryptoki
Secure key
OpenSSL
CCA/EP11 token
CCA/EP11 host library
DEK – Data Encryption Key ; MEK – Master Encryption Key
• Data Encryption Key is protected by a Master Encryption Key, which in turn is protected by a Secure Key.
• The Secure Key is itself protected by a Key stored in a LinuxONE Hardware Security Module (HSM).

Figure 2-11 Transparent Data Encryption secured by encryption keys

In 4.4, “FUJITSU Enterprise Postgres encryption on IBM LinuxONE with crypto cards” on page 126, we describe the use cases of implementing TDE on IBM LinuxONE by using file-based and CryptoCard-based encryption and keystore management techniques.

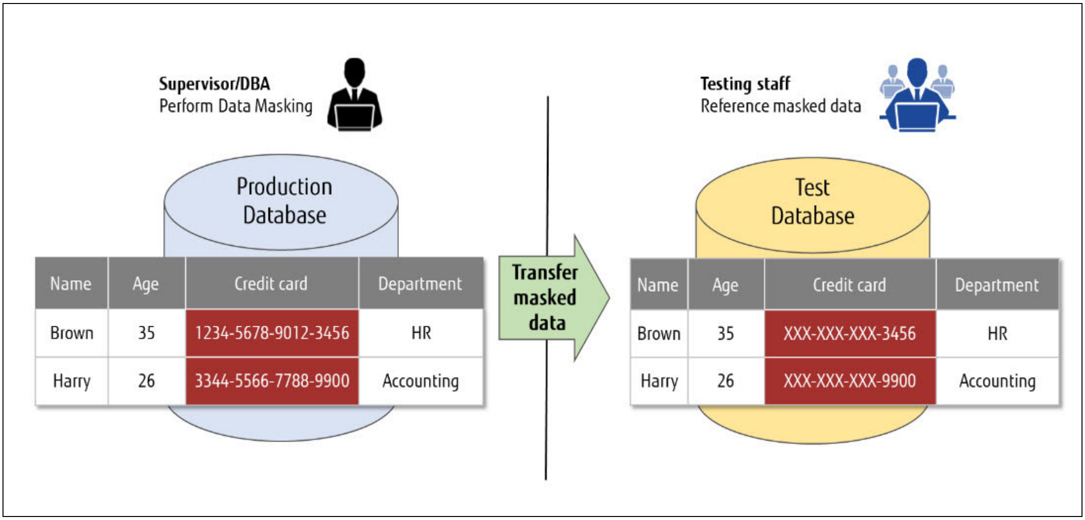
2.2.2 Data masking

Data masking enables data security governance by obfuscating specific columns or part of the columns of tables that store sensitive data while still maintaining the usability of the data.

Here are some common use cases for partial or full obfuscation of information:

- Test data management

  This is the process of sanitizing production data for use in testing of information systems with realistic data without exposing sensitive information to the application development or infrastructure management teams, who might not have the appropriate authority and clearance to access such information. Figure 2-12 shows the data masking use case for test data management.



Name	Age	Credit card	Department
Brown	35	1234-5678-9012-3456	HR
Harry	26	3344-5566-7788-9900	Accounting

Name	Age	Credit card	Department
Brown	35	XXX-XXX-XXX-3456	HR
Harry	26	XXX-XXX-XXX-9900	Accounting

Figure 2-12 Data privacy management by transferring masked production data for testing

- Compliance

  Various organizations collect and store large amounts of complex data as part of business operations and use analytics to run their business, which creates data privacy challenges for the organizations to adhere to different data privacy regulations and compliance based on regions, such as General Data Protection Regulation (GDPR), California Consumer Privacy Act (CCPA), PCI-DSS, and HIPAA (Health Insurance Portability and Accountability Act). Data masking is one of the most effective ways to be compliant to various regional and international data privacy legislation and compliance.

- Production data security

  While data in production is accessed by people with different roles such as system administrators, customer support, production application maintenance personnel, and business process analysts, it is important that data masking is role-sensitive. FUJITSU Data Masking provides the facility to implement role-sensitive masking. Figure 2-13 on page 53 shows a production environment example.

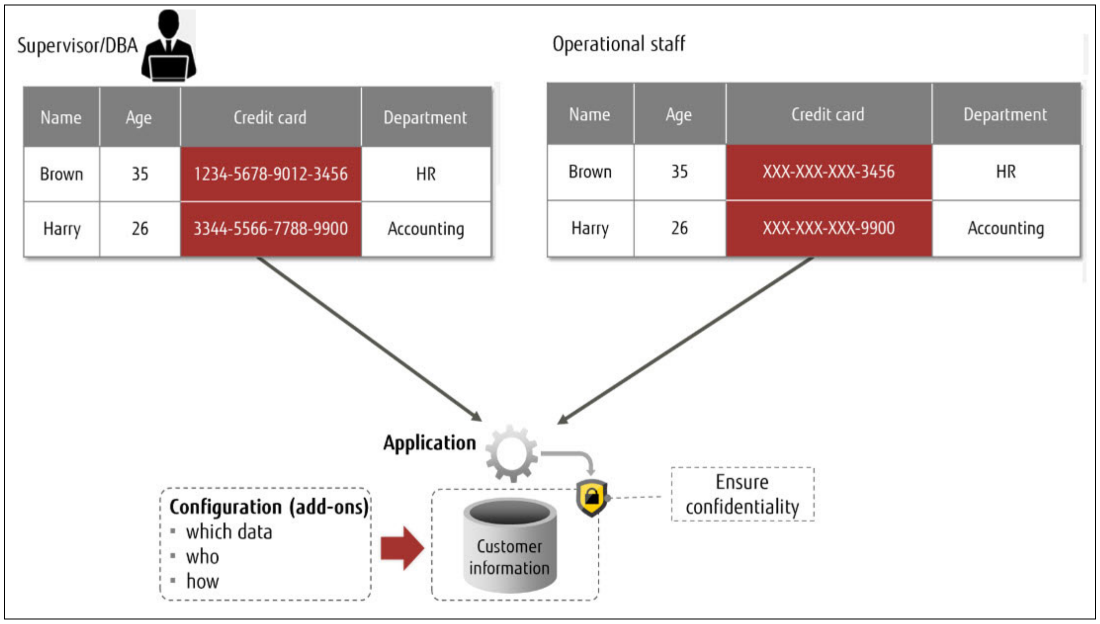


Figure 2-13 Production environment data masking example

Features of FUJITSU Data Masking technology

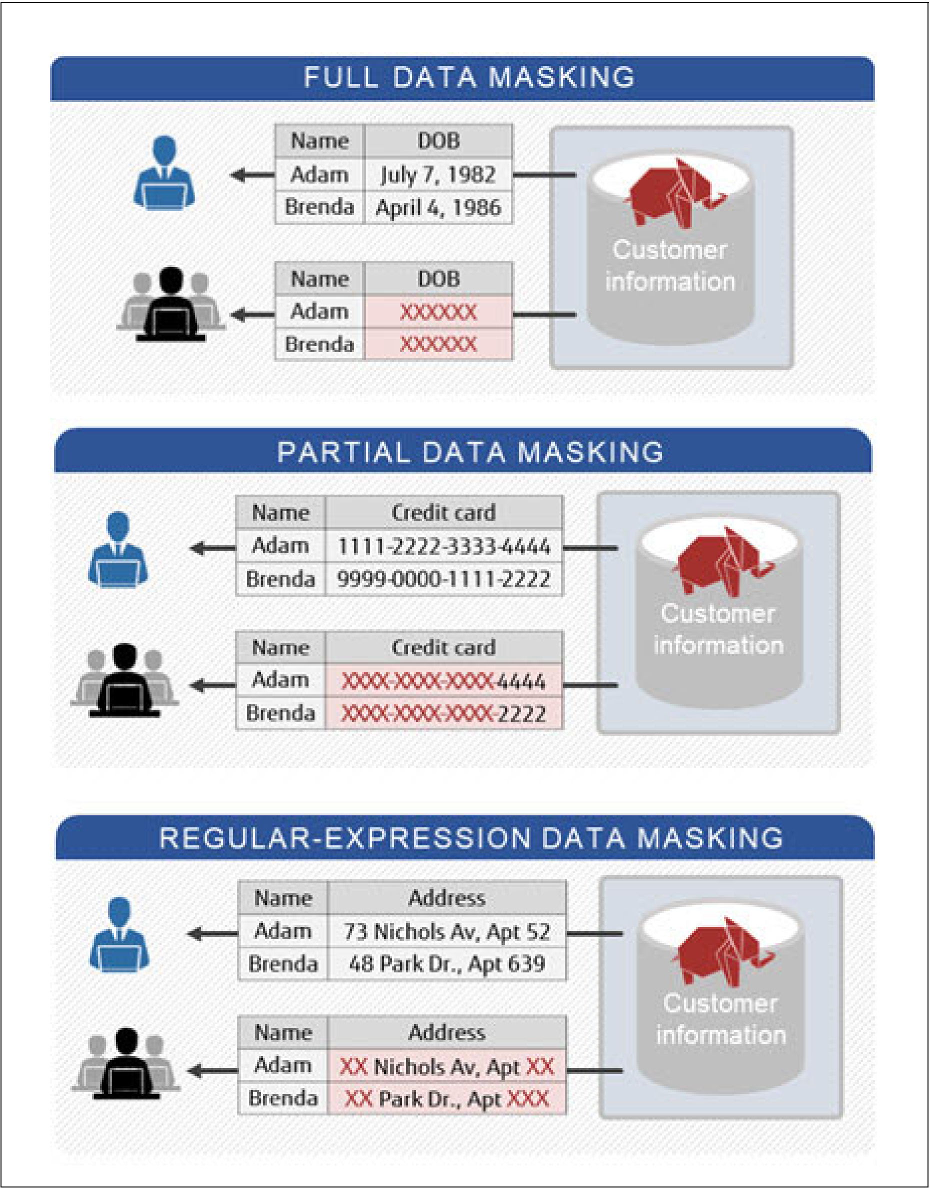
- FUJITSU Enterprise Postgres uses a flexible and easy to use policy-based data masking technique. There are advantages of using Fujitsu policy-based Data Masking over other the techniques.
- Policy-based data masking allows the development of data-sensitive masking policies for different data classifications.
- Data masking policies can be applied to tables for different columns that come under one data classification or another.
- After the policy-based data masking is applied, the policies can be disabled or enabled as required without needing to remove or reapply them.
- The FUJITSU Data Masking feature is irreversible because the relationship between the original data and the masked data is severed during the masking policy implementation to de-risk any reverse engineering to re-create the production data.
- The data masking policy is applied at the time that the data is accessed, so queried data is modified according to the masking policy before the results are returned in the query chain.
- Data masking policies can be configured to be applied to specific roles / conditions so that the original data cannot be viewed without the appropriate privileges.
- Referential integrity is maintained in the source database because the data masking policy is applied during the data access process.
- Masking policies generate data that can be same every time masked data is accessed, which maintains test data consistency across multiple iterations of accessing original data.
- Availability of catalog tables for querying data masking policy information to assess the current sensitive data policy state of a database.

Types of data masking

There are three different types of data masking that are available:

- Full masking: A whole column value can be obfuscated with alternative values.
- Partial masking: Part of a column value can be obfuscated with alternative values.
- Regular expression masking: The value of a column can be obfuscated through a regular expression statement.

Figure 2-14 shows the three different types of data masking techniques that are offered with FUJITSU Enterprise Postgres Advanced Edition.



Name	DOB
Adam	July 7, 1982
Brenda	April 4, 1986

Name	DOB
Adam	XXXXXX
Brenda	XXXXXX

Name	Credit card
Adam	1111-2222-3333-4444
Brenda	9999-0000-1111-2222

Name	Credit card
Adam	XXXX-XXXX-XXXX-4444
Brenda	XXXX-XXXX-XXXX-2222

Name	Address
Adam	73 Nichols Av, Apt 52
Brenda	48 Park Dr., Apt 639

Name	Address
Adam	XX Nichols Av, Apt XX
Brenda	XX Park Dr., Apt XXX

Figure 2-14 Types of data masking that are available with FUJITSU Enterprise Postgres

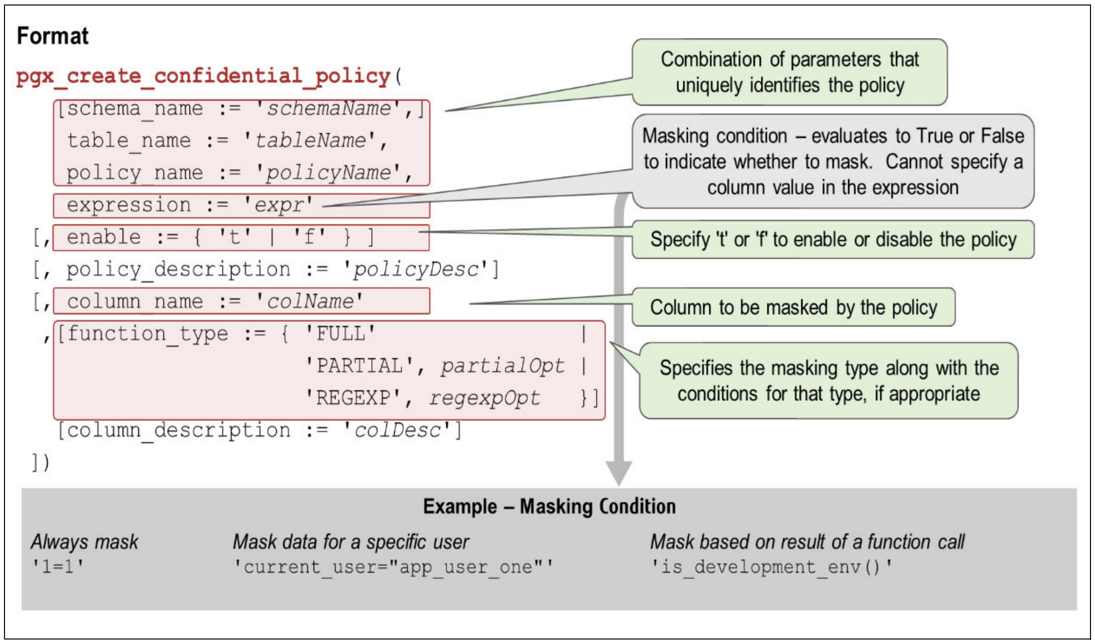
Data masking policy functions

FUJITSU Enterprise Postgres provides five data masking policy functions for creating, altering, enabling, disabling, and deleting the data masking policies. Table 2-2 on page 55 provides the list of these functions.

Table 2-2 Data masking policy functions

Data masking policy function	Description
`pgx_create_confidential_policy`	Create masking policies.
`pgx_alter_confidential_policy`	Change masking policies.
`pgx_enable_confidential_policy`	Enable or disable masking policies.
`pgx_update_confidential_values`	Changes replacement characters when full masking is specified for masking type.
`pgx_drop_confidential_policy`	Deletes masking policies.

Figure 2-15 shows the syntax for the Data Masking function `pgx_create_confidential_policy`, which is used to create full, partial, and regular expression data masking policies.



Format

```
pgx_create_confidential_policy(
    [schema_name := 'schemaName',]
    table_name := 'tableName',
    policy_name := 'policyName',
    expression := 'expr'
  [, enable := { 't' | 'f' } ]
  [, policy_description := 'policyDesc']
  [, column_name := 'colName'
   ,[function_type := { 'FULL'                |
                        'PARTIAL', partialOpt |
                        'REGEXP', regexpOpt   }]
    [column_description := 'colDesc']
  ])
```

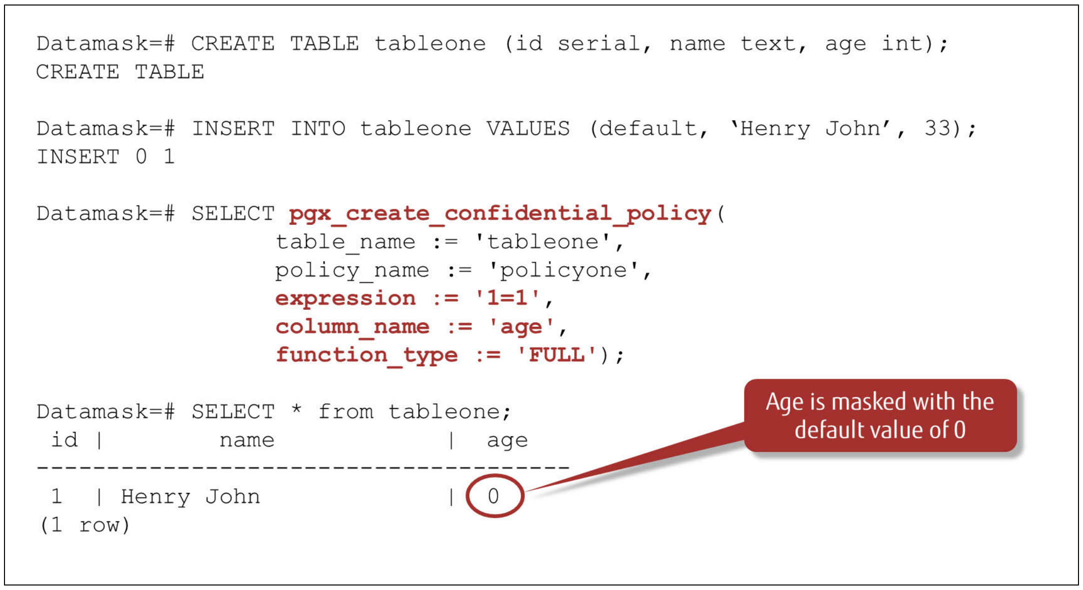
Combination of parameters that uniquely identifies the policy
Masking condition – evaluates to True or False to indicate whether to mask. Cannot specify a column value in the expression
Specify 't' or 'f' to enable or disable the policy
Column to be masked by the policy
Specifies the masking type along with the conditions for that type, if appropriate

Example – Masking Condition

Always mask	Mask data for a specific user	Mask based on result of a function call
`'1=1'`	`'current_user="app_user_one"'`	`'is_development_env()'`

Figure 2-15 Creating policy: Data masking system management function syntax

We now look at a simple example of creating a full data masking policy on a sample table. Figure 2-16 shows the implementation of a full data masking policy with the name policyone on table ‘tableone’ on column ‘age’. Column ‘age’ is defined as an integer column. By default, the masked data for the integer column is ‘0’. Here we see that the age of Henry John is 33 before applying a masking policy. The expression is defined as ‘1=1’, which implements an ‘always mask’ condition regardless of who accesses the column.



```
Datamask=# CREATE TABLE tableone (id serial, name text, age int);
CREATE TABLE

Datamask=# INSERT INTO tableone VALUES (default, 'Henry John', 33);
INSERT 0 1

Datamask=# SELECT pgx_create_confidential_policy(
                 table_name := 'tableone',
                 policy_name := 'policyone',
                 expression := '1=1',
                 column_name := 'age',
                 function_type := 'FULL');

Datamask=# SELECT * from tableone;
 id |         name           |  age
--------------------------------------
 1  | Henry John             |  0
(1 row)
```

Figure 2-16 Full data masking example: FUJITSU Enterprise Postgres

Note: For more information about various use cases and the use of data masking system management functions, see 4.4, “FUJITSU Enterprise Postgres encryption on IBM LinuxONE with crypto cards” on page 126, where we describe the steps to configure and implement the FUJITSU Enterprise Postgres Data Masking feature with examples.

2.2.3 Audit Log

Dedicated audit logging is a unique feature to FUJITSU Enterprise Postgres that helps organizations manage data accountability, traceability, and auditability. Audit Log is an important security feature that provides a level of protection while also meeting compliance expectations and protecting the brand and data asset value of an organization.

OSS PostgreSQL implements basic statement logging through the standard logging parameter `log_statement = all`. This implementation is acceptable for monitoring and other use cases, but this method does not provide the level of information that is required by security analysts for an audit. The information that is generated is not enough to list what the user requested. Ideally, the configuration must also establish what happened while the database was satisfying each request.

FUJITSU Enterprise Postgres Audit Logging enables organizations to trace and audit the usage of sensitive data and connection attempts of the database. Audit Logging provides a clear picture of data access by logging the following details:

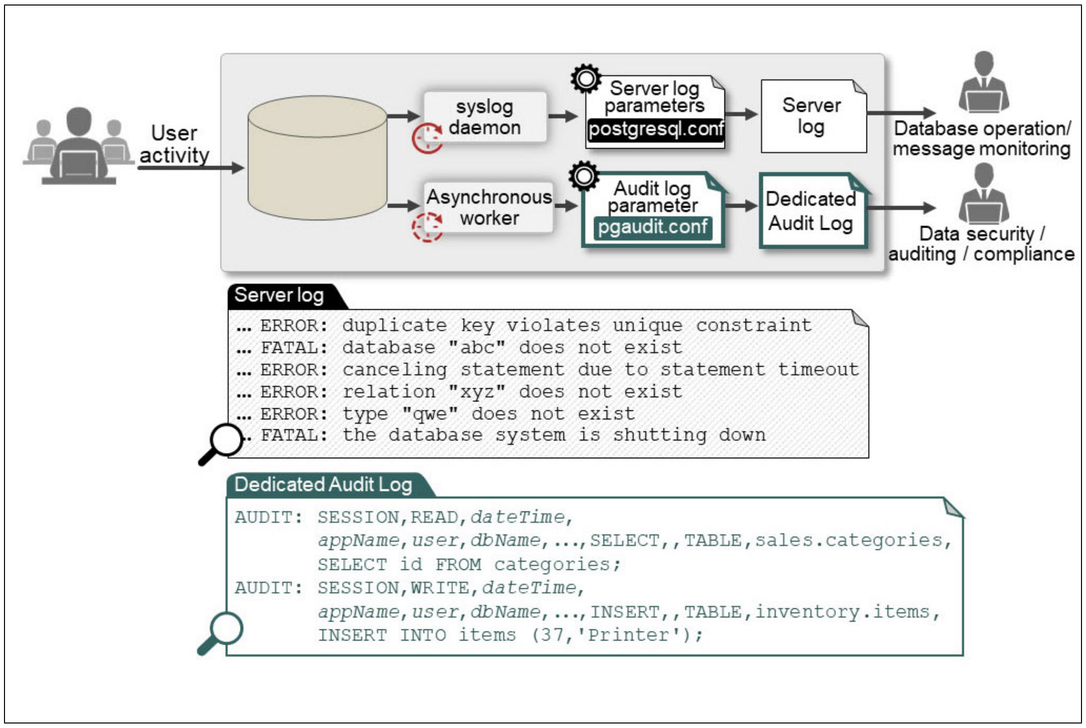
- What data is accessed.
- When the data is accessed.
- How the data is accessed and who accessed the data.

The information that is provided by audit logs can be used to counter security threats such as:

- Spoofing
- Unauthorized database access
- Privilege misuse

The Audit Log feature uses the **pgaudit** utility, which enables retrieval of details relating to database access as an audit log. Additionally, audit logs are externalized to a dedicated log file or server log to implement an efficient and accurate log monitoring.

Figure 2-17 shows the functioning differences between the server logs and the dedicated audit log (session log). The server log configuration parameters are set in the `postgresql.conf` file, but the audit log configuration is managed in the `pgaudit.conf` configuration file.



```
… ERROR: duplicate key violates unique constraint
… FATAL: database "abc" does not exist
… ERROR: canceling statement due to statement timeout
… ERROR: relation "xyz" does not exist
… ERROR: type "qwe" does not exist
… FATAL: the database system is shutting down
```

```
AUDIT: SESSION,READ,dateTime,
       appName,user,dbName,...,SELECT,,TABLE,sales.categories,
       SELECT id FROM categories;
AUDIT: SESSION,WRITE,dateTime,
       appName,user,dbName,...,INSERT,,TABLE,inventory.items,
       INSERT INTO items (37,'Printer');
```

Figure 2-17 FUJITSU Enterprise Postgres Audit Login action

With `pgaudit`, two types of audit log can be output:

- Session Audit Log
- Object Audit Log

Session Audit Logging

Session Audit Logging provides information about all the SQL statements that are run by the user in the background, information about starting and connecting to databases, and error information.

Object Audit Logging

When **SELECT**, **INSERT**, **UPDATE**, and **DELETE** are run for specific objects (tables and columns), Object Audit logging outputs them as a log. Object Audit Logging outputs object operations for which privileges are assigned to specified roles, as a log. Object Audit Logging can control log output at an even finer level of granularity than Session Audit Logging.

Table 2-3 provides information about the statements that are logged by Session Audit Log and Object Audit Log.

Table 2-3 Audit Log class statements for Session and Object Audit Logging

Session Audit Log class of statements	Object Audit Log command tags
► READ: **SELECT, COPY FROM** ► WRITE: **INSERT, UPDATE, DELETE, TRUNCATE, COPY TO** ► FUNCTION: Function call, DO ► ROLE: **GRANT, REVOKE, CREATE ROLE, ALTER ROLE, DROP ROLE** ► DDL: All DDLs (such as **CREATE** and **ALTER**) other than the DDLs of the ROLE class ► CONNECT: Events relating to connecting (request, authenticate, and disconnect) ► SYSTEM: Instance start, promotion to primary server ► BACKUP: **pg_basebackup** ► ERROR: Event completed by an error (PostgreSQL error codes other than 00). This class can be used if **ERROR** or a lower level is specified for the **log_min_messages** parameter in `postgresql.conf`. ► MISC: Other commands (such as **DISCARD**, **FETCH**, **CHECKPOINT**, and **VACUUM**)	► **SELECT** ► **INSERT** ► **UPDATE** ► **DELETE**

Example 2-1 shows an example of Object Audit Log and Session Audit Log outputs.

Example 2-1 Session Audit Log and Object Audit Log outputs

```
[fsepuser@rdbkpgr4 pgaudit_log] $ pwd
/home/fsepuser/testdb/pgaudit_log
[fsepuser@rdbkpgr4 pgaudit_log] $ cat pgaudit-2020-11-
08_220044log
AUDIT: OBJECT,1,1,WRITE,INSERT,TABLE,public.account,"insert into
account values(102,'user2','pwd102','test user2');",<not logged>

AUDIT:SESSION,WRITE,2020-11-08 22:01:19
EST,[local],13456,psql,fsepuser,postgres,3/2,1,1,INSERT,,TABLE,pu
blic.account,,"insert into account val-
ues(102,'user2','pwd102','test user2');",<not logged.

AUDIT: OBJECT,2,1,READ,SELECT,TABLE,public.account,select * from
account;<not logged>

AUDIT: SESSION,READ,2020-11-08 22:01:23
EST,[local,13456,psql,fsepuser,postgres,3/3,2,1,SELECT,,TABLE,pu
blic.account,,select * from account;,<not logged>
```

Note: An Audit Logging output where the setup has a dedicated log file follows the standard format, which can be treated as CSV, so the log file contents can easily be converted into SQL tables by performing simple steps. Such transformation requires more modules of the Foreign Data Wrapper (`file_fdw`) type. The `file_fdw` module comes packaged with FUJITSU Enterprise Postgres.

In 4.4, “FUJITSU Enterprise Postgres encryption on IBM LinuxONE with crypto cards” on page 126, we describe the use cases of configuring the Audit Log and using `file_fdw` to transform data into tables for ingesting, analyzing, and storing the audit log for compliance purposes.

2.3 Performance

One of the driving forces behind business innovation is using information to gain insight about business and respond as needed. This task becomes more challenging because the world generates 10 zettabytes (a trillion gigabytes) of data every year, and this number is projected to increase to 175 zettabytes in 2025.

Regardless of the choices that businesses make, IT systems must adapt to keep pace with exponential data growth and be able to quickly analyze growing amounts of data. Every minute that is saved in ingesting and analyzing data has tremendous value, so the demand to quickly perform analysis puts tremendous pressure on IT systems. Business and data analysts are always looking for faster ways of processing data aggregation and manipulation supporting business leaders to make business decisions.

FUJITSU Enterprise Postgres offers three performance features for enhancing ingesting and analyzing data by fully optimizing the processor, memory, storage, and I/O resources:

- VCI
- FUJITSU High-Speed Data Load
- Global Meta Cache (GMC)

2.3.1 Vertical Clustered Index: Increased aggregation performance

VCI is a proprietary implementation of in-memory columnar storage (IMCS), which provides faster data analysis to support business intelligence. VCI provides faster aggregation performance while reducing the I/O of processing the in-memory data. The design thinking behind VCI is to enable the businesses to perform real-time analytics on changing data sets without needing to Extract, Transform, and Load (ETL) data from Online Transaction Processing (OLTP) to Online Analytical Processing (OLAP) systems, and without any extra cost.

Columnar index

Columnar storage architecture has gained attention with the introduction of Hybrid Transactional Analytical Processing (HTAP), which is an alternative to merging OLTP with OLAP systems. One of the major challenges that exists in various product implementations of HTAP processing is maintaining the ACID compliance of an RDBMS. The goal of HTAP data processing is to remove the ETL processing to an extent that enables real-time analytics on the OLTP systems before they go to OLAP systems through ETL, which traditionally has been the *de facto* method to connect OLTP and OLAP DBMSs. Although row-oriented data is better suited to OLTP processing, it’s the column-oriented data that is best suited for data analysis because it improves aggregation performance.

Aggregations usually process a significant amount of data from a particular column or columns. For such aggregations in the traditional row data structure, columns that are not queried are also fetched, which leads to inefficient memory and CPU cache usage and causing slower query processing. However, columnar storage uses vertical architecture for placing rows adjacently because the query processing does not perform scans on columns that are not in the scope of aggregation computation, which improves aggregation query performance. VCI is the Fujitsu columnar storage implementation offering in the form of an index, hence the name VCI.

VCI has two components to achieve aggregation performance: One is on disk, and another one is in memory. Both the components are managed by the sophisticated *VCI Analysis Engine*.

Figure 2-18 shows that the VCI Analysis Engine combines VCI columnar storage on disk by using the **`pgx_prewarm_vci`** utility to preload VCI pages and make it memory resident. VCI data is stored in the dedicated portion of the shared buffers on memory. VCI memory allocation share is controlled by the configuration parameter **`reserve_buffer_ratio`**.

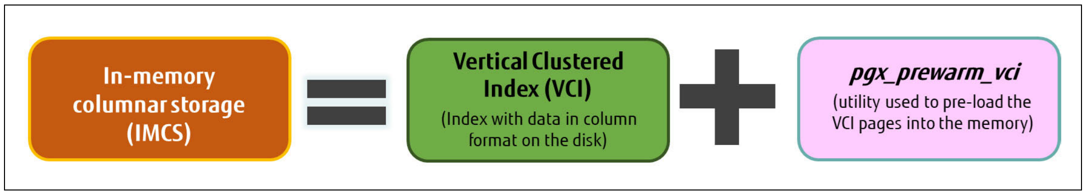

In-memory columnar storage (IMCS)
Vertical Clustered Index (VCI)
(Index with data in column format on the disk)
pgx_prewarm_vci
(utility used to pre-load the VCI pages into the memory)

Figure 2-18 VCI disk and memory components

VCI benefits

VCI provides the following benefits:

- Minimizes the impact on existing jobs and performs aggregation by using job data in real time.
- VCI is crash-safe. It also stores data on the disk, so aggregation jobs can be quickly resumed by using a VCI even if a failure occurs (when an instance is restarted).
- If the amount of memory that is used by VCI exceeds the set value, aggregation can continue by using VCI data on the disk to avoid impacting on-going operations.
- VCI is provided as an index, so no application modification is required.
- In HA architectures, VCI provides the opportunity to optimize the extra computing power that is available with standby servers for performing resource-intensive data analysis faster.

To create a VCI, use the **`CREATE INDEX`** statement, as shown in Example 2-2.

Example 2-2 Creating a Vertical Clustered Index by using the CREATE INDEX statement

```
CREATE INDEX item_type_vci
ON sale_hist
USING vci (item, type) WITH (stable_buffer=true);
```

VCI features

Here are some of the features of VCI:

- Disk compression: Compresses VCI data on the disk, minimizing the required disk space. Even if disk access is required, the read overhead is low.
- Parallel scan: Enhances aggregation performance by distributing aggregation processes to multiple CPU cores and then processing them in parallel.
- Preload feature: Preloads VCI data into memory before an application scans VCI and ensures stable query response times.
- Stable buffer feature: Maintains VCI data in memory, which improves query performance because it reduces disk I/Os by avoiding VCI eviction from memory by other jobs.

Note: The Preload and Stable buffer features keep VCI data in memory and minimize disk I/Os on each aggregation process.

VCI architecture

In a hybrid architecture that supports both row and columnar orientation of data, the row data must be converted to columnar data, which impacts the online transactions processing performance. This problem is solved in VCI by using two buffer storage areas that are called Write Optimized Store (WOS) for row data and Read Optimized Store (ROS) for columnar data.

Figure 2-19 (column A) shows the internal processing that occurs when an SQL query uses VCI for faster aggregation performance.

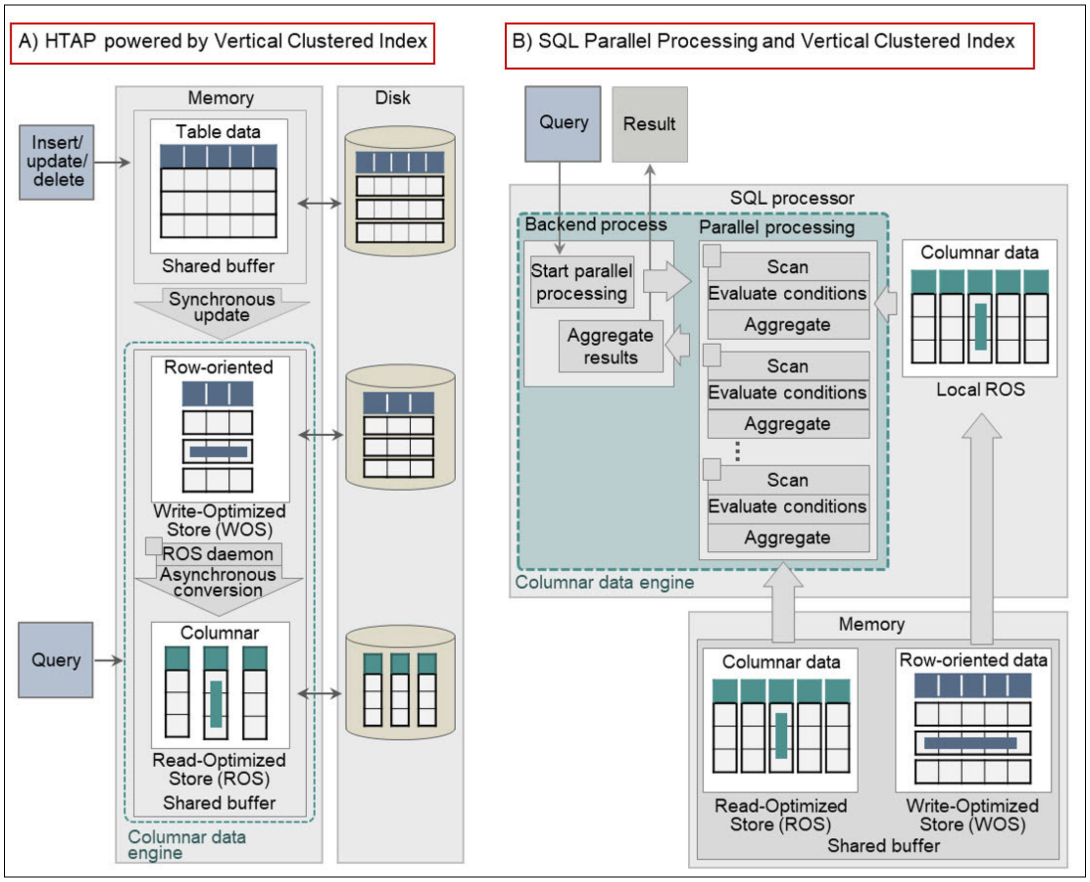


Figure 2-19 HTAP processing by using Vertical Clustered Index

When FUJITSU Enterprise Postgres runs an update operation, the data is written only to the WOS. In this step, the data compression or indexing does not avoid any overhead during the columnar format conversion, which improves performance. The performance is further increased by the columnar data engine writing only the record IDs and VCI-indexed columns to WOS. When a certain amount of data is written to WOS, it is converted into columnar data asynchronously and written to ROS by using indexing and compression. So, the columnar data engine balances data synchronization between OLTP transactions and OLAP performance by separating the processes of storing OLTP data and data conversion to columnar format for OLAP queries.

Figure 2-19 on page 61 (column B) shows that WOS and ROS are combined when data aggregation is performed. Internally, a Local ROS temporary area is created in the SQL processor from the row-oriented data that is stored in WOS. Then, the Local ROS data is compared with the ROS data that was already converted to columnar format. This task is performed to merge any differential data for each record. Next, aggregation is performed by using the result set, which gives the same query result as though the OLTP data was used.

The improved performance is achieved by the columnar data engine, which performs scan, aggregation, and analysis by using the memory buffers shown in Figure 2-19 on page 61 [A] and Figure 2-19 on page 61 [B]. During a system crash, the columnar data that is stored in buffers might be lost, but because the columnar data is also saved on disk, VCI is crash-safe. VCI columnar data can be immediately copied to memory after the system restart. Furthermore, VCI also supports data overflow from the buffers on disk, which ensures that the operations are maintained without obvious performance deterioration.

We now understand that the columnar data format improves read performance for aggregation, but it alone does not make columnar storage performant. To improve performance further, FUJITSU added the *Analysis Engine* for optimizing columnar data parallel processing, as shown in Figure 2-19 on page 61 [B]. The Analysis Engine can apply the same processing concurrently on multiple columns by using vector processing, thus improving the VCI performance.

Example 2-2 on page 60 shows that creating a VCI is a simple command that uses the **`CREATE INDEX`** statement. After the VCI index is created, the SQLs that have aggregation in scope perform aggregation in the usual way, which means that a VCI index can be created on existing tables without changing the application code or specifying which index or data format to use. This part is taken care of by the execution plan.

VCI configuration parameters

Table 2-4 shows important VCI configuration parameters that are set before VCI indexes are created and used in the database instance. The configuration of these parameters depends on the database workload under consideration. For example, **`reserve_buffer_ratio`** is an important parameter that sets the percentage of the overall shared buffer that will be fixed for the VCI operation. This percentage value is fine-tuned based on performance testing iterations.

Table 2-4 VCI configuration parameters in postgresql.conf

VCI parameter name	Setting value	Description
`shared_preload_libraries`	Literal '`vci, pg_prewarm`'	VCI and shared library to be preloaded at server start.
`session_preload_libraries`	Literal '`vci, pg_prewarm`'	VCI and shared library to be preloaded at connection start.

VCI parameter name	Setting value	Description
`reserve_buffer_ratio`	Percentage of shared memory to be used for stable buffer table.	Proportion of shared memory to be used for a stable buffer table.
`vci.control_max_workers`	Number of background workers that manage VCI.	Number of background workers that manage VCI.
`vci.max_parallel_degree`	Maximum number of background workers used for parallel scan.	Maximum number of background workers used for parallel scan.

Note: As VCI is a performance enhancement feature, a few factors are considered for performance tuning the VCI, which includes application workload, transactions per second, service-level agreements (SLAs), replication, and partitioning considerations. As a best practice, performance testing should be conducted for each use case that is considered for implementing VCI like any other new indexing.

For more information about the evaluation criteria for installing VCI, resource estimation, and configuration steps, see 9.1, “Installing Vertical Clustered Index (VCI)”, of the *FUJITSU Enterprise Postgres Operation Guide*.

Sample VCI configuration

Example 2-3 shows a sample configuration of VCI that is used in the lab environment that we used to create this publication. The `shared_buffers` entry is defined as 12 GB, and `reserve_buffer_ratio` is 30% of `shared_buffers`, which equals 3.6 GB.

Example 2-3 Sample VCI configuration on primary, standby, cascade, and backup database instances

```
$ vim /database/inst1/postgresql.conf
shared_preload_libraries = 'vci, pg_prewarm'
session_preload_libraries = 'vci, pg_prewarm'
shared_buffers = 12GB
reserve_buffer_ratio = 30
vci.control_max_workers = 8
vci.max_parallel_degree = 4
max_worker_processes = 18
```

Query planner

VCI optimizes the available CPU and memory within the server to enhance scan performance. It further speeds up execution time for the following use cases:

- Single table processing
- Queries processing large aggregations, such as simultaneous sum and average operations
- Queries fetching and processing many rows in the tables

Before the aggregation is performed, the *query planner* computes the execution cost, and based on an algorithm, it implements the most cost-efficient plan to determine whether VCI processing is regardless of the availability of VCI.

Example 2-4 explains the analysis output for a simple aggregation SQL. The query planner selects VCI Scan by using the VCI index `vci_item_type` on the `sale_hist` table and implements parallel degree 4 by using four workers to achieve performance.

Example 2-4 Explaining the analysis output by showing VCI query planner costs

```
EXPLAIN ANALYZE
       SELECT item, SUM(qty) FROM sale_hist
       GROUP BY item, type;
                                  QUERY PLAN
-----------------------------------------------------------------------------------
Custom Scan (VCI Aggregate) (cost=19403.15 .. 19403 . 16 rows=991261 width=47)
(actual time=58.505 .. 58.506 rows=100 loops=1)
Allocated Workers: 4
- > Custom Scan (VCI Scan) using vci_item_type on sale_hist
(cost=0.00..16925.00 rows=991261 width=47)
Planning time: 0.151 ms
Execution time: 86.910 ms
```

Through VCI, FUJITSU Enterprise Postgres enables organizations to consolidate critical business intelligence analytics queries on the OLTP systems from OLAP systems. Furthermore, the parallelization of the aggregation process further reduces the query execution time with VCI. Together, both features provide HTAP system design capability to the database designers.

Note: IBM and FUJITSU conducted various benchmark test on low, mid, and high IBM LinuxONE III configurations. The purpose of these benchmarks was to optimize and leverage Integrated Facility for Linux processing and I/O utilization. The benchmarks covered TPC-C, TPC-H, and customized HTAP benchmarks for different sizes of workloads. For more information about the benchmarks results, contact your IBM representative or see *Improving Data Performance*.

In Chapter 8, “Database performance features of FUJITSU Enterprise Postgres” on page 289, we describe a VCI use case implementation in a HA environment with streaming replication between primary and standby database instances. We showcase the use case to load balance analytical queries by using VCI on the standby database instance without off loading them to separate OLAP systems.

2.3.2 FUJITSU High-Speed Data Load: pgx_loader

FUJITSU High-Speed Data Load is designed to perform bulk loading by using multiple parallel processes that depend on the availability of the number of CPU cores or virtual Integrated Facility for Linux (IFL) processors that are available. This approach makes the best use of the available virtual IFL processors or CPU cores without pre-configuring and performance tuning the environment.

FUJITSU High-Speed Data Load distributes the data from an input file to several parallel workers. Each worker performs data conversion, table creation, and index creation, which tremendously reduces the data load time.

Architecture

Generally, bulk data load utilities run several **INSERT** SQL statements by using various techniques that try to efficiently read chunks of data, covert the data into the correct format, update the table and associated indexes, and maintain business rules that are enforced through database referential integrity constraints.

FUJITSU High-Speed Data Load uses the PostgreSQL **COPY** command with multiple parallel workers where the parallelism degree is maintained by the utility itself per the IFL processors / CPU availability during run time. The degree of parallelism is controlled through the configuration parameters.

As shown in Figure 2-20, FUJITSU High-Speed Data Load dynamically allocates parallel workers that are tasked to serially perform data conversion, data insertions into table, and index keys creation. Depending on the number of IFL processors / CPU cores that are available, performance gains are achieved compared to the PostgreSQL COPY command alone.

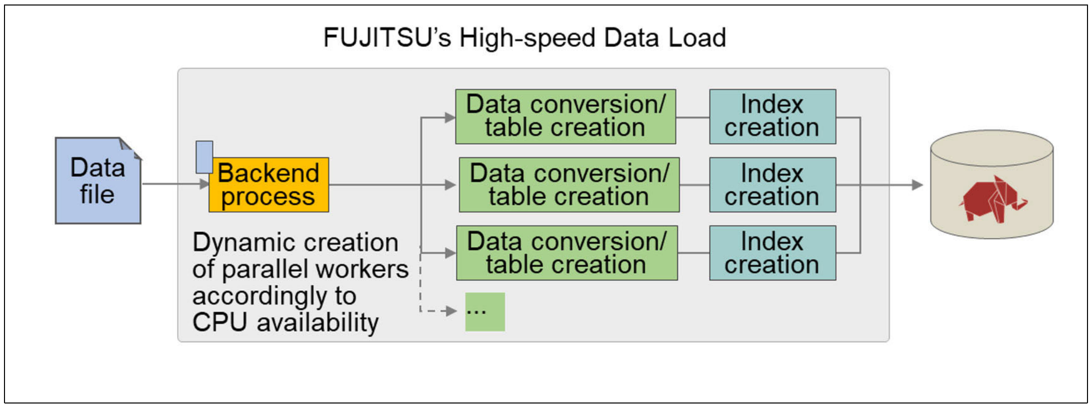

FUJITSU's High-speed Data Load
Data file
Backend process
Dynamic creation of parallel workers accordingly to CPU availability
Data conversion/ table creation
Index creation
Data conversion/ table creation
Index creation
Data conversion/ table creation
Index creation
...

Figure 2-20 FUJITSU High-Speed Data Load parallel processing of input file while optimizing IFL usage

Installation

Installing FUJITSU High-Speed Data Load is a simple process that uses the `pgx_loader` extension, as shown in Example 2-5. The `pgx_loader` can be started in two modes: load mode and recovery mode.

Example 2-5 FUJITSU High-Speed Data Load installation through CREATE EXTENSION

```
postgres=# CREATE EXTENSION pgx_loader;
CREATE EXTENSION
```

Configuration

There are a few parameters that should be updated in the `postgresql.conf` file.Table 2-5 on page 66 shows three important parameters that impact the FUJITSU High-Speed Data Load utility performance.

Table 2-5 FUJITSU High-Speed Data Load configuration parameters in postgresql.conf

Parameter	Setting	Required?
max_prepared_transactions	Add the number of transactions that can be prepared by parallel workers during data load to the parameter's current value. The resulting value must be equal to or greater than the maximum number of parallel workers that is used with this feature.	Yes
max_worker_processes	Number of parallel workers to perform a data load.	Yes
max_parallel_workers	Add the maximum number of parallel workers to be used in a parallel query by this feature to the parameter's current value. The resulting value must be equal to or greater than the number of parallel workers that is used with this feature.	Yes

The listed parameters are GUC parameters, which means that the existing values should be updated if FUJITSU High-Speed Data Load is installed. For example, if FUJITSU High-Speed Data Load is configured on a server with two instances and the parallel degree set to 8, then the product of 2 x 8 = 16 should be added to each of the parameters.

Table 2-6 shows the calculated values of the parameters.

Table 2-6 Example for calculating parameters values that are related to FUJITSU High-Speed Data Load

Parameter	Initial value	New value
max_prepared_transactions		=> 5 + [2 x 8] = 21
max_worker_processes		=> 8 + [2 x 8] = 24
max_parallel_workers		=> 4 + [2 x 8] = 20

Note: Up to 128 parallel workers can be specified to use FUJITSU High-Speed Data Load. FUJITSU High-Speed Data Load also creates dynamic shared memory and shared memory message queues during the load process that are used to send data from the back end (input files) to the parallel workers and for error messages.

Loading data

The **pgx_loader** command loads data from an external file into FUJITSU Enterprise Postgres tables and commits or roll backs transactions that are prepared during the load. **pgx_loader** can be started in two modes: load mode and recovery mode. In Example 2-6, we use **pgx_loader** in load mode to copy a sales table from the `sales.csv` file with degree of parallelism of four.

Example 2-6 Using pgx_loader to load data

```
$ pgx_loader load -j 4 -c "COPY sales FROM '/path/sales.csv' WITH CSV"
LOAD 2000
```

Note: Option **`-j=4`** specifies that the **`COPY`** command should use four background workers (parallel workers) to simultaneously perform data conversion, table creation, and index creation.

For more information about the **`pgx_loader`** command syntax, see 2.2, "pgx_loader", in the *FUJITSU Enterprise Postgres 11 Reference Guide*. For more information about the **`COPY FROM`** command, see "COPY" in the same section.

Checking results

FUJITSU High-Speed Data Load provides information about transactions that are prepared by **`pgx_loader`** through the **`pgx_loader_state`** table.Example 2-7 shows an SQL query that is used to check whether all the data was successfully loaded.

Example 2-7 SQL to check the existence of any pgx_loader transaction after the load completes

```
SELECT gid
FROM   pg_prepared_xacts
WHERE  gid IN
       ( SELECT 'pgx_loader' || gid
              FROM   pgx_loader.pgx_loader_state
              WHERE  state = 'rollback'
              AND    role_oid IN
                     ( SELECT oid
                            FROM   pg_roles
                            WHERE  rolname = 'myrole')
              AND relation_oid IN
                     ( SELECT relid
                            FROM   pg_stat_all_tables
                            WHERE  relname = 'sales');
```

Note: The **`pgx_loader_state`** table and **`pgx_loader_state_id_seq`** sequence are updated by FUJITSU High-Speed Data Load. Do not update these database objects directly by using SQL.

If there is any issue, run **`pgx_loader`** in recovery mode by using the **`-t`** option, as shown in Example 2-8.

Example 2-8 Using pgx_loader to load data

```
$ pgx_loader recovery -t sales
```

In Chapter 8, "Database performance features of FUJITSU Enterprise Postgres" on page 289, we describe FUJITSU High-Speed Data Load use cases and compare the results with the output of the OSS PostgreSQL **`COPY`** command. We also describe the performance tuning considerations for FUJITSU High-Speed Data Load.

2.3.3 Global Meta Cache

Caching is considered an important database performance feature. Database performance experts try to reduce I/O as much as possible by using different techniques at hardware and software layers. Hence, fetching data from storage frequently impacts the performance of workloads that use large amounts of random I/O, such as OLTP and HTAP workloads. This problem is overcome in PostgreSQL by caching data in memory, which tremendously improves performance. PostgreSQL caches table data, indexes, and query execution plans.

Query processing goes through a few steps before a query execution plan is created. The steps include parsing the query, creating a plan, and running the plan. To perform these steps, the PostgreSQL process accesses the system catalogs, during which the system catalog data is cached in memory per process.

Moreover, the metadata that is required to access the tables that are referenced in SQL queries and stored in catalog tables is also aggregated and cached. This cached information in the memory is called a *meta cache.* The purpose of a meta cache is to improve the query processing performance by providing metadata in memory to avoid metadata searches in the catalog table each time. As the number of transactions and connections and their size increases, the cache size increases because the metadata is cached per process, which increases the system's overall meta cache and causes cache bloat, which impacts the performance. This performance issue is resolved by GMC.

FUJITSU GMC is a method to avoid cache bloat by centrally managing the meta cache in shared memory instead of per process. Because the meta cache is managed centrally, it is called GMC. Figure 2-21 on page 69 shows this phenomenon with GMC OFF and ON.

Architecture

Without GMC, the meta cache is stored in per-process memory, and with the GMC feature on, the per-process meta cache is cached in the GMC area on shared memory. Figure 2-21 on page 69 shows the difference between the GMC off and GMC on scenarios. With GMC on, the PostgreSQL process searches the meta cache for each process and uses the meta cache header to access the meta cache in the GMC area. When the meta cache is not found in the GMC area, then the PostgreSQL process can access the system catalogs and create the meta cache.

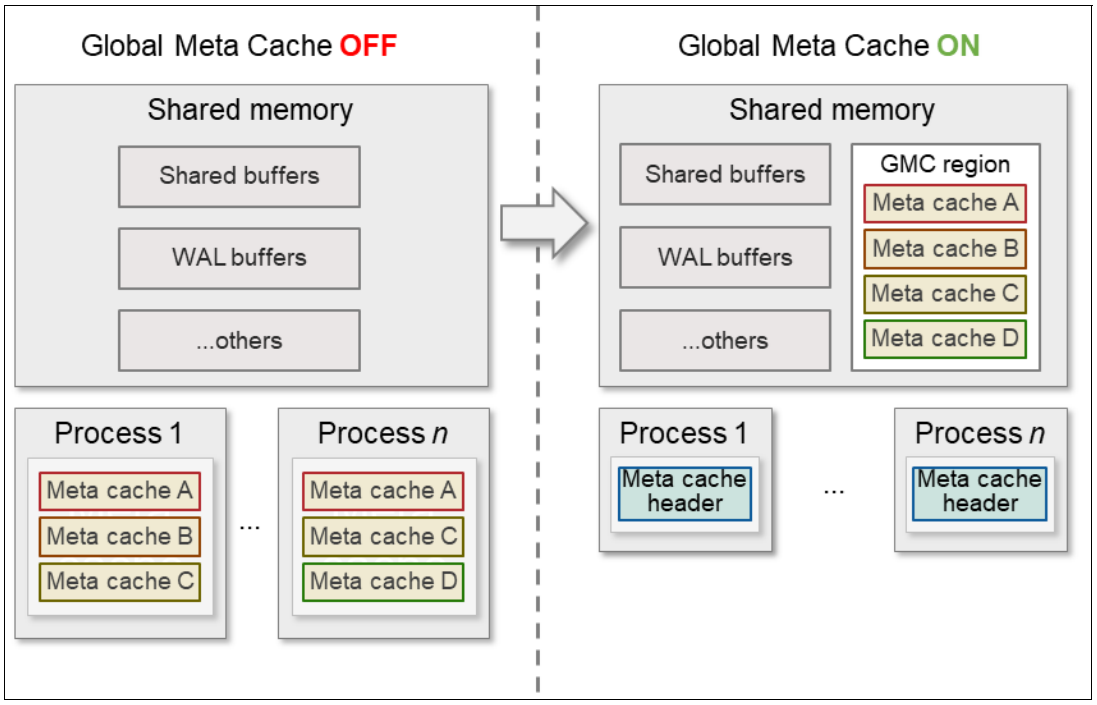


Figure 2-21 Meta cache without GMC and with GMC

After the transaction commits, the GMC area cache is deleted or created. The deletion of the GMC area cache is deferred if any other transactions are referring to the cache when the deletion of the GMC area cache is started. The deletion is deferred until there are no more references.

Note: GMC is disabled by default.

Benefits

Here are some of the benefits of GMC:

- Each process can access the GMC and does not need the information in per-process memory if the memory of the processes is loaded into the GMC.
- A process does not need to scan the catalog tables if the data is loaded into GMC, which reduces the amount of memory that is required throughout the system.
- GMC is a feature for sharing meta caches between processes, so it works well on systems with a high number of resources that are accessed and connections.
- Best use case for GMC is to consider using it when the size of meta cache for each process exceeds the number of installed memories or takes up a large portion of memory, which squeezes memory allocations to the database cache or OS cache sizes.

Enabling GMC

FUJITSU Enterprise Postgres provides the **`pgx_global_metacache`** parameter in `postgresql.conf` to specify the size of the GMC in shared memory. Example 2-9 shows a sample configuration of the **`pgx_global_metacache`** parameter.

Example 2-9 pgx_global_metacache parameter example

```
pgx_global_metacache = 900 MB
```

In Example 2-9, the maximum allocated GMC is 900 MB. The inactive meta caches are removed from the GMC area if the total number of meta caches on shared memory exceeds this value of 900 MB. If a cache cannot be created in the GMC area because all 900 MB is in use, then the cache is created in the local memory of the back-end process on a *ad hoc* basis.

Note: GMC is disabled by specifying `pgx_global_metacache=0 MB`. The minimum allowed value is 10 MB.

In Chapter 8, "Database performance features of FUJITSU Enterprise Postgres" on page 289, we demonstrate the implementation of GMC and describe the difference in memory utilization with and without GMC on. We also demonstrate the use of the **`pgx_stat_gmc`** table to capture the cache hit ratio and understand the effectiveness of using GMC.

2.3.4 Performance tuning

FUJITSU Enterprise Postgres selects the least expensive query plan based on the query planner estimates. The query planner estimates the cost by using algorithms that parse SQL statements and statistical information from catalog tables. For mission-critical systems, it might not be possible to have the most updated statistics. Often, the volatility of the statistics of database objects leads to unstable query plans and impacts performance. To resolve these issues, FUJITSU Enterprise Postgres uses optimizer hints and locked statistics to achieve stable query performance.

Optimizer hints

Optimizer hints enable FUJITSU Enterprise Postgres to stabilize the query plan in mission-critical systems. Users can use **`pg_hint_plan`** to specify a query plan as a hint in each individual SQL statement.

FUJITSU Enterprise Postgres uses the open-source module **`pg_hint_plan`** to provide optimizer hints for query planning. The query planner estimates the cost of each possible plan that is available for the SQL statement and picks the low-cost plan because the query planned considers it the best plan, but that plan might not be best because the query planner does not know some details like the association between the columns. Optimizer hints come handy in such situations because it suggests to the planner to choose a specified plan.

Note: For more information about **`pg_hint_plan`**, see GitHub.

Fixed statistical information

Locked statistics is a feature that locks the various relational statistics in custom views. During query plan generation, the optimizer component of the server calculates the costs based on these locked statistics. This way, query plan generation is stabilized because every time these locked statistics are used to generate the plan, the risk of a sudden execution plan change in the customer environment is reduced. This function is achieved by installing and using **`pg_dbms_stats`**. Locked statistics also help to reproduce the same query plan in both testing and production environments.

Note: For more information about **`pg_dbms_stats`**, see GitHub.

2.3.5 Data compression by using IBM zEnterprise Data Compression

As described in 2.2.3, "Audit Log" on page 56, FUJITSU Enterprise Postgres provides a unique backup and recovery feature through the **`pgx_dmpall`** and **`pgx_rcvall`** commands, which enable administrators to perform online backups with continuous archiving or recover backed up data and archived WALs with a single action.

In addition, FUJITSU Enterprise Postgres on IBM LinuxONE works with the on-chip compression accelerator IBM zEnterprise Data Compression (IBM zEDC) to ensure efficient backup operations in the event of database or hardware failures. IBM zEDC reduces CPU costs by 90% and processing time during data compression by 40% compared to traditional software compression. For more information about IBM zEDC, see Data compression with the Integrated Accelerator for zEDC.

2.4 Fujitsu developed database management software

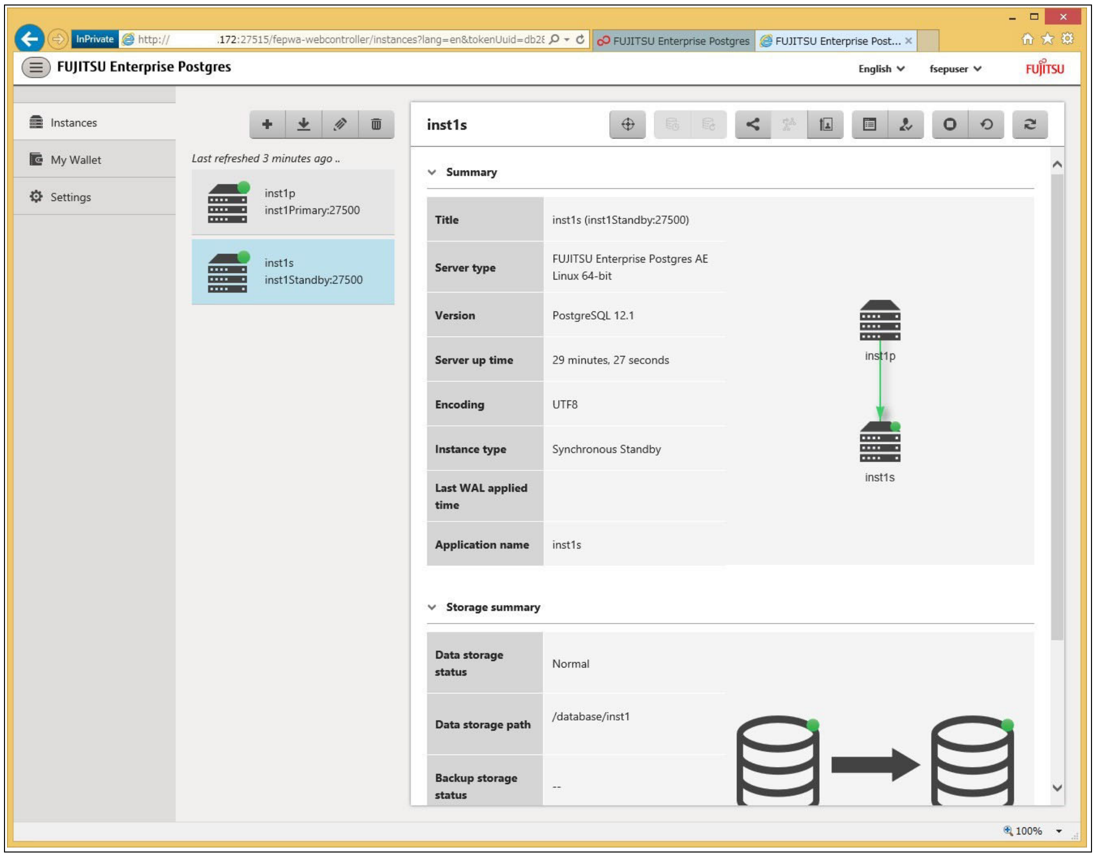
Fujitsu developed specific software tools that are used in database management. In this section, we describe WebAdmin, which is the Fujitsu GUI tool for many tasks, such as database installation and database operations management. WebAdmin can be used for the following tasks:

- FUJITSU Enterprise Postgres setup. Instances can be created easily with minimal input because the tool automatically determines the optimal settings for operation.
- Creating and monitoring a streaming replication cluster.
- Database backups and recovery.
- Manage FUJITSU Enterprise Postgres instances in a single server or instances that are spread across multiple servers.

WebAdmin features include:

- Fast Recovery, which uses WebAdmin as a technology to shorten the rebuild time:
  - Recovery that uses WebAdmin requires less time and effort because WebAdmin automatically determines the scope of the operation.
- Wallet feature
  - The Wallet feature in WebAdmin is used to conveniently create and store instance username and passwords.
  - This feature can be used to create usernames and passwords when creating remote standby instances through WebAdmin.
  - After the credentials are created in Wallet, they can be used repeatedly.

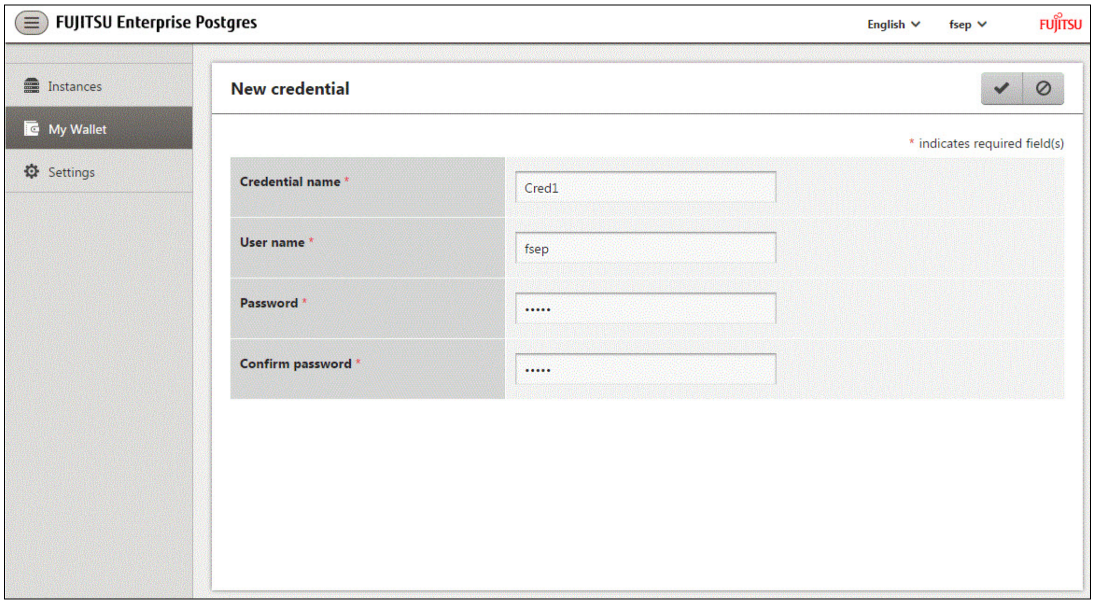
Figure 2-22 and Figure 2-23 shows the implementation of the Fujitsu database multiplexing feature and Wallet feature.



Title	inst1s (inst1Standby:27500)
Server type	FUJITSU Enterprise Postgres AE Linux 64-bit
Version	PostgreSQL 12.1
Server up time	29 minutes, 27 seconds
Encoding	UTF8
Instance type	Synchronous Standby
Last WAL applied time	
Application name	inst1s

Data storage status	Normal
Data storage path	/database/inst1
Backup storage status	--

Figure 2-22 WebAdmin showing the database multiplexing feature implementation with primary and standby instances

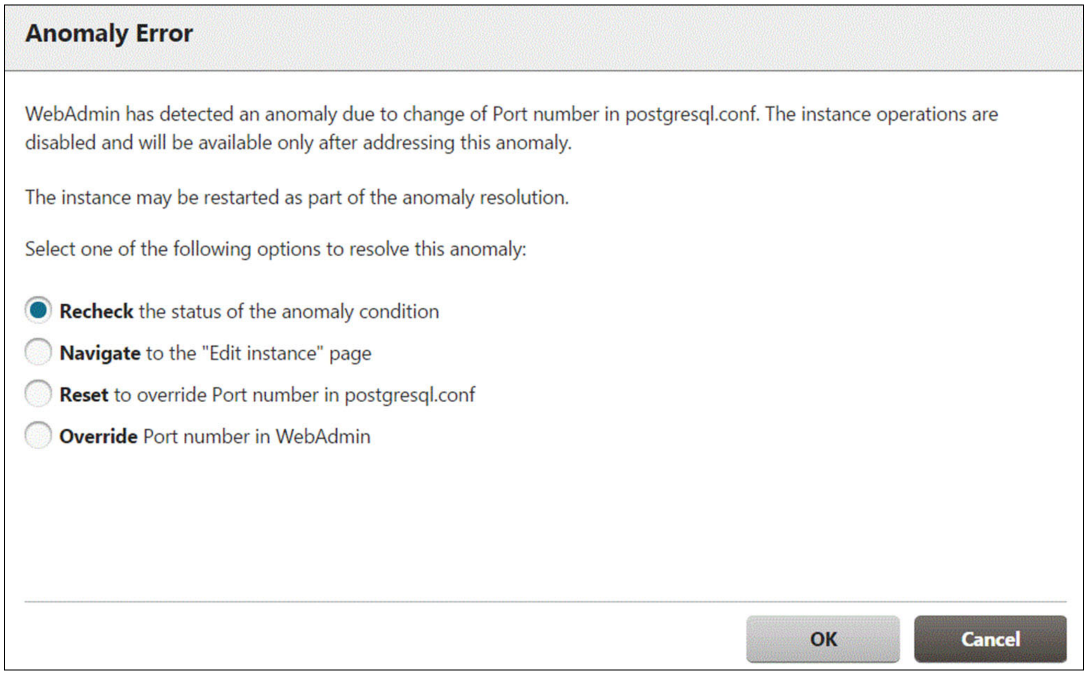


Credential name *	Cred1
User name *	fsep
Password *	•••••
Confirm password *	•••••

Figure 2-23 WebAdmin Wallet feature for creating and storing database instance credentials

- Anomaly Detection and Resolution: When certain operations are performed through the CLI, they can cause anomalies in WebAdmin. These operations are:
  - Changes to the **`port`** and **`backup_destination`** parameters in `postgresql.conf`.
  - Changes to the MC configuration of cluster replication that is added through WebAdmin.
  - WebAdmin checks for anomalies when an instance is selected for viewing or any instance operation is performed.

  Figure 2-24 shows an example of anomaly detection due to a change in port number in `postgresql.conf`, and suggests actions to resolve the issue.



Anomaly Error

WebAdmin has detected an anomaly due to change of Port number in postgresql.conf. The instance operations are disabled and will be available only after addressing this anomaly.

The instance may be restarted as part of the anomaly resolution.

Select one of the following options to resolve this anomaly:

- [✓ **Recheck** the status of the anomaly condition]
- **Navigate** to the "Edit instance" page
- **Reset** to override Port number in postgresql.conf
- **Override** Port number in WebAdmin

Figure 2-24 WebAdmin anomaly detection feature

WebAdmin configurations

WebAdmin can be installed as either of two configurations:

- Single server
- Multi-server

Note: WebAdmin does not support encrypted communication between a browser and server or between servers. Hence, when using WebAdmin in either configuration, the network communication path should be built on a private network with no external access.

WebAdmin single-server configuration

In a single-server configuration, WebAdmin is installed on the same database server as the FUJITSU Enterprise Postgres Server component. Figure 2-25 shows a single-server WebAdmin configuration.

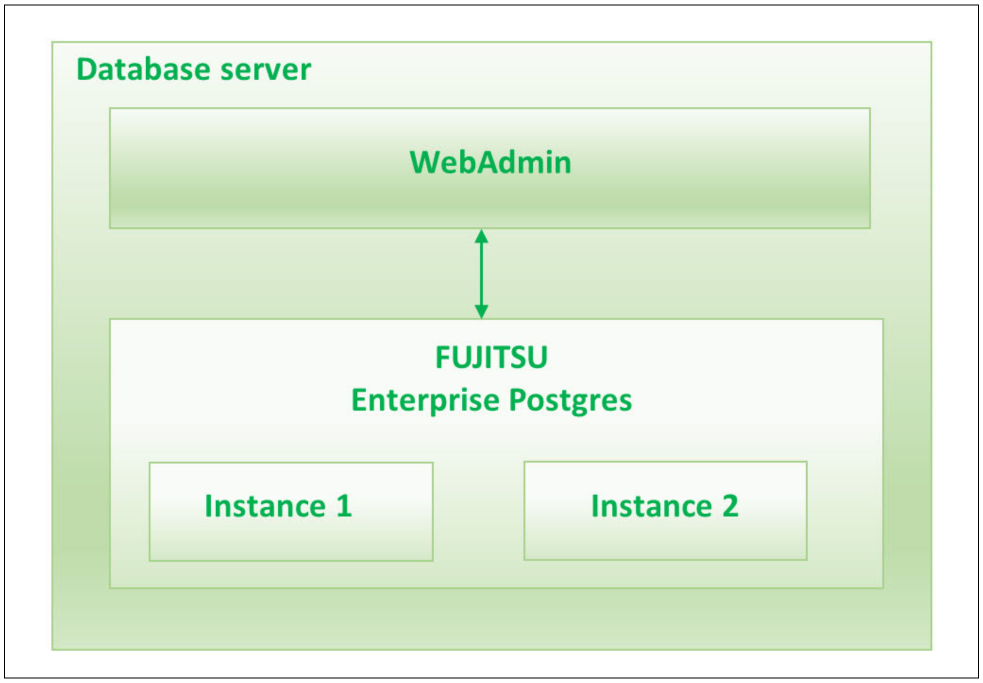


Figure 2-25 WebAdmin single-server configuration

WebAdmin multi-server configuration

A multi-server WedAdmin configuration enables the creation and operation of instances on multiple FUJITSU Enterprise Postgres Database servers. Figure 2-26 on page 75 and Figure 2-27 on page 75 shows two multi-server configurations.

In Figure 2-26, WebAdmin is installed on a dedicated WebAdmin server to collectively manage the instances that are stored on the database servers.

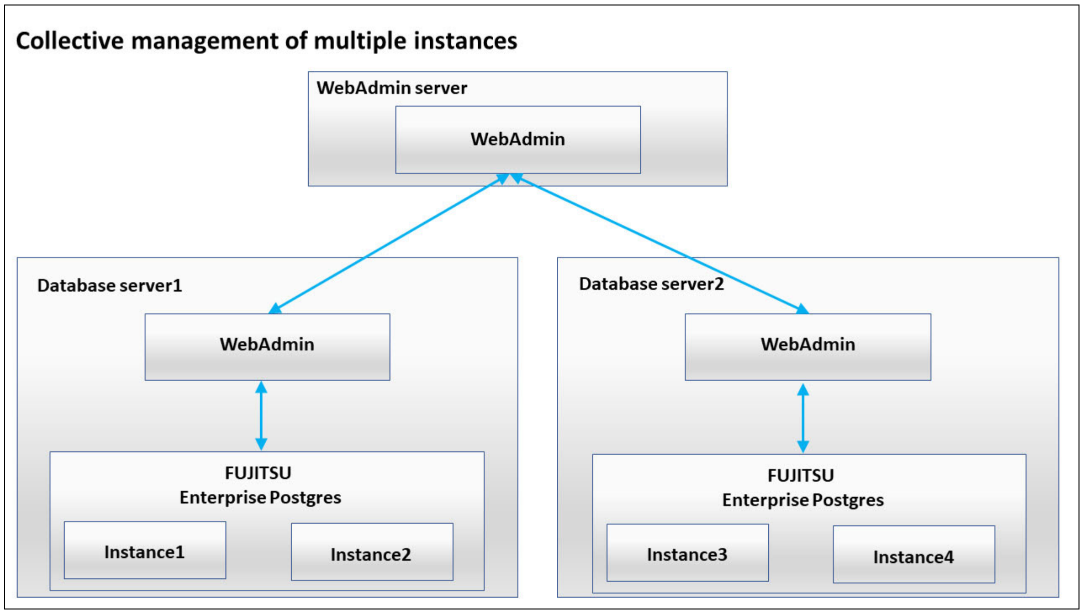


Figure 2-26 WebAdmin multi-server configuration without database multiplexing

Figure 2-27 shows another use case of a WebAdmin multi-server configuration. This configuration uses FUJITSU Enterprise Postgres in data multiplexing mode with Server Assistant on the arbitration server. In this scenario, WebAdmin is installed and configured on the arbitration server.

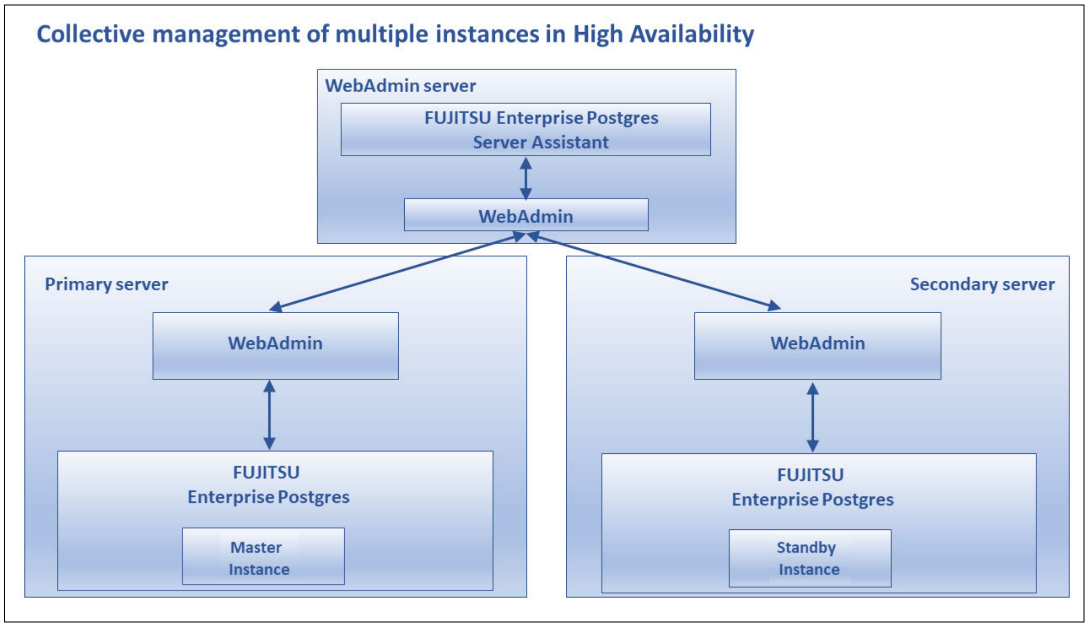

Collective management of multiple instances in High Availability
WebAdmin server
FUJITSU Enterprise Postgres Server Assistant
WebAdmin
Primary server
WebAdmin
FUJITSU Enterprise Postgres
Master Instance
Secondary server
WebAdmin
FUJITSU Enterprise Postgres
Standby Instance

Figure 2-27 WebAdmin multi-server configuration with database multiplexing and high availability

Note: Several `postgresql.conf` parameters can be changed by WebAdmin, and a few parameters can be set by WebAdmin during instance startup. **`shared_buffers`**, **`work_mem`**, **`effective_cache_size`**, and **`maintenance_work_mem`** is automatically set by WebAdmin per the installed memory.

In Chapter 5, "High availability and high reliability architectures" on page 165, we describe use case scenarios of setting up WebAdmin and using it to configure data multiplexing and Server Assistant for HA of FUJITSU Enterprise Postgres.

Note: For more information about WebAdmin installation instructions, see FUJITSU Enterprise Postgres Installation/Setup.

2.5 Application development features

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

- Embedded SQL, Java, and Open Database Connectivity integration
- Oracle compatible features

2.5.1 Embedded SQL, Java, and Open Database Connectivity integration

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

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

In Chapter 7, "Application development, compatibility features, and migration" on page 251, we describe the setup of JDBC, ODBC drivers, examples of using the `libpq` library and compiling a sample code, and embedded SQL in C.

Note: For more information about application development instructions, see *FUJITSU Enterprise Postgres Application Development Guide*.

2.5.2 Oracle compatible features

Table 2-7 on page 77 lists the features that are compatible with Oracle databases. These features ensure that any application that uses an Oracle database also can use FUJITSU Enterprise Postgres.

Table 2-7 Oracle compatible features

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

Note: For more information about the ORAFCE function, see the `README.asciidoc` file found in `fujitsuEnterprisePostgresInstallDir/share/doc/extension/`.

For more information about the Oracle compatibility feature usage and precautions, see *FUJITSU Enterprise Postgres Installation/Setup Guide*.

2.6 FUJITSU Enterprise Postgres supported open-source software peripheral devices

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

The contrib modules that are listed in Table 2-8 are supported by the Fujitsu development team for resolving issues, which means that customers do not need to wait for the community to provide a solution. Instead, customers get enterprise-grade 24x7 support for the database software and the database management tools.

Table 2-8 OSS software peripheral devices that are supported by Fujitsu

Description	OSS name	Version and level
Oracle-compatible SQL features	`Orafce`	3.8.0
Failover, connection pooling, and load balancing	`Pgpool-II`	4.1
Connection to the Oracle database server	`oracle_fdw`	2.2.0
Collection and accumulation of statistics	`pg_statsinfo`	12.0

Description	OSS name	Version and level
Performance tuning (statistics management, and query tuning)	`pg_hint_plan`	12.1.3.5
	`pg_dbms_stats`	1.3.11
Table reorganization	`pg_repack`	1.4.5
Backup and restore management	`pg_rman`	1.3.9
Log analysis	`pgBadger`	11.1
Full-text search (multibyte)	`pg_bigm`	1.2
JDBC driver	`PostgreSQL JDBC driver`	42.2.8
ODBC driver	`psqlODBC`	12.01.0000

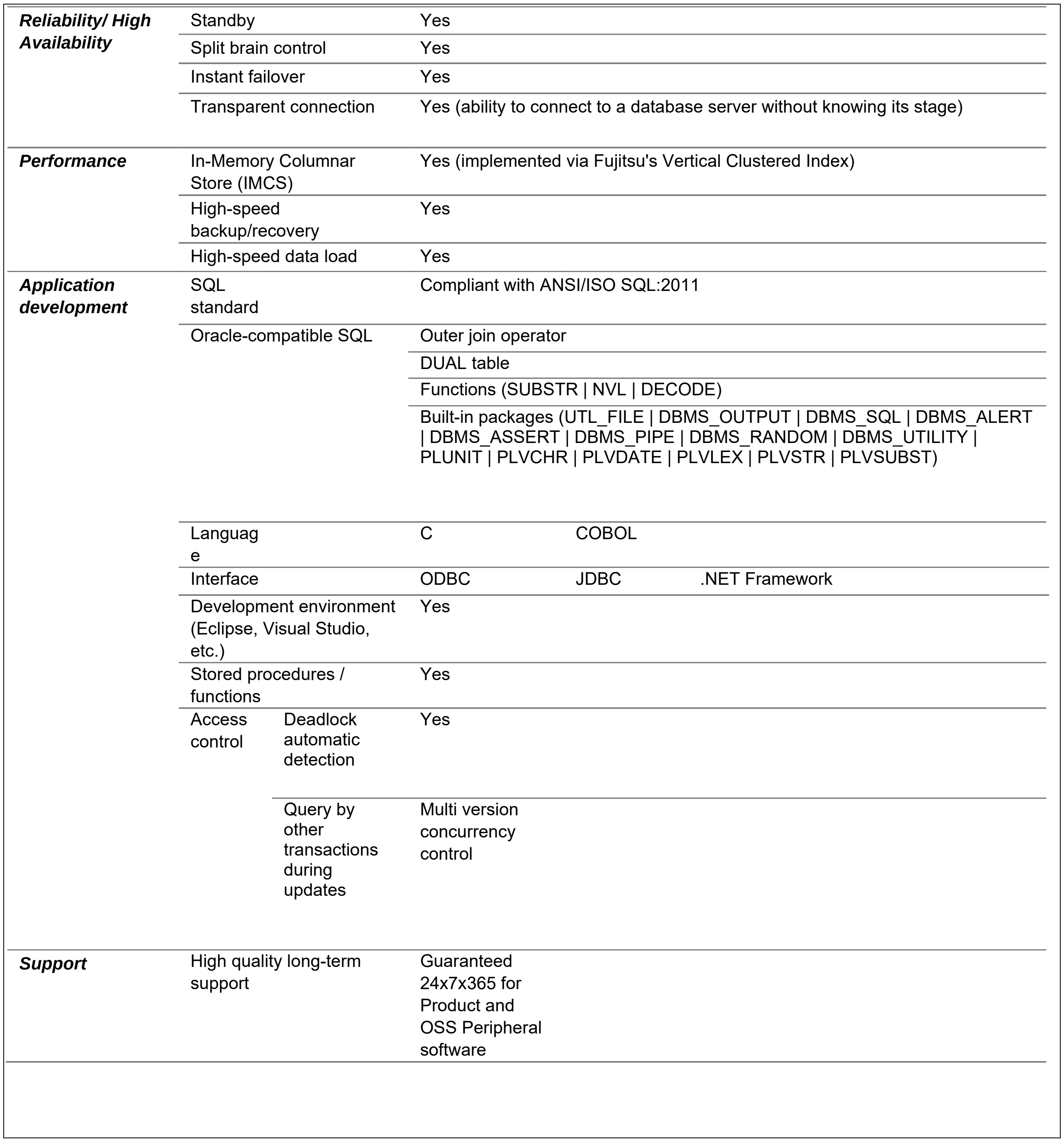
Note: For OSS peripheral tools version information, see the *FUJITSU Enterprise Postgres General Description Guide*.

2.7 FUJITSU Enterprise Postgres data sheet

In this section, we provide the FUJITSU Enterprise Postgres data sheet (Figure 2-28 and Figure 2-29 on page 80).

Item			**FUJITSU Enterprise Postgres**		
Basic architecture	Max. database capacity		Unlimited		
	Max. number of columns in table		1,600		
	Max. row length in table		1.6 TB		
	Max. number of rows in table		Unlimited		
	Max. number of indexes per table		Unlimited		
	Index storage format in table		B-tree	GiST/SP-GiST	
			hash	GIN	
			BRIN	VCI (Fujitsu's In-Memory Columnar Index)	
	Data types	Character types	CHARACTER	NCHAR	
			CHARACTER VARYING	NCHAR VARYING	
			text		
		Numeric types	bigint	integer	smallint
			bigserial	numeric	smallserial
			decimal	real	
			double precision	serial	
		Datetime types	date	time	time with time zone
			interval	timestamp	timestamp with time zone
		Binary data types	bytea		
		XML	Yes		
		JSON	Yes		
	Character set	UNICODE	Yes		
	Multilingual support		Yes (149 locales)		
Security	Transparent data encryption		256-bit PCI-DSS compliant Integration with IBM LinuxONE CryptoCards for hardware-based encryption FIPS 140-2 Level 4 compliant - the highest level of certification achievable for commercial cryptographic devices		
	Data masking		Policy Based Data Masking Full masking / Partial masking / Regular expression masking		
	Dedicated audit log		Yes (compliant with PCI-DSS)		

Figure 2-28 FUJITSU Enterprise Postgres data sheet: 1 of 2



Reliability/ High Availability	Standby		Yes		
	Split brain control		Yes		
	Instant failover		Yes		
	Transparent connection		Yes (ability to connect to a database server without knowing its stage)		
Performance	In-Memory Columnar Store (IMCS)		Yes (implemented via Fujitsu's Vertical Clustered Index)		
	High-speed backup/recovery		Yes		
	High-speed data load		Yes		
Application development	SQL standard		Compliant with ANSI/ISO SQL:2011		
	Oracle-compatible SQL		Outer join operator		
			DUAL table		
			Functions (SUBSTR | NVL | DECODE)		
			Built-in packages (UTL_FILE | DBMS_OUTPUT | DBMS_SQL | DBMS_ALERT | DBMS_ASSERT | DBMS_PIPE | DBMS_RANDOM | DBMS_UTILITY | PLUNIT | PLVCHR | PLVDATE | PLVLEX | PLVSTR | PLVSUBST)		
	Language		C	COBOL	
	Interface		ODBC	JDBC	.NET Framework
	Development environment (Eclipse, Visual Studio, etc.)		Yes		
	Stored procedures / functions		Yes		
	Access control	Deadlock automatic detection	Yes		
		Query by other transactions during updates	Multi version concurrency control		
Support	High quality long-term support		Guaranteed 24x7x365 for Product and OSS Peripheral software		

Figure 2-29 FUJITSU Enterprise Postgres data sheet: 2 of 2

3

Lab environment overview

This chapter describes the configuration of the IBM LinuxONE lab environment that was used for demonstrating FUJITSU Enterprise Postgres 12 use cases in this publication, which includes:

- A FUJITSU Enterprise Postgres Database that was installed on IBM LinuxONE in the following ways:
  - Stand-alone IBM z/VM virtual servers running SUSE Enterprise Linux Server 12 SP4 and Red Hat Enterprise Linux Server 8.2 operating systems (OSs).
  - High availability (HA) configurations of the same.
  - Stand-alone and HA configurations, such as Red Hat OpenShift operators. This configuration leveraged Red Hat CoreOS.
- Implementing backups for FUJITSU Enterprise Postgres Database by using native database tools and with IBM Spectrum Protect.
- FUJITSU Enterprise Postgres integration with IBM LinuxONE hardware security module (HSM) for Transparent Data Encryption (TDE).
- Steps to install a stand-alone FUJITSU Enterprise Postgres 12 server on Red Hat Enterprise Linux Server 8 and SUSE Linux Enterprise Servers 12 SP4.

This chapter covers the following topics:

- Overview
- FUJITSU Enterprise Postgres 12 server installation on LinuxONE
- Installing and configuring a stand-alone database instance on a Red Hat Enterprise Linux 8 guest
- Installing and configuring a stand-alone database instance on a SUSE Linux Enterprise Server 12 Linux guest

3.1 Overview

The lab environment that was configured for this publication included 22 Linux guests on four z/VM logical partitions (LPARs) running on two different types of IBM LinuxONE servers.

LPARs ARIES32, ARIES33, and ARIES34 were z/VM LPARs running on an IBM LinuxONE III multi-frame server. LPAR LEPUS23 was a z/VM LPAR running on an IBM LinuxONE Rockhopper II server. All four LPARs were running z/VM 7.1 as the first-level OS and hypervisor.

Across ARIES32, ARIES34, and LEPUS23, eight Linux guests were installed with Red Hat Enterprise Linux 8.2 and nine Linux guests were installed with SUSE Enterprise Linux Server 12 SP4. ARIES 33 consisted of seven Linux guests that were dedicated to hosting the Red Hat OpenShift environments that were configured with Red Hat CoreOS.

LPAR LEPUS23 was dedicated to hosting the management functions that are required for the various use cases, such as the Client App server, which was used to emulate client traffic to any of the FUJITSU Enterprise Postgres Database as needed.

Regarding HA and disaster recovery (DR) use cases, it was important to ensure that the lab environment delivered a valid and easily referenced HA configuration. The hardware partitioning capabilities that use PR/SM and z/VM deliver Evaluation Assurance Level (EAL) 5+ isolation to the virtual servers on an IBM LinuxONE.

EAL5+ isolation is equivalent to two physically separate mid-range or commodity servers that are deployed in independent racks. Viewed from the Linux OS, each guest can be treated as a separate physical serve with its own processor, memory, storage, and network resources. This isolation enabled the Lab environment to emulate FUJITSU Enterprise Postgres 12 servers being deployed on multiple IBM LinuxONE servers and across multiple data centers.

Figure 3-1 on page 83 demonstrates the layout for all guests that were deployed for this publication on different LPARs and IBM LinuxONE servers.

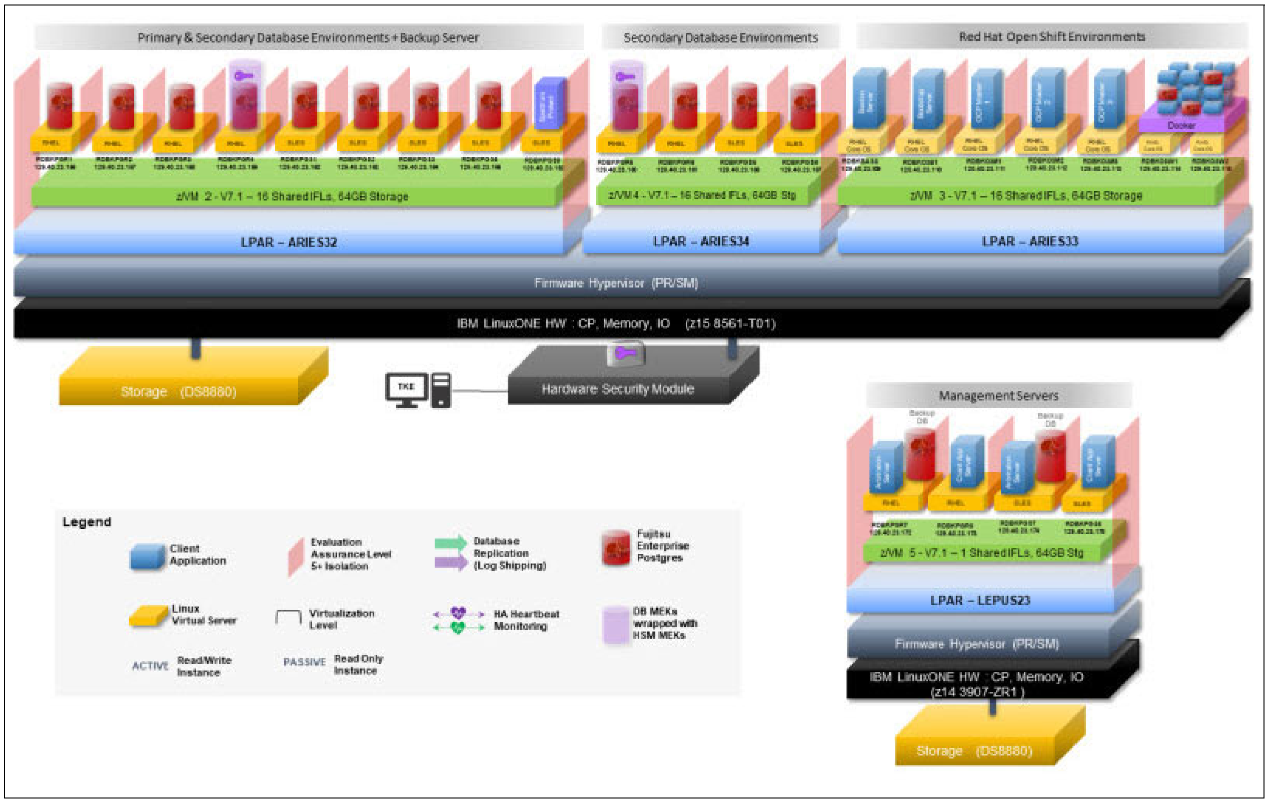

Primary & Secondary Database Environments + Backup Server
Secondary Database Environments
Red Hat Open Shift Environments
Docker
z/VM 2 - V7.1 – 16 Shared IFLs, 64GB Storage
z/VM 4 - V7.1 – 16 Shared IFLs, 64GB Stg
z/VM 3 - V7.1 – 16 Shared IFLs, 64GB Storage
LPAR – ARIES32
LPAR – ARIES34
LPAR – ARIES33
Firmware Hypervisor (PR/SM)
IBM LinuxONE HW : CP, Memory, IO (z15 8561-T01)
Storage (DS8880)
TKE
Hardware Security Module
Management Servers
Backup DB
Backup DB
z/VM 5 - V7.1 – 1 Shared IFLs, 64GB Stg
LPAR – LEPUS23
Firmware Hypervisor (PR/SM)
IBM LinuxONE HW : CP, Memory, IO (z14 3907-ZR1)
Storage (DS8880)
Legend
Client Application
Evaluation Assurance Level 5+ Isolation
Database Replication (Log Shipping)
Fujitsu Enterprise Postgres
Linux Virtual Server
Virtualization Level
HA Heartbeat Monitoring
DB MEKs wrapped with HSM MEKs
ACTIVE
Read/Write Instance
PASSIVE
Read Only Instance

Figure 3-1 Linux guests layout

3.1.1 High availability lab environment for FUJITSU Enterprise Postgres 12 on Red Hat Enterprise Linux 8

Five Red Hat Enterprise Linux 8 guests were used to establish a HA lab for the FUJITSU Enterprise Postgres 12 databases, with a primary server running on LPAR ARIES32, and a secondary server running on LPAR ARIES34 to emulate two physically separate servers that used EAL5+ isolation.

The Arbitration Server that is used in maintaining HA database configurations must be deployed on an independent LPAR from that of both the primary and secondary databases to prevent a split-brain scenario, which is described in Chapter 5, “High availability and high reliability architectures” on page 165. The Arbitration Server was deployed on LEPUS23 on the second IBM LinuxONE server.

All use cases covering FUJITSU Enterprise Postgres 12 server's HA, disaster preparedness, backup and recovery, and replication modes with Red Hat Enterprise Linux 8 were set up and run within the lab environment that is shown in Figure 3-2.

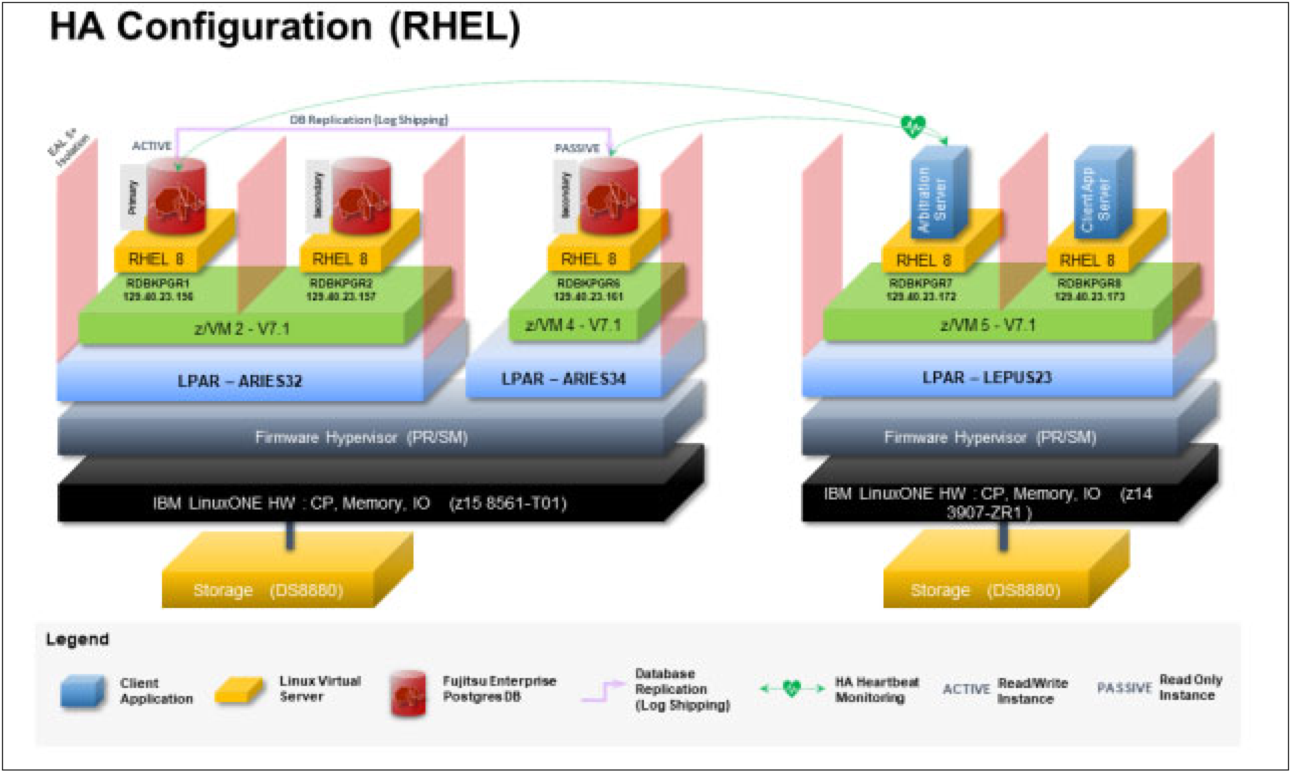


Figure 3-2 High availability lab environment with FUJITSU Enterprise Postgres 12 on Red Hat Enterprise Linux 8

Another secondary database server was also deployed on LPAR ARIES32 to use in horizontal scaling use cases.

3.1.2 High availability lab environment for FUJITSU Enterprise Postgres 12 on SUSE Linux Enterprise Server 12 SP4

Five SUSE Linux Enterprise Server 12 SP4 guests were used to establish a HA lab for FUJITSU Enterprise Postgres 12 with a primary server running on LPAR ARIES32 and a secondary server running on LPAR ARIES34 to emulate two physically separate servers that used EAL5+ isolation.

The Arbitration Server that was used in maintaining HA database configurations must be deployed on an independent LPAR from that of both the primary and secondary databases to prevent a split-brain scenario, which is described in Chapter 5, “High availability and high reliability architectures” on page 165. The Arbitration Server was deployed on LEPUS23 on the second IBM LinuxONE server.

All use cases covering FUJITSU Enterprise Postgres 12 server's HA, disaster preparedness, backup and recovery, and replication modes with SUSE Linux Enterprise Server 12 SP4 were set up and run within the lab environment that is shown in Figure 3-3 on page 85.

Another secondary database server was also deployed on LPAR ARIES32 for use in horizontal scaling use cases.

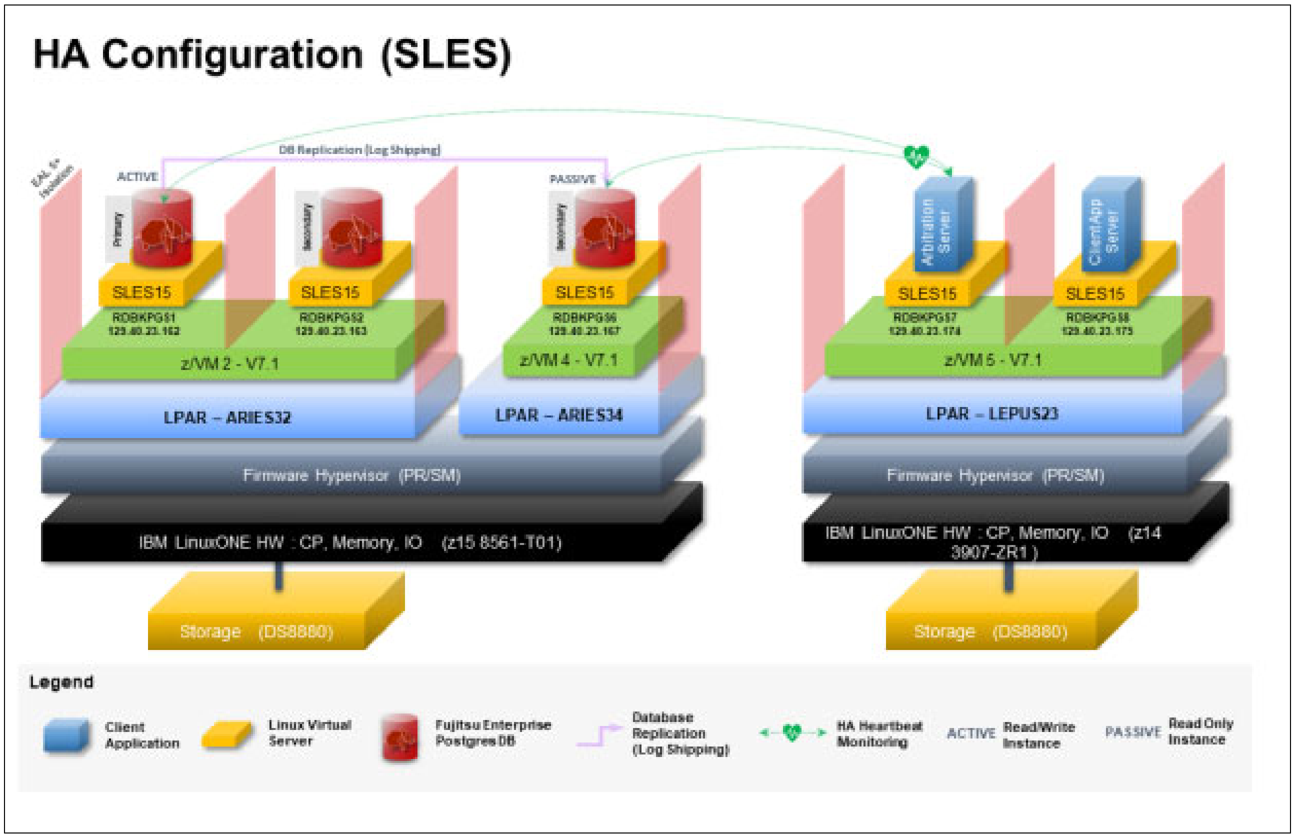


Figure 3-3 High availability lab environment with FUJITSU Enterprise Postgres 12 on SUSE Linux Enterprise Server 12 SP4

For more information about the HA configurations that were used and supported on IBM LinuxONE, see Chapter 5, "High availability and high reliability architectures" on page 165.

3.1.3 Security feature lab environment for FUJITSU Enterprise Postgres 12 on Red Hat Enterprise Linux 8

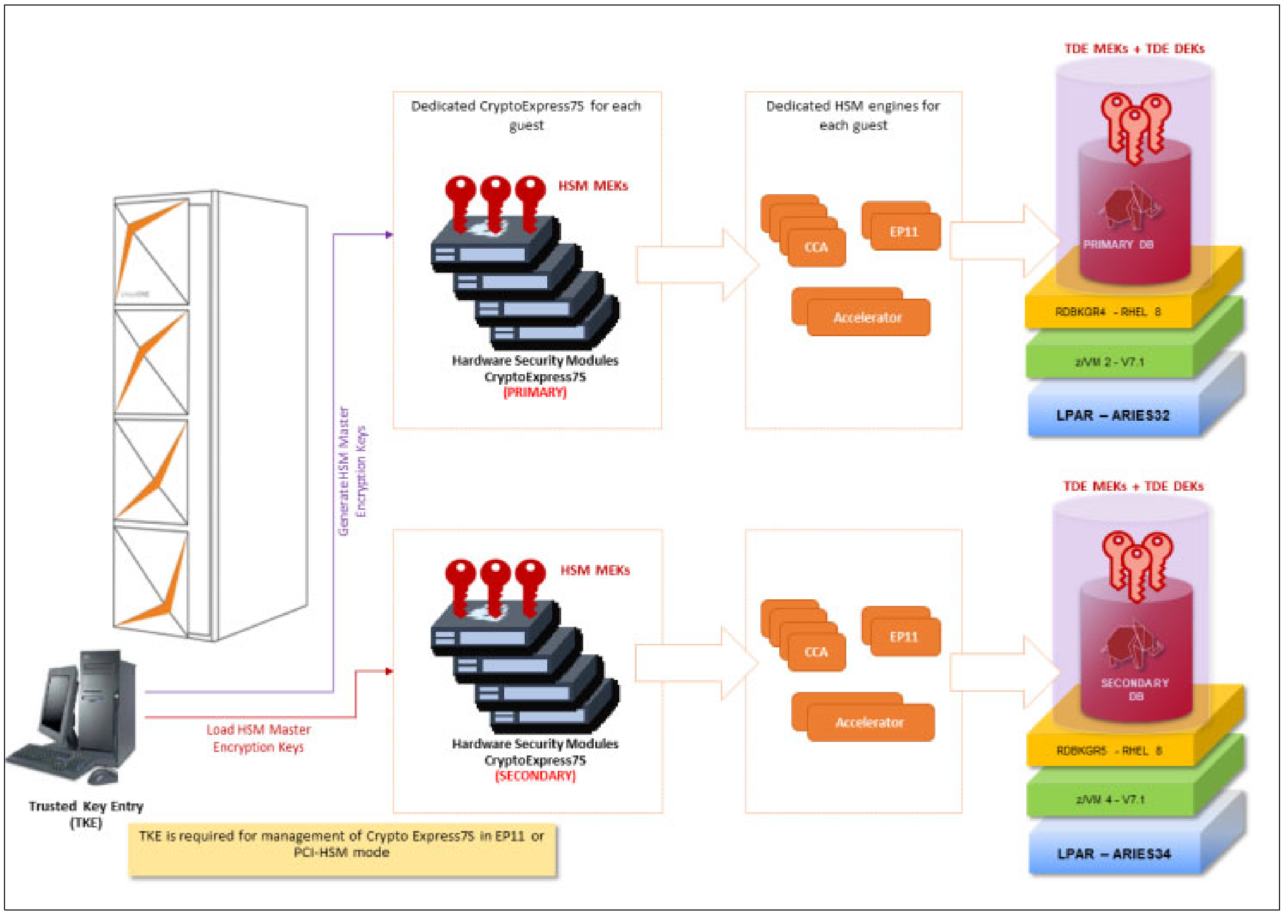
Two Red Hat Enterprise Linux 8 guests with dedicated HSMs were used to establish a security feature lab environment for FUJITSU Enterprise Postgres 12 servers, with a primary server running on LPAR ARIES32 and a secondary server running on LPAR ARIES34.

Four CryptoExpress7S cards were dedicated to each Red Hat Enterprise Linux 8 guest. Each CryptoExpress7S card can be customized as two separated HSM engines with different types. In our lab environment, the four dedicated cards were configured as two EP11, four Common Cryptographic Architecture (CCA), and two accelerator engines, for each guest.

The use cases for FUJITSU Enterprise Postgres 12 server's HSM-related security features with Red Hat Enterprise Linux 8 were set up and run on the lab environment that is shown in Figure 3-4 on page 86.

Although the HSM Accelerator engine was configured in this environment, FUJITSU Enterprise Postgres does not support or leverage this feature. The Accelerator is required for other functions that are performed at the z/VM and OS level, such as secure networking (TLS and Secure Sockets Layer (SSL)), and secure directory services connectivity.

Note: Because there is no material difference between a Red Hat Enterprise Linux server guest and a SUSE Linux guest when it comes to configuring and using the HSM, the lab environment and security use cases that are covered in this book refer only to Red Hat Enterprise Linux virtual servers.



Dedicated CryptoExpress7S for each guest
Dedicated HSM engines for each guest
TDE MEKs + TDE DEKs
HSM MEKs
CCA
EP11
Accelerator
PRIMARY DB
RDBKGR4 - RHEL 8
z/VM 2 - V7.1
LPAR – ARIES32
Hardware Security Modules CryptoExpress7S (PRIMARY)
Generate HSM Master Encryption Keys
Load HSM Master Encryption Keys
Hardware Security Modules CryptoExpress7S (SECONDARY)
SECONDARY DB
RDBKGR5 - RHEL 8
z/VM 4 - V7.1
LPAR – ARIES34
Trusted Key Entry (TKE)
TKE is required for management of Crypto Express7S in EP11 or PCI-HSM mode

Figure 3-4 Security feature lab environment for FUJITSU Enterprise Postgres 12 on Red Hat Enterprise Linux 8

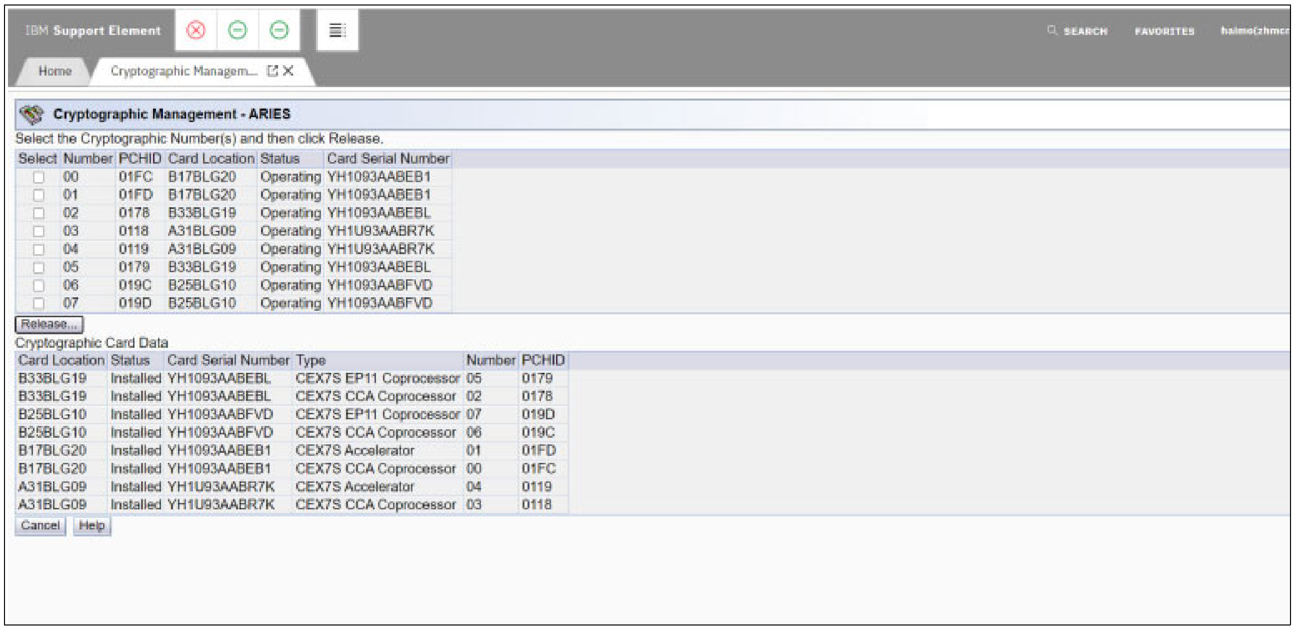
In addition, five smart cards were used to load the HSM master keys for the EP11 certificate authority (CA), EP11 Admin, and the Trusted Key Entry (TKE) Administrator, with each smart card configured as follows:

- EP11 CA.
- EP11 CA backup (optional but recommended).
- EP11 Admin 1.
- EP11 Admin 2.
- TKE Administrator: Initially set up when the TKE was first used for generating CCA keys. Although only a single card was used for the lab environment, a backup card is recommended for client environments.

In a client environment, the number of smart cards that are used depends on the number of HSM engines that are required for HA in production/DR environments, and the requisite number of independent key parts for organizational security, for example, the minimum number of authorized individuals that are required to create/update MEKs on the HSM.

CryptoExpress7S feature configuration

Using the Cryptographic Management task on the IBM LinuxONE III Support Element (SE) GUI, a list of CryptoExpress feature cards that are installed on the IBM LinuxONE server with their configuration data is shown in Figure 3-5.



Cryptographic Management - ARIES

Select the Cryptographic Number(s) and then click Release.

Select	Number	PCHID	Card Location	Status	Card Serial Number
☐	00	01FC	B17BLG20	Operating	YH1093AABEB1
☐	01	01FD	B17BLG20	Operating	YH1093AABEB1
☐	02	0178	B33BLG19	Operating	YH1093AABEBL
☐	03	0118	A31BLG09	Operating	YH1U93AABR7K
☐	04	0119	A31BLG09	Operating	YH1U93AABR7K
☐	05	0179	B33BLG19	Operating	YH1093AABEBL
☐	06	019C	B25BLG10	Operating	YH1093AABFVD
☐	07	019D	B25BLG10	Operating	YH1093AABFVD

Release...

Cryptographic Card Data

Card Location	Status	Card Serial Number	Type	Number	PCHID
B33BLG19	Installed	YH1093AABEBL	CEX7S EP11 Coprocessor	05	0179
B33BLG19	Installed	YH1093AABEBL	CEX7S CCA Coprocessor	02	0178
B25BLG10	Installed	YH1093AABFVD	CEX7S EP11 Coprocessor	07	019D
B25BLG10	Installed	YH1093AABFVD	CEX7S CCA Coprocessor	06	019C
B17BLG20	Installed	YH1093AABEB1	CEX7S Accelerator	01	01FD
B17BLG20	Installed	YH1093AABEB1	CEX7S CCA Coprocessor	00	01FC
A31BLG09	Installed	YH1U93AABR7K	CEX7S Accelerator	04	0119
A31BLG09	Installed	YH1U93AABR7K	CEX7S CCA Coprocessor	03	0118

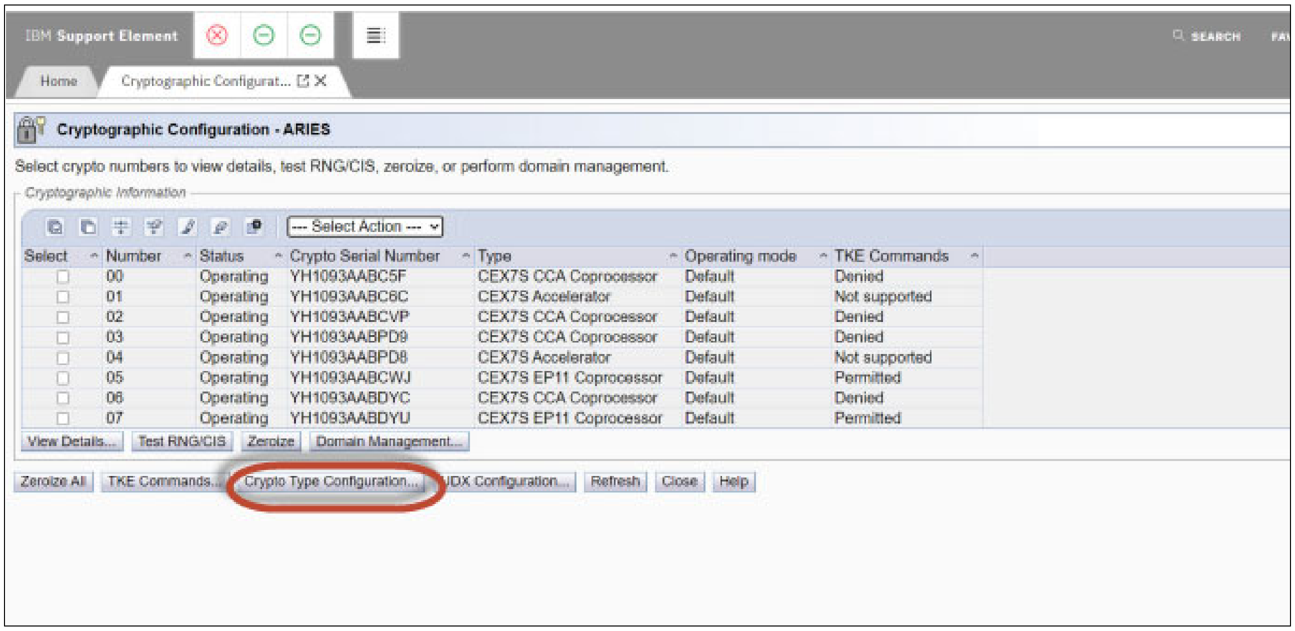
Cancel Help

Figure 3-5 IBM LinuxONE III cryptographic management task GUI

Each row in the Cryptographic Card Data table shows a CryptoExpress coprocessor with a unique value for the physical channel identifier (PCHID) and number. Also shown are the configured operation types for each coprocessor. For example, the type of coprocessor whose number is '02 is configured as a CCA coprocessor.

The relationship between coprocessors and physical feature cards is indicated by the Card Location and Card Serial Number, where the same location and serial number indicate the same physical feature card. For example, coprocessors numbers 06 and 07 are on the same physical CryptoExpress7S card in B25BLG10, with the serial number YJ1093AABFVD.

When using the Cryptographic Configuration GUI task, the type of coprocessor can be changed by selecting the corresponding coprocessor from the list, and then clicking **Crypto Type Configuration**, as shown in Figure 3-6.



Cryptographic Configuration - ARIES

Select crypto numbers to view details, test RNG/CIS, zeroize, or perform domain management.

Cryptographic Information

Select	Number	Status	Crypto Serial Number	Type	Operating mode	TKE Commands
☐	00	Operating	YH1093AABC5F	CEX7S CCA Coprocessor	Default	Denied
☐	01	Operating	YH1093AABC6C	CEX7S Accelerator	Default	Not supported
☐	02	Operating	YH1093AABCVP	CEX7S CCA Coprocessor	Default	Denied
☐	03	Operating	YH1093AABPD9	CEX7S CCA Coprocessor	Default	Denied
☐	04	Operating	YH1093AABPD8	CEX7S Accelerator	Default	Not supported
☐	05	Operating	YH1093AABCWJ	CEX7S EP11 Coprocessor	Default	Permitted
☐	06	Operating	YH1093AABDYC	CEX7S CCA Coprocessor	Default	Denied
☐	07	Operating	YH1093AABDYU	CEX7S EP11 Coprocessor	Default	Permitted

Figure 3-6 Cryptographic configuration GUI task

The correct configuration of coprocessor types for installed physical feature cards can achieve high resiliency, availability, and serviceability (RAS) at the hardware component level.

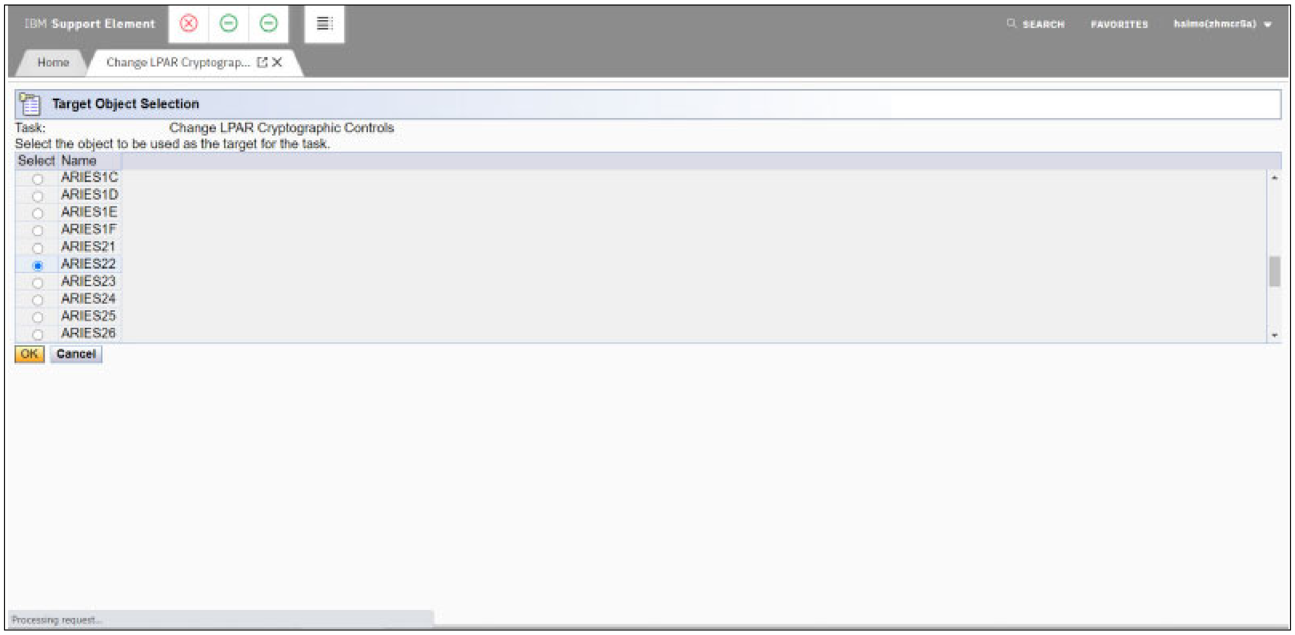
To showcase this RAS, four physical CryptoExpress7S cards from different locations, each with two coprocessors configured as different types, were selected for the lab environment. This level of redundant configuration ensured that an unplanned outage of any one physical card did not lead to a complete loss of any type of coprocessors.

For example, in the event of a loss of the CryptoExpress7S card at B17BLG20, the lab environment loses access to CCA coprocessor number 00 and accelerator number 01, but still has CCA coprocessor number 03 and accelerator number 04.

CryptoExpress card allocation to logical partitions

Note: During the initial build of our lab environment, some HSM configuration screen captures for LPARs ARIES32 and ARIES34 were not captured. As a result, a new LPAR ARIES22 was configured specifically to capture screen captures.

Assignment of the CryptoExpress cards to specific LPARs is initiated by clicking **Change LPAR Cryptographic Controls** task from the IBM LinuxONE III SE, selecting the LPAR (ARIES22) from the target object selection list, and clicking **OK**, as shown in Figure 3-7 on page 89.


IBM Support Element

Home | Change LPAR Cryptograp...

Target Object Selection

Task: Change LPAR Cryptographic Controls

Select the object to be used as the target for the task.

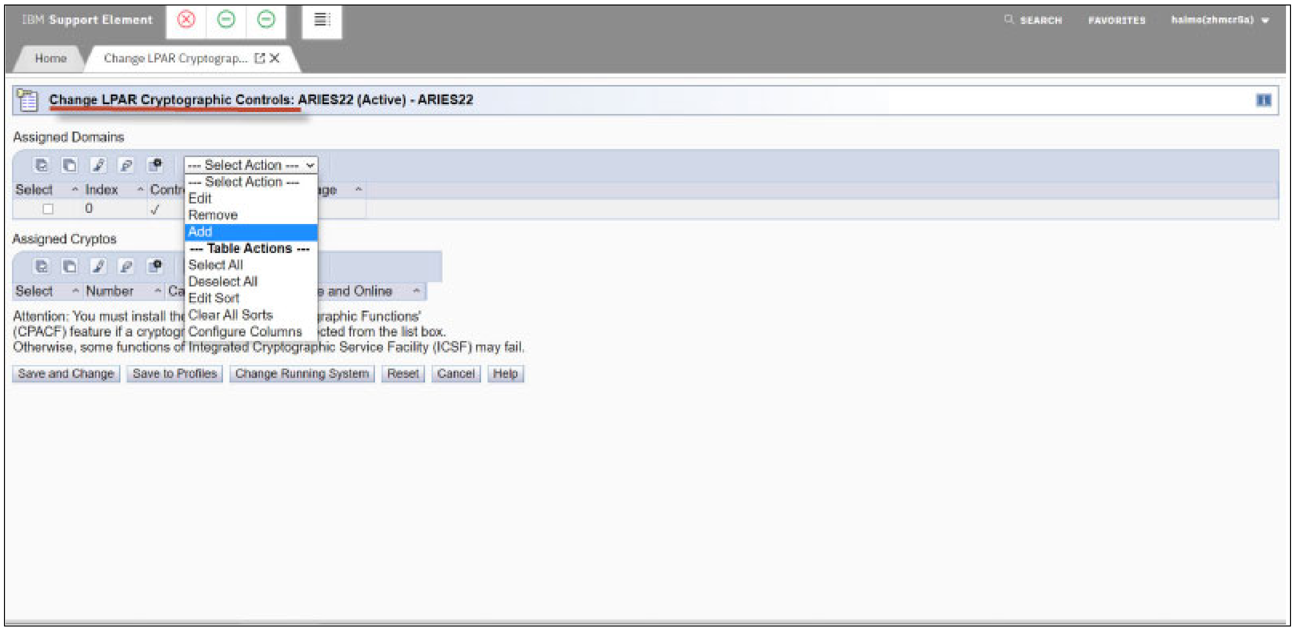
Select	Name
○	ARIES1C
○	ARIES1D
○	ARIES1E
○	ARIES1F
○	ARIES21
●	ARIES22
○	ARIES23
○	ARIES24
○	ARIES25
○	ARIES26

OK | Cancel

Processing request...

Figure 3-7 Change LPAR Cryptographic Controls view

The Change LPAR Cryptographic Controls: ARIES22(Active) - ARIES2 opens (Figure 3-8). The first step to assigning HSMs to an LPAR is to configure cryptographic domains to the LPAR. Add a control or control and usage domain to the LPAR by selecting **Assigned Domains** → **Select Actions**.


IBM Support Element

Home | Change LPAR Cryptograp...

Change LPAR Cryptographic Controls: ARIES22 (Active) - ARIES22

Assigned Domains

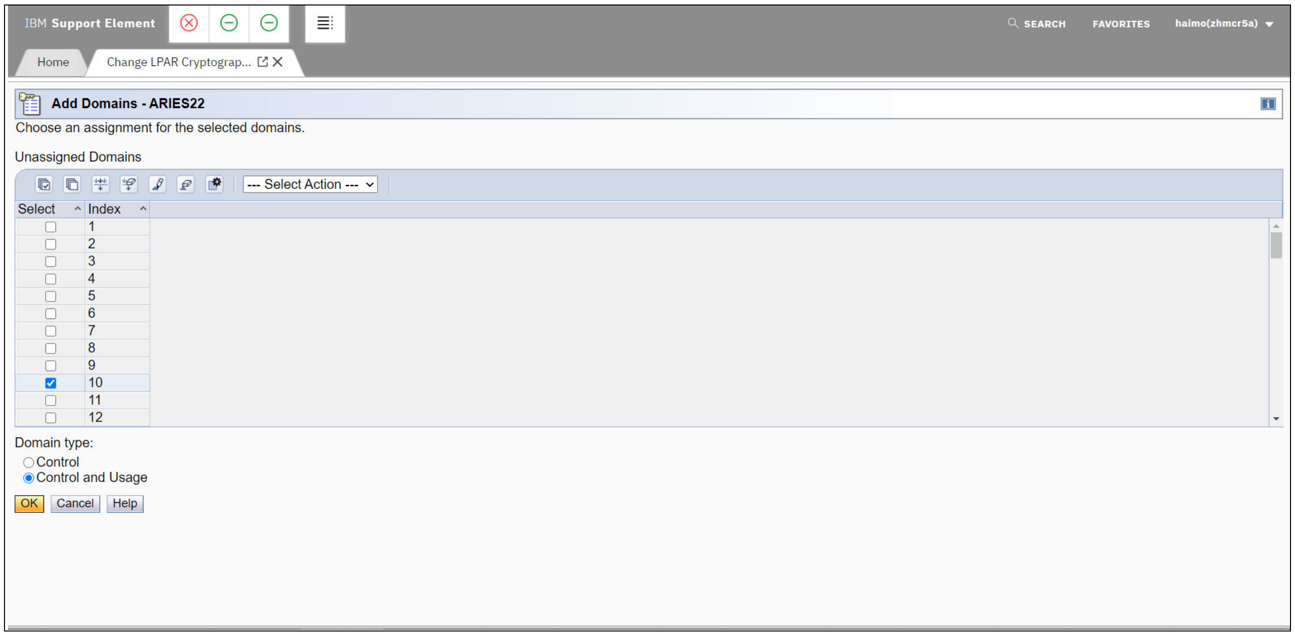
Assigned Cryptos

Attention: You must install the raphic Functions' (CPACF) feature if a cryptogr cted from the list box. Otherwise, some functions of Integrated Cryptographic Service Facility (ICSF) may fail.

Save and Change | Save to Profiles | Change Running System | Reset | Cancel | Help

Figure 3-8 Adding a domain to an LPAR view

Select **Domain Type** → **Control and Usage** to change Domain 10 to one of the usage domains of LPAR ARIES22 for cryptographic functions (Figure 3-9). An LPAR's control domains are the cryptographic domains for which remote secure administration functions can be established and administered from the LPAR.



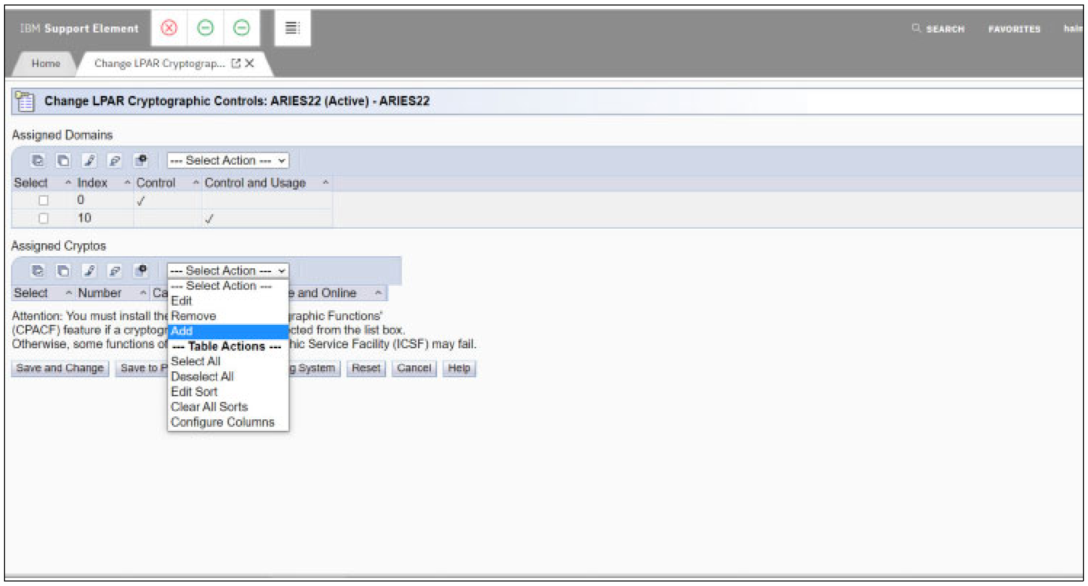
Choose an assignment for the selected domains.

Select	Index
☐	1
☐	2
☐	3
☐	4
☐	5
☐	6
☐	7
☐	8
☐	9
✓	10
☐	11
☐	12

Domain type:
Control [✓ Control and Usage]

Figure 3-9   Adding a domain to an LPAR

Assign coprocessors to the LPAR by selecting **Select Actions** in the Assigned Cryptos pane in the Change LPAR Cryptographic Controls: ARIES22(Active) - ARIES22 window (Figure 3-10).

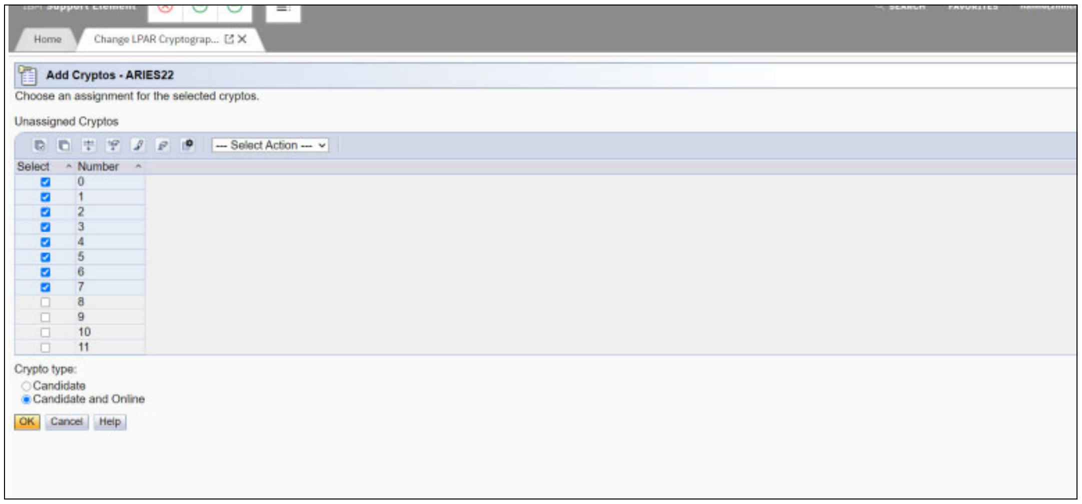


Select	Index	Control	Control and Usage
☐	0	✓	
☐	10		✓

Attention: You must install the raphic Functions'
(CPACF) feature if a cryptogr cted from the list box.
Otherwise, some functions of hic Service Facility (ICSF) may fail.

Figure 3-10   Adding crypto to LPAR-1

As shown in Figure 3-11, cryptos 0 - 7 were added to LPAR ARIES22 by selecting **Candidate and Online** in the Crypto type pane. Those crypto coprocessors will be assigned and brought online at the next activation of the LPAR. The Candidate option indicates that those crypto coprocessors will be assigned to the LPAR but must be activated manually when needed (for example, spare cards).



Add Cryptos - ARIES22

Choose an assignment for the selected cryptos.

Unassigned Cryptos

Select	Number
✓	0
✓	1
✓	2
✓	3
✓	4
✓	5
✓	6
✓	7
	8
	9
	10
	11

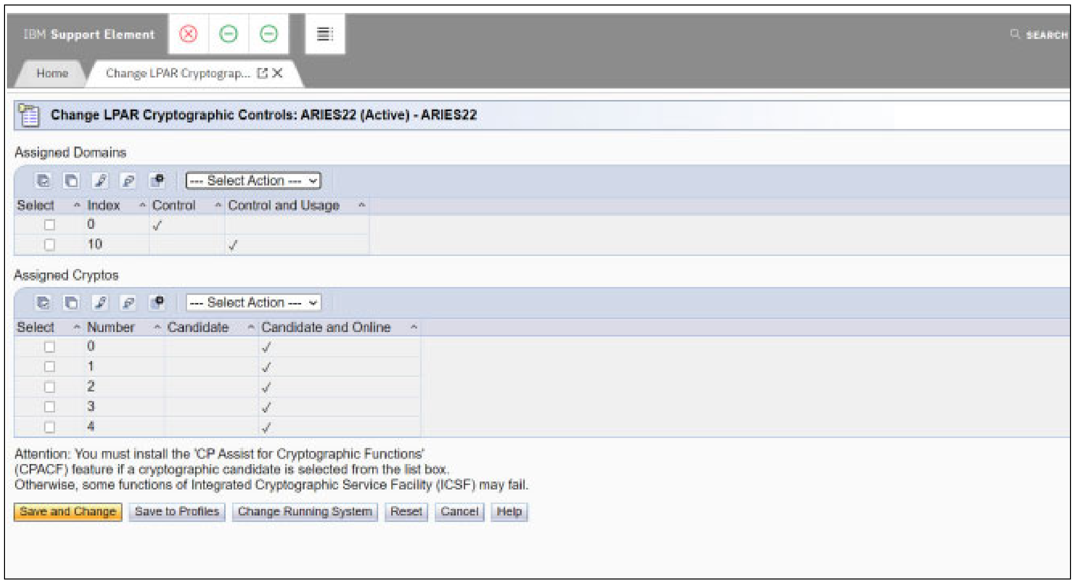
Crypto type:
Candidate [✓ Candidate and Online]

Figure 3-11 Adding crypto to LPAR-2

In the Change LPAR Cryptographic Controls: ARIES22(Active) - ARIES22 window, you can review the current assigned domains and crypto coprocessors to LPAR ARIES22 and modify the assignments as needed by clicking **Select Action**, and then clicking **Edit**, **Remove**, or **Add** under the Assigned Domains and Assigned Cryptos panes.

Click **Save to Profiles** to save any changes to the LPAR's profile, with the changes coming into effect when the LPAR is next activated or restarted. Clicking **Change Running System** applies the changes immediately without saving them to the LPAR's profile, which means these changes are when the LPAR is next activated or restarted.

For this environment, we clicked **Save and Change** as shown in Figure 3-12 to save the changes in LPAR ARIES22's profile and immediately applied the changes.



Change LPAR Cryptographic Controls: ARIES22 (Active) - ARIES22

Assigned Domains

--- Select Action ---

Select	Index	Control	Control and Usage
☐	0	✓	
☐	10		✓

Assigned Cryptos

--- Select Action ---

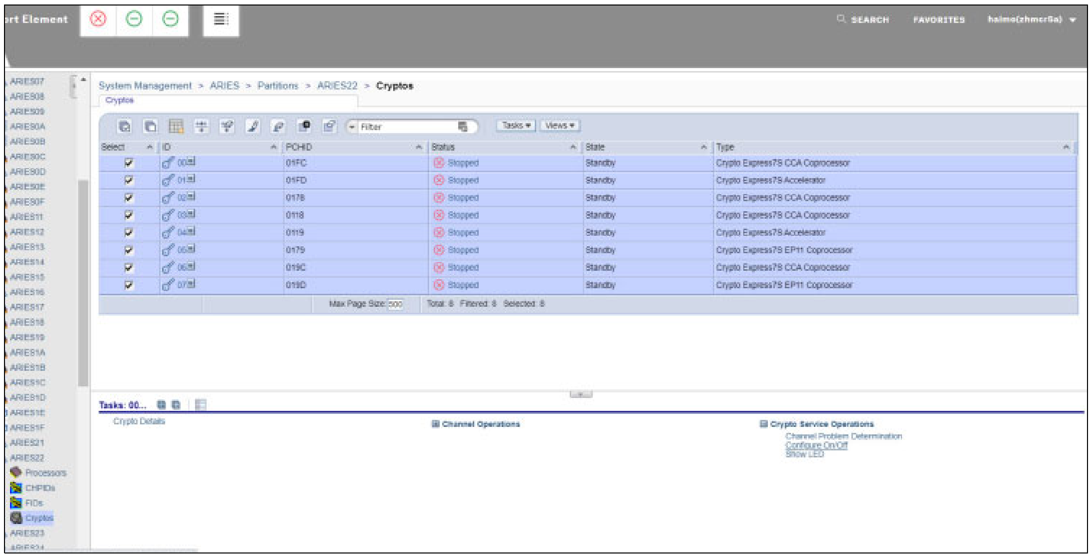
Select	Number	Candidate	Candidate and Online
☐	0		✓
☐	1		✓
☐	2		✓
☐	3		✓
☐	4		✓

Attention: You must install the 'CP Assist for Cryptographic Functions' (CPACF) feature if a cryptographic candidate is selected from the list box. Otherwise, some functions of Integrated Cryptographic Service Facility (ICSF) may fail.

Save and Change | Save to Profiles | Change Running System | Reset | Cancel | Help

Figure 3-12 Save and Change to LPAR cryptographic control: 1 of 2

As shown in Figure 3-13, the newly added crypto coprocessors 00 - 07 were assigned to LPAR ARIES22 with a Stopped status, which means the coprocessors must be brought online manually before the OS can see and use them.



System Management > ARIES > Partitions > ARIES22 > **Cryptos**

Select	ID	PCHID	Status	State	Type
☑	00	01FC	Stopped	Standby	Crypto Express7S CCA Coprocessor
☑	01	01FD	Stopped	Standby	Crypto Express7S Accelerator
☑	02	0178	Stopped	Standby	Crypto Express7S CCA Coprocessor
☑	03	0118	Stopped	Standby	Crypto Express7S CCA Coprocessor
☑	04	0119	Stopped	Standby	Crypto Express7S Accelerator
☑	05	0179	Stopped	Standby	Crypto Express7S EP11 Coprocessor
☑	06	019C	Stopped	Standby	Crypto Express7S CCA Coprocessor
☑	07	019D	Stopped	Standby	Crypto Express7S EP11 Coprocessor

Max Page Size: 500 Total: 8 Filtered: 8 Selected: 8

Tasks: 00...

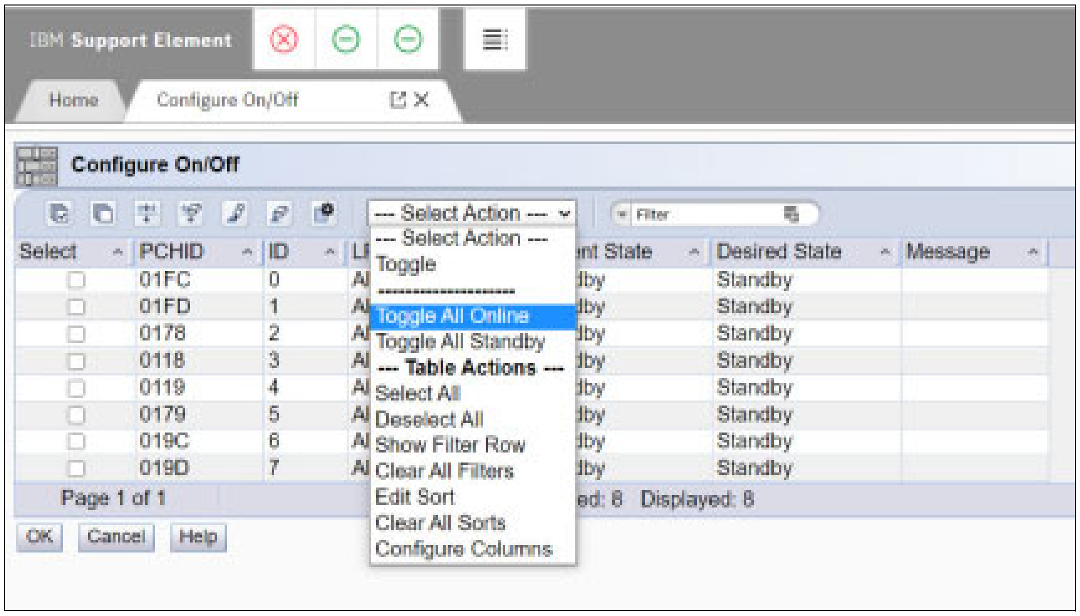
Crypto Details

Channel Operations

Crypto Service Operations
- Channel Problem Determination
- Configure On/Off
- Show LED

Figure 3-13 Save and Change to LPAR cryptographic control: 2 of 2

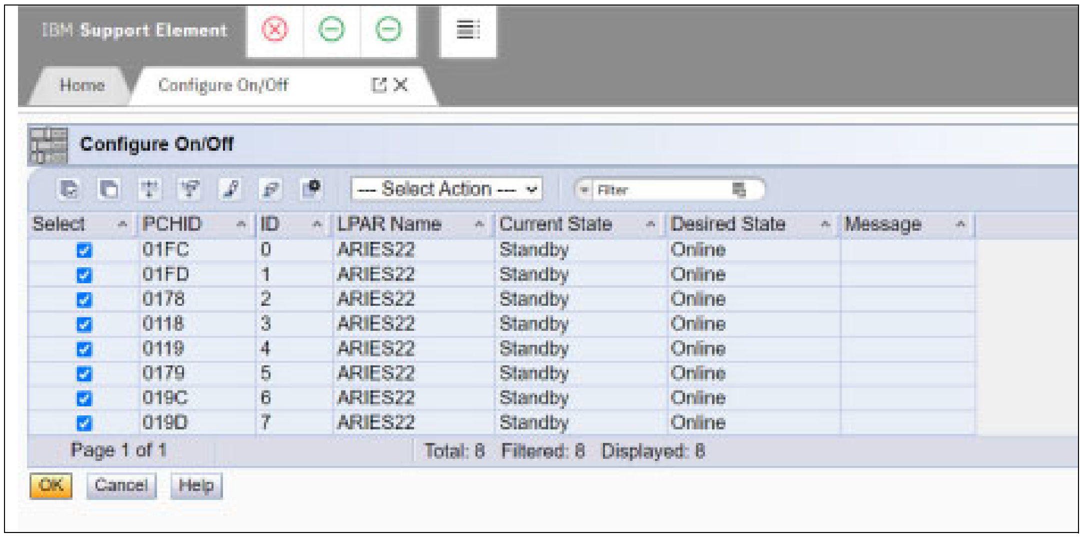
Select all the crypto coprocessors of LPAR ARIES22, click **Configure On/Off** task, and then select **Select Action** → **Toggle All Online** to simultaneously activate all coprocessors, as shown in Figure 3-14 on page 93.



Select	PCHID	ID	LI	...nt State	Desired State	Message
☐	01FC	0	Al	...dby	Standby	
☐	01FD	1	Al	...dby	Standby	
☐	0178	2	Al	...dby	Standby	
☐	0118	3	Al	...dby	Standby	
☐	0119	4	Al	...dby	Standby	
☐	0179	5	Al	...dby	Standby	
☐	019C	6	Al	...dby	Standby	
☐	019D	7	Al	...dby	Standby	

Figure 3-14 Configuring LPAR crypto online: 1 of 2

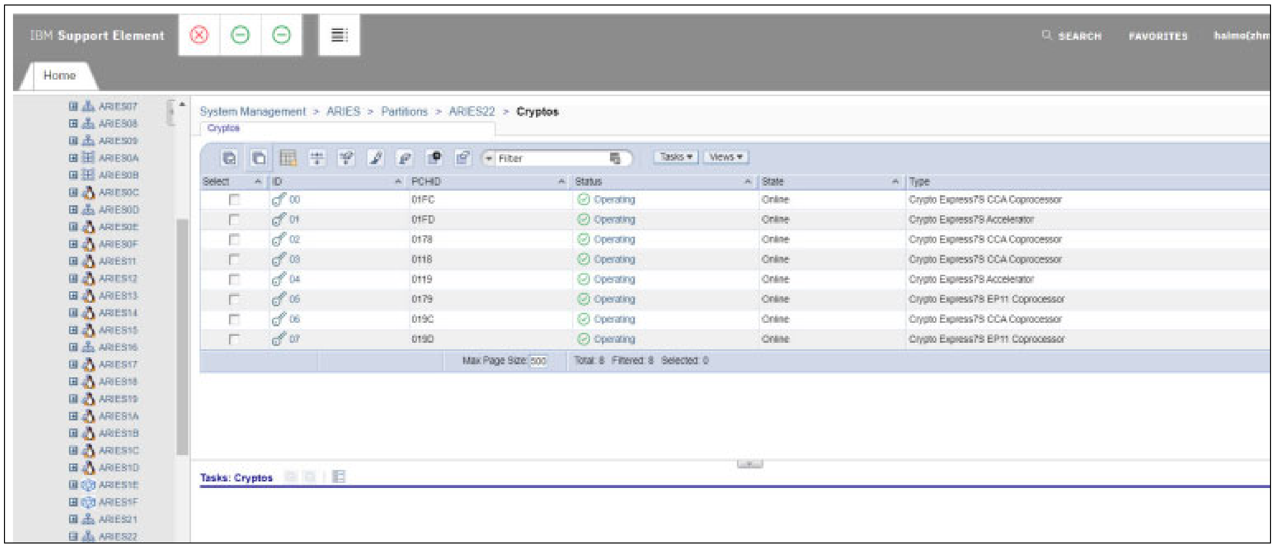
Verify that the crypto ID, LPAR name and Desired Status are correct, as shown in Figure 3-15, and then to click **OK**.



Select	PCHID	ID	LPAR Name	Current State	Desired State	Message
✓	01FC	0	ARIES22	Standby	Online	
✓	01FD	1	ARIES22	Standby	Online	
✓	0178	2	ARIES22	Standby	Online	
✓	0118	3	ARIES22	Standby	Online	
✓	0119	4	ARIES22	Standby	Online	
✓	0179	5	ARIES22	Standby	Online	
✓	019C	6	ARIES22	Standby	Online	
✓	019D	7	ARIES22	Standby	Online	

Figure 3-15 Configuring LPAR crypto online: 2 of 2

Figure 3-16 shows cryptos 00 - 07 of LPAR ARIES22 all showing the Operating status and in the Online state.



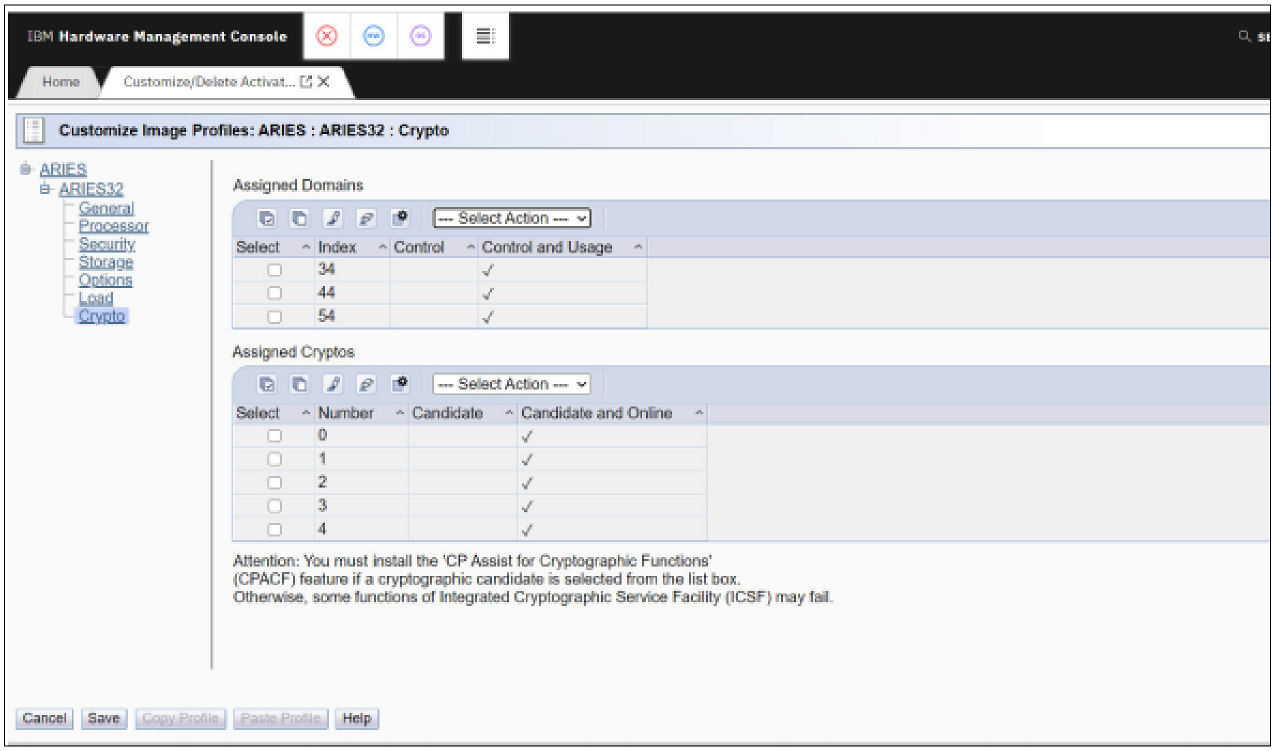
System Management > ARIES > Partitions > ARIES22 > Cryptos

Select	ID	PCHID	Status	State	Type
☐	00	01FC	Operating	Online	Crypto Express7S CCA Coprocessor
☐	01	01FD	Operating	Online	Crypto Express7S Accelerator
☐	02	0178	Operating	Online	Crypto Express7S CCA Coprocessor
☐	03	0118	Operating	Online	Crypto Express7S CCA Coprocessor
☐	04	0119	Operating	Online	Crypto Express7S Accelerator
☐	05	0179	Operating	Online	Crypto Express7S EP11 Coprocessor
☐	06	019C	Operating	Online	Crypto Express7S CCA Coprocessor
☐	07	019D	Operating	Online	Crypto Express7S EP11 Coprocessor

Max Page Size: 500 Total: 8 Filtered: 8 Selected: 0

Figure 3-16 Configuring LPAR crypto online

Figure 3-17 shows an example of the crypto and domain assign settings of LPAR ARIES 32, where cryptos 00 - 07 are assigned, but only 00 - 04 are in use. Domains 34, 44, and 54 are assigned as Control and Usage domains for the LPAR, and domain 34 is dedicated to our RHLE8 instance for the TDE use cases.



Customize Image Profiles: ARIES : ARIES32 : Crypto

Assigned Domains

Select	Index	Control	Control and Usage
☐	34		✓
☐	44		✓
☐	54		✓

Assigned Cryptos

Select	Number	Candidate	Candidate and Online
☐	0		✓
☐	1		✓
☐	2		✓
☐	3		✓
☐	4		✓

Attention: You must install the 'CP Assist for Cryptographic Functions' (CPACF) feature if a cryptographic candidate is selected from the list box. Otherwise, some functions of Integrated Cryptographic Service Facility (ICSF) may fail.

Figure 3-17 Crypto settings of LPAR ARIES32

3.1.4 Red Hat OpenShift lab environment for FUJITSU Enterprise Postgres 12

Seven Red Hat Enterprise Linux CoreOS (RHCOS) guests were set up for the use cases demonstrating FUJITSU Enterprise Postgres 12 databases that were deployed as Red Hat OpenShift Operators with the installation running on LPAR ARIES33 (Figure 3-18). For more information about containerization use cases, see Chapter 10, “FUJITSU Enterprise Postgres cluster on Red Hat OpenShift Container Platform” on page 373.

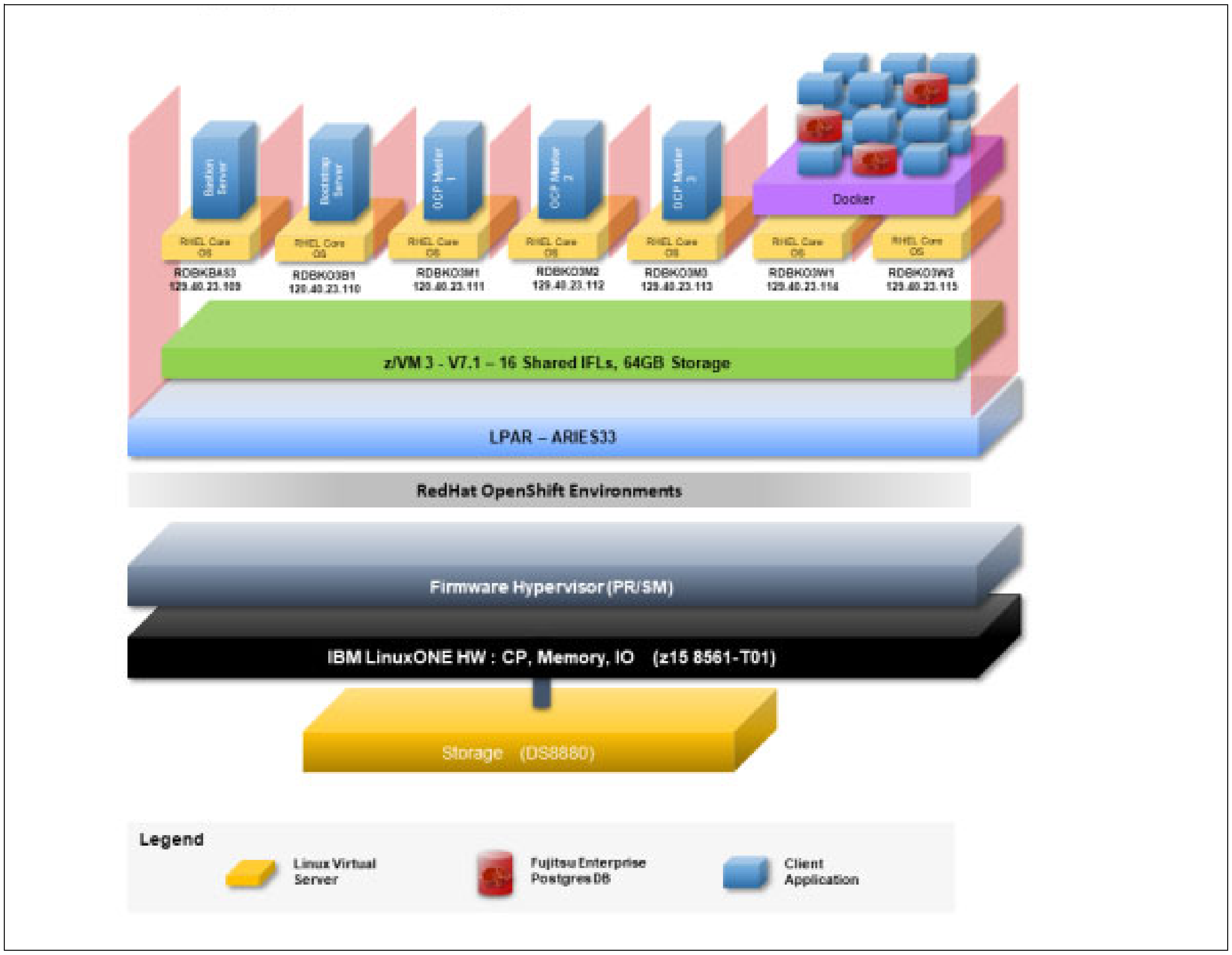


Figure 3-18 Red Hat OpenShift lab with FUJITSU Enterprise Postgres 12 on Red Hat Enterprise Linux CoreOS

3.1.5 Application development feature lab environment with FUJITSU Enterprise Postgres 12

One Red Hat Enterprise Linux 8 guest and one SUSE Linux Enterprise Server 12 SP4 guest is set up for use cases demonstrating FUJITSU Enterprise Postgres 12 server's application development features and installation running on LPAR ARIES32. Our environment is shown in Figure 3-19.

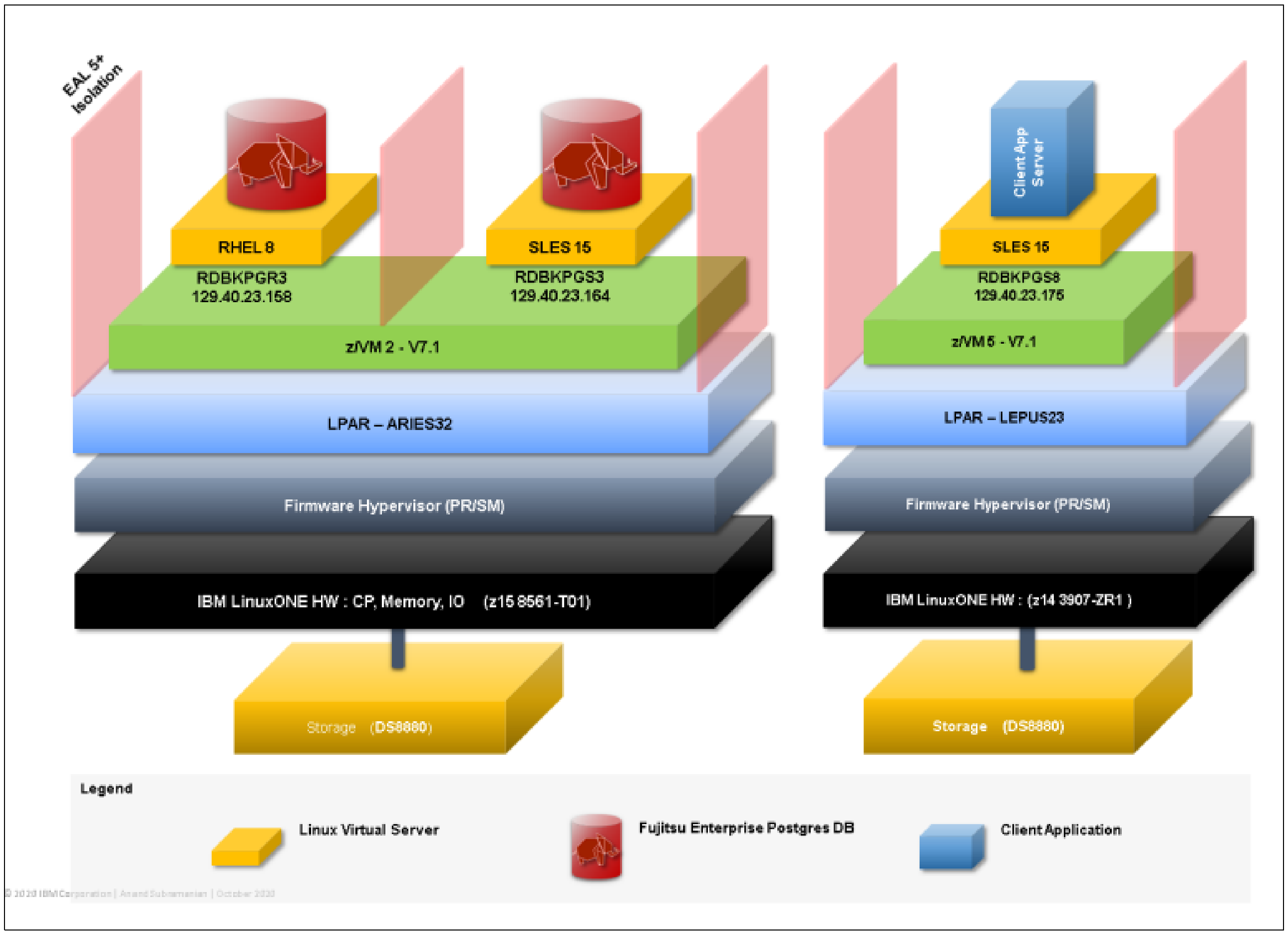


Figure 3-19 Application development feature lab environment with FUJITSU Enterprise Postgres 12

3.1.6 Summary of lab environment

Our lab environment is summarized in Table 3-1 on page 97 and sorted by server name and LPAR.

Table 3-1 Lab environment summary by IBM LinuxONE servers and logical partitions.

IBM LinuxONE server	LPAR	Guest	Lab environment
IBM LinuxONE III Multi Frames	ARIES32	RDBKGR1	HA on Red Hat Enterprise Linux 8
		RDBKGR2	
		RDBKGR3	Application Development Feature on Red Hat Enterprise Linux 8
		RDBKGR4	Security Feature on Red Hat Enterprise Linux 8
		RDBKGS1	HA on SUSE Linux Enterprise Server 12 SP4
		RDBKGS2	
		RDBKGS3	Application Development Feature on SUSE Linux Enterprise Server 12 SP4
	ARIES34	RDBKGR5	Security Feature on Red Hat Enterprise Linux 8
		RDBKGR6	HA on Red Hat Enterprise Linux 8
		RDBKGS5	Security Feature on SUSE Linux Enterprise Server 12 SP4
IBM LinuxONE Rockhopper II	LEPUS23	RDBKGR7	HA on Red Hat Enterprise Linux 8
		RDBKGR8	
		RDBKGS7	HA on SUSE Linux Enterprise Server 12 SP4
		RDBKGS8	

Table 3-2 summarizes our lab environment, which is sorted by the environment and type of server.

Table 3-2 Lab environment summary by lab environment

Lab environment	IBM LinuxONE server	LPAR	Guest	Environment highlights
HA on Red Hat Enterprise Linux 8	IBM LinuxONE III Multi Frames	ARIES32	RDBKGR1	
			RDBKGR2	
		ARIES34	RDBKGR6	
	IBM LinuxONE Rockhopper II	LEPUS23	RDBKGR7	
			RDBKGR8	
HA on SUSE Linux Enterprise Server 12 SP4	IBM LinuxONE III Multi Frames	ARIES32	RDBKGS1	
			RDBKGS2	
		ARIES34	RDBKGS6	
	IBM LinuxONE Rockhopper II	LEPUS23	RDBKGS7	
			RDBKGS8	

Lab environment	IBM LinuxONE server	LPAR	Guest	Environment highlights
Security Feature on Red Hat Enterprise Linux 8	IBM LinuxONE III Multi Frames	ARIES32	RDBKGR4	4x dedicated CryptoExpress7S cards customized as 8x separated HSM engines.
		ARIES34	RDBKGR5	4x dedicated CryptoExpress7S cards customized as 8x separated HSM engines.
Security Feature on SUSE Linux Enterprise Server 12 SP4	IBM LinuxONE III Multi Frames	ARIES32	RDBKGS4	
		ARIES34	RDBKGS5	
Application Development Feature on RHLE8	IBM LinuxONE III Multi Frames	ARIES32	RDBKGR3	HA on SUSE Linux Enterprise Server 12 SP4.
Application Development Feature on SUSE Linux Enterprise Server 12 SP4	IBM LinuxONE III Multi Frames	ARIES32	RDBKGS3	

3.2 FUJITSU Enterprise Postgres 12 server installation on LinuxONE

In this section, we describe the steps to install a stand-alone FUJITSU Enterprise Postgres 12 Advanced Edition server on Red Hat Enterprise Linux 8 and SUSE Linux Enterprise Server 12 SP4 virtual machines (VMs) in IBM LinuxONE III.

3.2.1 Prerequisites

In this section, we describe the hardware and operating requirements for the installation.

Hardware prerequisites

- Memory: At least 512 MB of memory is required.
- HSM: If TDE is used with an HSM, then IBM Z Crypto Express adapter must support either the CCA coprocessor or the EP11 coprocessor.
- A minimum of 500 MB of disk space.
- Recommended file system: ext4.

Note: In the use cases that are described in this publication, it is presumed that the reader is familiar with the stand-alone installation steps that are described in this section.

OS prerequisites

Table 3-3 on page 99 provides the list of packages that are required for operations on Red Hat Enterprise Linux 8. Some of the packages are required for specific operations, as listed in the Description column.

Table 3-3   FUJITSU Enterprise Postgres 12 Advanced Edition prerequisite packages on Red Hat Enterprise Linux 8

Package name	Description
`alsa-lib`	
`audit-libs`	
`cyrus-sasl-lib`	
`pcp-system-tools`	Required when using parallel scan.
`glibc`	
`libnsl2`	
`libicu`	Provides collation support.
	Install 60.x.
`libgcc`	
`libmemcached`	Required when using Pgpool-II.
`libstdc++`	
`libtool-ltdl`	Required when using Open Database Connectivity (ODBC) drivers.

`llvm`	Version 8.0.x of llvm is required to run SQL with runtime compilation (just-in-time (JiT) compilation). Install `llvm-libs` from Application Streams.
	FUJITSU Enterprise Postgres 12 uses runtime compilation by default. If you do not want to use runtime compilation, turn off the **`jit`** parameter in `postgresql.conf`. You do not need to install `llvm` if you turn off the **`jit`** parameter.
	Installing `llvm` without turning off the **`jit`** parameter might result in an error when running SQL.
	Note: If JIT compilation that uses `llvm` is performed on Red Hat Enterprise Linux 8, a verbose server message is logged one or more times on first query execution for each associated database connection:
	`<Message>`
	`ORC error: No callback manager available for s390x-ibm-linux`
	This error message results from a known deficiency in one of the features of the LLVM library for IBM Z, but this feature is not used by the server and the error does not otherwise affect LLVM operation.
	To prevent this message from being logged, JIT compilation can be disabled by turning of the value of the configuration variable **`jit`**.

`ncurses-libs`	
`net-tools`	
`nss-softokn-freebl`	
`opencryptoki`	Required if TDE is used with an HSM. Install 3.12.1 or later into the default directory.
`pam`	Required when using PAM authentication.
`perl-libs`	Required when using PL/Perl. Install 5.26.
`python3`	Required when using PL/Python based on Python 3. Install 3.7.x.
`redhat-lsb`	
`libselinux`	Required for `sepgsql`.
`tcl`	Required when using PL/Tcl. Install 8.6.

Package name	Description
`unzip`	
`xz-libs`	
`zlib`	
`java-1.8.0-openjdk`	Required when using database multiplexing and WebAdmin. Use build 1.8.0_222-b10 or later for the S390x architecture.

Table 3-4 provides the list of packages that are required for operations on SUSE 12. Some of the packages are required for specific operations, as listed in the Description column.

Table 3-4 FUJITSU Enterprise Postgres 12 Advanced Edition prerequisite packages on SUSE 12

Package name	Description
`alsa-lib`	
`audit-libs`	
`dstat`	Required when using parallel scan.
`glibc`	
`libgcc`	
`libicu`	Provides collation support. Install 52.
`libmemcached`	Required when using Pgpool-II.
`libstdc++`	
`libtool-ltdl`	Required when using ODBC drivers.
`llvm`	Install Version 7.0.x of `llvm` to run SQL with runtime compilation (JiT compilation) and add the directory where the shared library `libLLVM -*.so` is located next to the environment variable `LD_LIBRARY_PATH`. FUJITSU Enterprise Postgres 12 uses runtime compilation by default. If you do not want to use runtime compilation, turn off the **`jit`** parameter in `postgresql.conf`. You do not need to install `llvm` if you turn off the **`jit`** parameter. Failure to install `llvm` without turning off the **`jit`** parameter might result in an error when run SQL.
`LLVM-libs`	Install version 7.0.x. (*1).
`ncurses-libs`	
`net-tools`	
`nss-softokn-freebl`	
`opencryptoki`	Required if TDE is used with an HSM. Install 3.10 or later into the default directory.
`pam`	Required when using PAM authentication.
`perl-libs`	Required when using PL/Perl. Install 5.18.
`python3`	Required when using PL/Python based on Python 3. Install 3.7.x.

Package name	Description
sysstat	Required when using the **pgx_fjqssinf** command. Set up the **sar** command after installation.
tcl	Required when using PL/Tcl. Install 8.6.
xz-libs	
zlib	
java-1.8.0-openjdk	Required when using database multiplexing and WebAdmin. Use build 1.8.0_181-b13 or later for the S390x architecture.

Next, we describe the step by step installation procedure on Red Hat Enterprise Linux 8 and SUSE Linux Enterprise Server 12.

3.3 Installing and configuring a stand-alone database instance on a Red Hat Enterprise Linux 8 guest

To install and configure a stand-alone database instance on a Red Hat Enterprise Linux 8 Linux guest, complete the following steps:

1. Run the following commands to install the international components for Unicode libraries through `libicu`. Also, install the library `libns`, which provides a transport layer interface to the networking services.

   ```
   # yum install libnsl
   # yum install libicu
   ```

2. Next, set up `JAVA_HOME`. For this example, we used OpenJDK Version 1.8.0 and used the following command:

   ```
   #export
   JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre"
   ```

3. Mount the FUJITSU Enterprise Postgres 12 Advanced Edition ISO and start the installation as shown in Example 3-1.

Example 3-1 Mounting the ISO

```
#mount -t iso9660 -r -o loop /fep12iso/fsep12_ae_linux64.iso /mnt2
#cd /mnt2/SERVER/Linux/packages/r80s390x
# rpm -ivh FJSVfsep-SV-12-1200-0.el8.s390x.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:FJSVfsep-SV-12-1200-0.el8        ################################# [100%]
```

4. Install the WebAdmin component of the FUJITSU Enterprise Postgres, as shown in Example 3-2. WebAdmin provides an easy way to create and manage FUJITSU Enterprise Postgres clusters by using a GUI.

Example 3-2 Installing WebAdmin

```
#mount -t iso9660 -r -o loop /fep12iso/fsep12_ae_linux64.iso /mnt2
#cd /mnt2/WEBADMIN/Linux/packages/r80s390x
# rpm -ivh FJSVfsep-WAD-12-1200-0.el8.s390x.rpm
Verifying...                          ################################# [100%]
```

```
Preparing...                          ################################# [100%]
Updating / installing...
   1:FJSVfsep-WAD-12-1200-0.el8       ################################# [100%]
```

5. This step is optional for a stand-alone installation because Mirroring Controller (MC) is required in HA implementations that have more than one database server that is configured as the primary and standby. Run the command shown in Example 3-3 if there is a requirement to convert this installation into a HA implementation in the future.

Example 3-3 Installing Mirroring Controller

```
#/opt/fsepv12server64/bin/mc_update_jre_env
INFO:
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre/lib/s390x/server/l
ibjvm.so is in use.
```

6. Create the database administrator user ID `fsepuser`, and set the password to `fsepuser` by using the following commands:

   ```
   #useradd -m fsepuser
   #passwd fsepuser
   ```

7. After the user ID is created, prepare the directories (see Example 3-4 on page 103) to store the FUJITSU Enterprise Postgres binary files, transaction logs, and backup files. Figure 3-20 shows the components that should be logically placed in directories and different types of storage devices for performance and reducing I/O bottlenecks.

Note: For more information about estimating disk space requirements, see Appendix E, “Estimating Database Disk Space Requirements” in *FUJITSU Enterprise Postgres 12 Installation and Setup Guide for Server*.

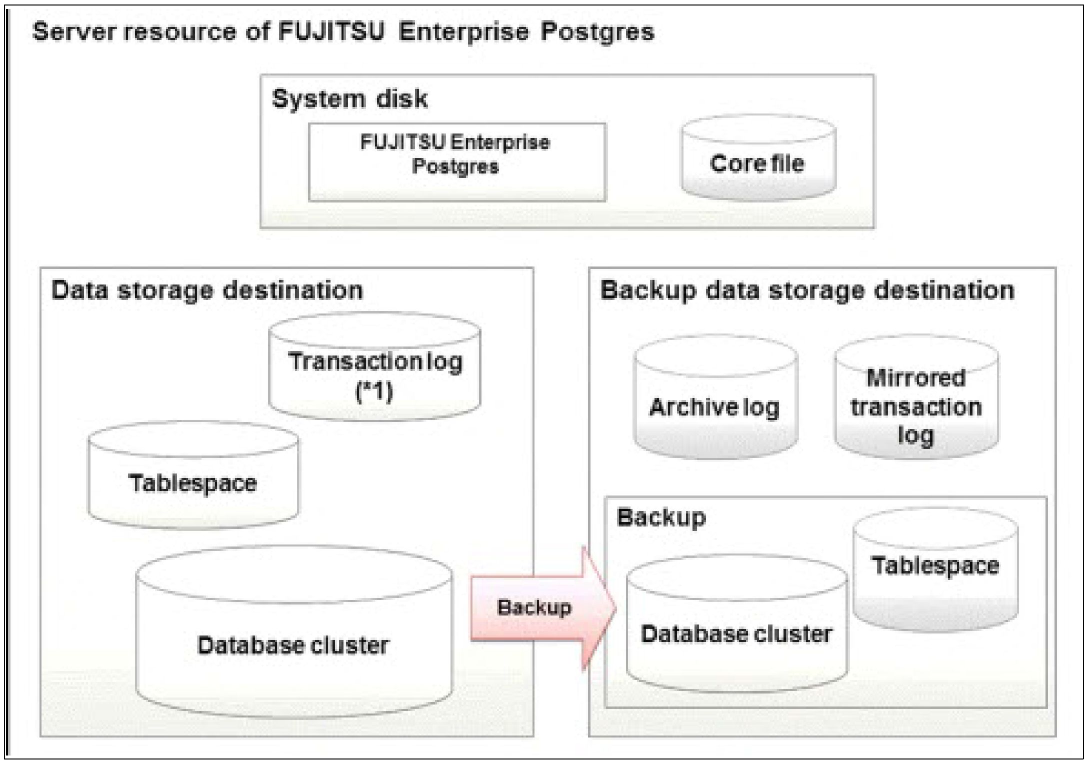


Figure 3-20 Server resource for FUJITSU Enterprise Postgres

Example 3-4 Preparing the directories

```
# mkdir -p /database/inst1
# chown -R fsepuser:fsepuser /database
# chmod 700 /database/inst1
# mkdir -p /transaction/inst1
# chown -R fsepuser:fsepuser /transaction
# chmod 700 /transaction/inst1
# mkdir -p /backup/inst1
# chown -R fsepuser:fsepuser /backup
# chmod 700 /backup/inst1
```

8. To enable communication, grant access to the ports 27500, 27515, 27516, 27540, and 27541 by editing the `fsep.xml` file (Example 3-5).

   Port 27500 is used by FUJITSU Enterprise Postgres, ports 27515 and 27516 are used by WebAdmin, and ports 27540 and 27541 are used by the MC.

Example 3-5 Granting access to the ports

```
# vim /etc/firewalld/services/fsep.xml
# cat /etc/firewalld/services/fsep.xml
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>FSEP12</short>
  <description>FUJITSU Enterprise Postgres Database Server</description>
  <port protocol="tcp" port="27500"/>
  <port protocol="tcp" port="27515"/>
  <port protocol="tcp" port="27516"/>
  <port protocol="tcp" port="27540"/>
  <port protocol="tcp" port="27541"/>
</service>
# firewall-cmd --reload
# firewall-cmd --get-services | grep fsep
# firewall-cmd --permanent --zone=public --add-service=fsep
# firewall-cmd --reload
# firewall-cmd --list-all | grep fsep
```

9. Configure the environment variables, as shown in Example 3-6.

Example 3-6 Configuring the environment variables

```
# su - fsepuser
$ PATH=/opt/fsepv12server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

10. Create the database instance `inst1`, as shown in Example 3-7.

Example 3-7 Creating the database instance

```
$ initdb -D /database/inst1 --waldir=/transaction/inst1 --lc-collate="C"
--lc-ctype="C" --encoding=UTF8
The files belonging to this database system will be owned by user "fsepuser".
This user must also own the server process.
The database cluster will be initialized with locales
  COLLATE:  C
  CTYPE:    C
  MESSAGES: en_US.UTF-8
  MONETARY: en_US.UTF-8
  NUMERIC:  en_US.UTF-8
  TIME:     en_US.UTF-8
The default text search configuration will be set to "english". (15541)
Data page checksums are disabled. (18153)
fixing permissions on existing directory /database/inst1 ... ok (15516)
fixing permissions on existing directory /transaction/inst1 ... ok (15516)
creating subdirectories ... ok (15516)
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ...  (19842)America/New_York
creating configuration files ... ok (15516)
running bootstrap script ... ok (15516)
performing post-bootstrap initialization ... ok (15516)
syncing data to disk ... ok (15516)
initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.
Success. You can now start the database server using:
    pg_ctl -D /database/inst1 -l logfile start
```

11. Now that the first database instance is created, configure the instance to accept remote connections, as shown in Example 3-8. By default, FUJITSU Enterprise Postgres allows connections only from the local machine where the database server is installed. You also need to turn on the logging collector to capture the log messages into the log files. This parameter can be set only at the server start.

Example 3-8 Configuration file modifications

```
$ vim /database/inst1/postgresql.conf
listen_addresses = '*'
port = 27500
logging_collector = on
```

12. Configure the host-based authentication by setting up the `pg_hba.conf` file (Example 3-9). The parameters in `pg_hba.conf` control the authentication process in PostgreSQL and in FUJITSU Enterprise Postgres.

Example 3-9 Configuring host-based authentication

```
$ vim /database/inst1/pg_hba.conf
host    all             all             xxx.xx.xx.0/24     trust
```

With this configuration, all the connections originating from the xxx.xx.xx.0 network can access the stand-alone database instance `inst1`. Make sure that you provide the accurate network range in place of xxx.xx.xx.0/24.

Note: FUJITSU Enterprise Postgres supports several authentication methods, including TRUST, IDENT, PEER, Generic Security Standard Application Programming Interface (GSSAPI), LDAP, Salted Challenge Response Authentication Mechanism (SCRAM), SHA-256, MD5, and password. For more information about host-based authentication methods, see 4.4, “FUJITSU Enterprise Postgres encryption on IBM LinuxONE with crypto cards” on page 126.

13.Start the database instance `inst1` on the stand-alone database server. The command and output are shown in Example 3-10.

Example 3-10 Starting the database instance

```
$ pg_ctl start -D /database/inst1
waiting for server to start....2020-12-25 00:19:37.058 EST [2154304] LOG:
starting PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (GCC) 8.3.1
20190507 (Red Hat 8.3.1-4), 64-bit
2020-12-25 00:19:37.060 EST [2154304] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2020-12-25 00:19:37.060 EST [2154304] LOG:  listening on IPv6 address "::", port
27500
2020-12-25 00:19:37.061 EST [2154304] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2020-12-25 00:19:37.067 EST [2154304] LOG:  redirecting log output to logging
collector process
2020-12-25 00:19:37.067 EST [2154304] HINT:  Future log output will appear in
directory "log".
 done
server started
```

14.After the database instance starts, verify the connection by running the **psql** command-line user interface (CUI) from the stand-alone database server, as shown in Example 3-11. Enter the actual IP address of the stand-alone server in place of xxx.xx.xx.xxx in the **psql** command. Press the Enter key and wait for the `postgres=#` prompt.

Example 3-11 Verifying the connection

```
$ psql -h <xxx.xx.xx.xxx> -p 27500 -d postgres -U fsepuser
psql (12.1)
Type "help" for help.
postgres=#
```

The installation of stand-alone FUJITSU Enterprise Postgres 12 Advanced Edition on a Red Hat Enterprise Linux 8 Linux guest is complete.

3.4 Installing and configuring a stand-alone database instance on a SUSE Linux Enterprise Server 12 Linux guest

To install and configure a stand-alone database instance on a SUSE Linux Enterprise Server 12 Linux guest, complete the following steps:

1. Install Java by running the **zypper** command, as shown in Example 3-12.

Example 3-12 Installing Java

```
# zypper install java
Loading repository data...
Reading installed packages...
'java' not found in package names. Trying capabilities.
Resolving package dependencies...
The following 4 NEW packages are going to be installed:
  java-1_8_0-openjdk java-1_8_0-openjdk-headless javapackages-tools libgif6
```

```
4 new packages to install.
Overall download size: 43.3 MiB. Already cached: 0 B. After the operation,
additional 56.7 MiB will be used.
Continue? [y/n/...? shows all options] (y): y
Retrieving package javapackages-tools-2.0.1-11.1.s390x
                                                (1/4),  51.1 KiB (129.9 KiB unpacked)
Retrieving: javapackages-tools-2.0.1-11.1.s390x.rpm .....................[done]
Retrieving package libgif6-5.0.5-12.1.s390x
                                                (2/4),  20.8 KiB ( 39.4 KiB unpacked)
Retrieving: libgif6-5.0.5-12.1.s390x.rpm ................................[done]
Retrieving package java-1_8_0-openjdk-headless-1.8.0.181-27.26.2.s390x
                                                (3/4),  42.9 MiB ( 55.9 MiB unpacked)
Retrieving: java-1_8_0-openjdk-headless-1.8.0.181-27.26.2.s390x.rpm ......[done]
Retrieving package java-1_8_0-openjdk-1.8.0.181-27.26.2.s390x
                                                (4/4), 370.8 KiB (647.2 KiB unpacked)
Retrieving: java-1_8_0-openjdk-1.8.0.181-27.26.2.s390x.rpm ...............[done]
Checking for file conflicts: .............................................[done]
(1/4) Installing: javapackages-tools-2.0.1-11.1.s390x ....................[done]
(2/4) Installing: libgif6-5.0.5-12.1.s390x ...............................[done]
(3/4) Installing: java-1_8_0-openjdk-headless-1.8.0.181-27.26.2.s390x <12%>==[-]

(3/4) Installing: java-1_8_0-openjdk-headless-1.8.0.181-27.26.2.s390x
..........................................................................
.......................................................................[do
ne]
Additional rpm output:
update-alternatives: using /usr/lib64/jvm/jre-1.8.0-openjdk/bin/java to provide
/usr/bin/java (java) in auto mode
update-alternatives: using /usr/lib64/jvm/jre-1.8.0-openjdk to provide
/usr/lib64/jvm/jre-openjdk (jre_openjdk) in auto mode
update-alternatives: using /usr/lib64/jvm/jre-1.8.0-openjdk to provide
/usr/lib64/jvm/jre-1.8.0 (jre_1.8.0) in auto mode
(4/4) Installing: java-1_8_0-openjdk-1.8.0.181-27.26.2.s390x
..........................................................................
..........................................................................
......[done]
```

2. Set up `JAVA_HOME` by using the following command. For this example, we used OpenJDK Version 1.8.0.

   ```
   #export JAVA_HOME="/usr/lib64/jvm/java-1.8.0-openjdk-1.8.0/jre"
   ```

3. Mount the FUJITSU Enterprise Postgres 12 Advanced Edition ISO and start the installation, as shown in Example 3-13.

Example 3-13 Starting the installation

```
# mount -t iso9660 -r -o loop /fep12iso/fsep12_ae_linux64.iso /mnt2
# cd /mnt2/SERVER/Linux/packages/SUSE12s390x
# rpm -ivh FJSVfsep-SV-12-1200-0.s390x.rpm
Preparing...                          ################################# [100%]
Updating / installing...
   1:FJSVfsep-SV-12-1200-0            ################################# [100%]
```

4. Install the WebAdmin component of FUJITSU Enterprise Postgres. WebAdmin provides an easy way to create and manage FUJITSU Enterprise Postgres clusters through a GUI. Follow the commands shown in Example 3-14 on page 107.

Example 3-14 Installing WebAdmin

```
# cd
# umount /mnt2
# mount -t iso9660 -r -o loop /fep12iso/fsep12_ae_linux64.iso /mnt2
# cd /mnt2/WEBADMIN/Linux/packages/SUSE12s390x/
# rpm -ivh FJSVfsep-WAD-12-1200-0.s390x.rpm
Preparing...                          ################################# [100%]
Updating / installing...
   1:FJSVfsep-WAD-12-1200-0           ################################# [100%]
```

5. This step is optional for a stand-alone installation because MC is required in HA implementations with more than one database server that is configured as primary and standby. Run the commands that are shown in Example 3-15 if there is a requirement to convert this installation into a HA implementation in the future.

Example 3-15 Installing Mirroring Controller

```
# /opt/fsepv12server64/bin/mc_update_jre_env
INFO: /usr/lib64/jvm/java-1.8.0-openjdk-1.8.0/jre/lib/s390x/server/libjvm.so is in
use.
```

6. Create the database admin user ID `fsepuser` and set the password by using the following commands:

```
#useradd -m -U fsepuser
#passwd fsepuser
```

7. After the user ID is created, the next step is to prepare the directories to store the FUJITSU Enterprise Postgres binary files, transaction logs, and backup (Example 3-16). Figure 3-21 shows the components that should be logically placed in directories, and the different types of storage devices for performance and reducing I/O bottlenecks.

Note: For more information about estimating disk space requirements, see Appendix E, "Estimating Database Disk Space Requirements" in *FUJITSU Enterprise Postgres 12 Installation and Setup Guide*.

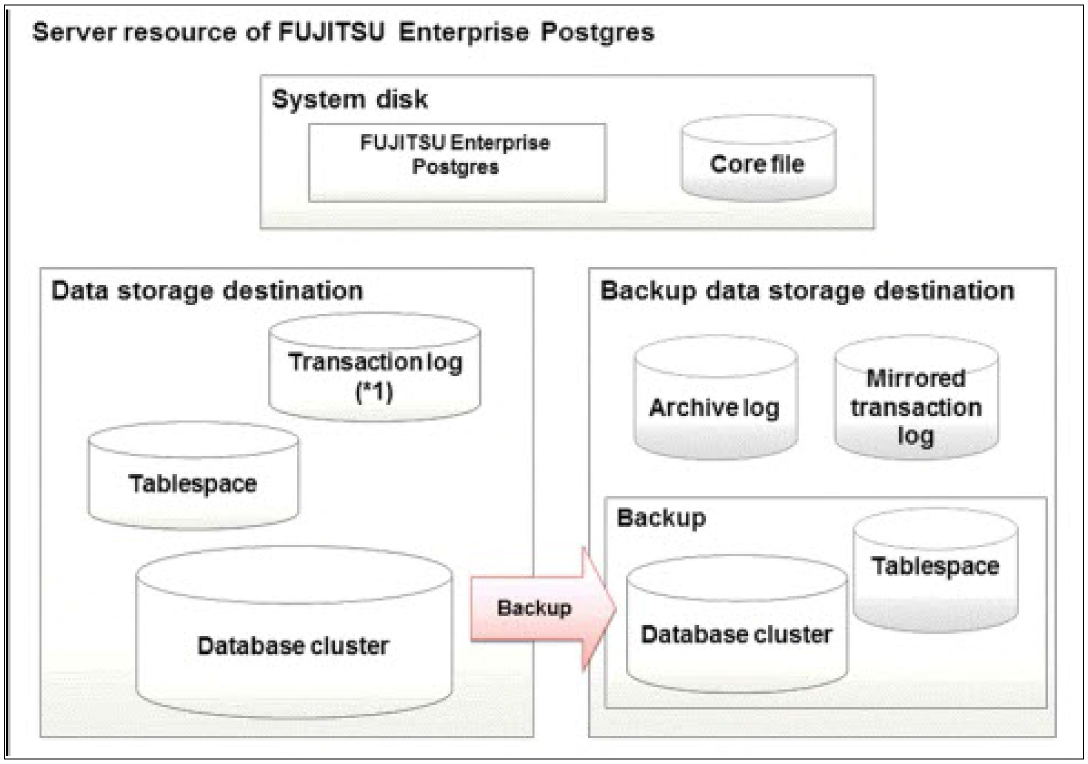


Figure 3-21 Server resource for FUJITSU Enterprise Postgres

Example 3-16 Preparing the directories

```
# mkdir -p /database/inst1
# chown -R fsepuser:fsepuser /database
# chmod 700 /database/inst1
# mkdir -p /transaction/inst1
# chown -R fsepuser:fsepuser /transaction
# chmod 700 /transaction/inst1
# mkdir -p /backup/inst1
# chown -R fsepuser:fsepuser /backup
# chmod 700 /backup/inst1
```

8. To enable communication, grant access to ports 27500, 27515, 27516, 27540, and 27541, as shown in Example 3-17 on page 109. Port 27500 is used by FUJITSU Enterprise Postgres, ports 27515 and 27516 are used by WebAdmin, and ports 27540 and 27541 are used by MC.

Example 3-17 Enabling communication

```
# SUSEfirewall2 open EXT TCP 27500
# SUSEfirewall2 open EXT TCP 27515
# SUSEfirewall2 open EXT TCP 27516
# SUSEfirewall2 open EXT TCP 27540
# SUSEfirewall2 open EXT TCP 27541
# SUSEfirewall2 stop
<38>Feb 18 20:21:44 SuSEfirewall2[14092]: Firewall rules unloaded.
# SUSEfirewall2 start
<38>Feb 18 20:21:49 SuSEfirewall2[14126]: Setting up rules from
/etc/sysconfig/SuSEfirewall2 ...
<38>Feb 18 20:21:49 SuSEfirewall2[14126]: using default zone 'ext' for interface
eth0
<38>Feb 18 20:21:50 SuSEfirewall2[14126]: Firewall rules successfully set
```

9. Configure the environment variables, as shown in Example 3-18.

Example 3-18 Configuring the environment variables

```
# su - fsepuser
$ PATH=/opt/fsepv12server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

10. Create the database instance `inst1`, as shown in Example 3-19.

Example 3-19 Creating the database instance

```
$ initdb -D /database/inst1 --waldir=/transaction/inst1 --lc-collate="C"
--lc-ctype="C" --encoding=UTF8
The files belonging to this database system will be owned by user "fsepuser".
This user must also own the server process.
The database cluster will be initialized with locales
  COLLATE:  C
  CTYPE:    C
  MESSAGES: en_US.UTF-8
  MONETARY: en_US.UTF-8
  NUMERIC:  en_US.UTF-8
  TIME:     en_US.UTF-8
The default text search configuration will be set to "english". (15541)
Data page checksums are disabled. (18153)
fixing permissions on existing directory /database/inst1 ... ok (15516)
fixing permissions on existing directory /transaction/inst1 ... ok (15516)
creating subdirectories ... ok (15516)
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ...  (19842)America/New_York
creating configuration files ... ok (15516)
running bootstrap script ... ok (15516)
performing post-bootstrap initialization ... ok (15516)
syncing data to disk ... ok (15516)
initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.
```

```
Success. You can now start the database server using:
    pg_ctl -D /database/inst1 -l logfile start
```

11. After the first database instance is created, configure the instance to accept remote connections (Example 3-20). By default, FUJITSU Enterprise Postgres allows connections only from the local machine on which the database server is installed. Turn on the logging collector to capture log messages into log files. This parameter can be set only at the server start.

Example 3-20 Configuring the instance to accept remote connections

```
$ vim /database/inst1/postgresql.conf
listen_addresses = '*'
port = 27500
logging_collector = on
```

12. Configure the host-based authentication by setting up the `pg_hba.conf` file, as shown in Example 3-21. The parameters in `pg_hba.conf` control the authentication process in PostgreSQL and in FUJITSU Enterprise Postgres.

Example 3-21 Configuring host-based authentication

```
$ vim /database/inst1/pg_hba.conf
host    all             all             xxx.xx.xx.0/24          trust
```

With the configuration that is shown in Example 3-20 and Example 3-21, all the connections originating from the xxx.xx.xx.0 network can access the stand-alone database instance `inst1`. Ensure that you provided the correct network range in place of xxx.xx.xx.0/24.

Note: FUJITSU Enterprise Postgres supports several authentication methods, including TRUST, IDENT, PEER, GSSAPI, LDAP, SCRAM, SHA-256, MD5, and password. For more information about the host-based authentication method, see 4.4, "FUJITSU Enterprise Postgres encryption on IBM LinuxONE with crypto cards" on page 126.

13. Start the database instance `inst1` on the stand-alone database server by running the command that is shown in Example 3-22.

Example 3-22 Starting the database instance

```
$ pg_ctl start -D /database/inst1
waiting for server to start....2021-02-18 20:26:09.272 EST [14680] LOG:  starting
PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (SUSE Linux) 4.8.5, 64-bit
2021-02-18 20:26:09.272 EST [14680] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2021-02-18 20:26:09.272 EST [14680] LOG:  listening on IPv6 address "::", port
27500
2021-02-18 20:26:09.273 EST [14680] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2021-02-18 20:26:09.279 EST [14680] LOG:  redirecting log output to logging
collector process
2021-02-18 20:26:09.279 EST [14680] HINT:  Future log output will appear in
directory "log".
 done
server started
```

14. Verify the connection by starting the **psql** CUI from the stand-alone database server, as shown in Example 3-23. Enter the IP address of your stand-alone server in place of xxx.xx.xx.xxx in the **psql** command. Press the Enter key and wait for the `postgres=#` prompt.

Example 3-23 Verifying the connection

```
$ psql -h xxx.xx.xx.xxx -p 27500 -d postgres -U fsepuser
psql (12.1)
Type "help" for help.
postgres=#
```

15. To stop the instance, run the command that is shown in Example 3-24.

Example 3-24 Stopping the instance

```
$ pg_ctl stop -D /database/inst1
waiting for server to shut down.... done
server stopped
```

The installation of stand-alone FUJITSU Enterprise Postgres 12 Advanced Edition on a SUSE Linux Enterprise Server 12 Linux guest is complete.

4

Data security with Transparent Data Encryption, Data Masking, and Audit Log

The Transparent Data Encryption (TDE) policy-based data masking and audit log features are available with FUJITSU Enterprise Postgres Advanced Edition on LinuxONE. TDE, when integrated with IBM LinuxONE Crypto Express adaptors, enables master encryption key management by using the Hardware Security Module (HSM), which means that no one may access the master encryption key because it is HSM-managed, which makes the underlying data tamper proof. FUJITSU Enterprise Postgres Data Masking obfuscates sensitive data without any changes to the existing SQLs and presents the results to the user depending on their role and the active data masking policy on the underlying FUJITSU Enterprise Postgres database objects. Additionally, FUJITSU Enterprise Postgres Audit Log enables tracing and auditing the usage of sensitive data and connection attempts of the database.

In this chapter, we describe the following concepts and use cases:

- IBM Linux cryptography and FUJITSU TDE concepts
- Implementing TDE with built-in, file-based KeyStore management
- Implementing TDE with LinuxONE CryptoCard hardware-based KeyStore management
- Implementing full, partial, and regular expression-based data masking
- Implementing FUJITSU Audit Log for auditing usage and access of sensitive data

This chapter covers the following topics:

- LinuxONE cryptography
- IBM LinuxONE cryptography features
- Transparent Data Encryption
- FUJITSU Enterprise Postgres encryption on IBM LinuxONE with crypto cards
- Data Masking feature

- Audit Logging
- Authentication

4.1 LinuxONE cryptography

In this chapter, we describe some of the concepts that are related to the cryptography features of an IBM LinuxONE server that is leveraged by FUJITSU Enterprise Postgres for its TDE use cases.

Here are some concepts that are used in technical documents that are related to IBM LinuxONE cryptography:

- Hardware secure module (HSM)

  An HSM is a tamper-proof cryptographic device that protects secrets (typically master keys) from being inspected. IBM CryptoExpress Common Cryptographic Architecture (CCA) coprocessors and EP11 coprocessors are certified as HSMs. The use of the word *CryptoExpress coprocessor* is interchangeable with *HSM* in this book.

  Each CryptoExpress coprocessor is assigned a unique decimal number starting from 0 as its index for logical partition (LPAR) and operating system (OS) access.

  Because a CryptoExpress feature card can have one or two coprocessors in it, consider the redundancy boundary when planning coprocessor assignment for LPARs on a IBM LinuxONE server.

- Cryptographic domain

  A cryptographic domain is a physical set of registers or queue registers in a CryptoExpress coprocessor, which is protected within the CryptoExpress card's secure boundary.

  Each domain of a CryptoExpress coprocessor constitutes a (virtual) HSM and maintains a domain-specific master key or a set of master keys.

  In the z/VM and Linux context, a cryptographic domain is also referred to as an adjunct processor (AP).

  Depending on the type of feature, a CryptoExpress coprocessor can have up to 85 domains, which are indexed by decimal numbers starting from 0.

  A domain can be identified by a combination of the index number of its CryptoExpress coprocessor and the domain's index number, which is also known as the adjunct processor queue number (APQN) in z/VM and Linux. For example, APQN 06.0005 means domain 05 of CryptoExpress coprocessor number 06.

  An active IBM LinuxONE LPAR can use only one domain in a coprocessor at any point-in-time (PiT), and the domain cannot be in use by another active LPAR. Multiple domains in different coprocessors can be used by an LPAR for availability and performance improvement.

- Types of keys:

  – A *clear key* is a key in plain text. The bit pattern of a clear key is the one that is used in the mathematical description of a cipher. Therefore, whoever knows the clear key can perform cryptographic operations (like encrypt or decrypt) by using that clear key.

  – A *master key* is a wrapping key or a key-encrypting-key (KEK) that is used to encrypt a key. In the EP11 documentation, this master key is often referred to as a wrapping key or the Master Encryption Key (MEK). Each domain of a CryptoExpress coprocessor can contain active master keys that are used to generate secure keys.

– A *secure key* is a key that is encrypted by an HSM master key. Secure keys can be used in cryptographic functions that are performed by the HSM. Thus, each cryptographic function on a secure key requires I/O operations to the HSM. A secure key is only valid in the HSM it was generated in.

The FUJITSU Enterprise Postgres naming convention uses two types of encryption keys: Data Encryption Keys (DEKs), and a MEK that is used at the database layer for TDE. These types are in addition to the MEK that is generated and stored within the HSM and used to wrap the database-level MEK.

To avoid confusion for this document, the following conventions are used to differentiate between the three types of keys that are used for TDE on an IBM LinuxONE server:

– HSM MEK: Wrapping key that is stored on the HSM only and used for wrapping the TDE MEK.

– TDE MEK: Encryption key that is used by the FUJITSU Enterprise Postgres Database to encrypt the TDE DEK, and on the IBM LinuxONE, it is wrapped with the HSM MEK for extra security.

– TDE DEK: Encryption key that is used within the FUJITSU Enterprise Postgres Database for database table level encryption. It is wrapped with the TDE MEK for extra security.

Figure 4-1 provides an overview of FUJITSU Enterprise Postgres TDE on IBM LinuxONE.

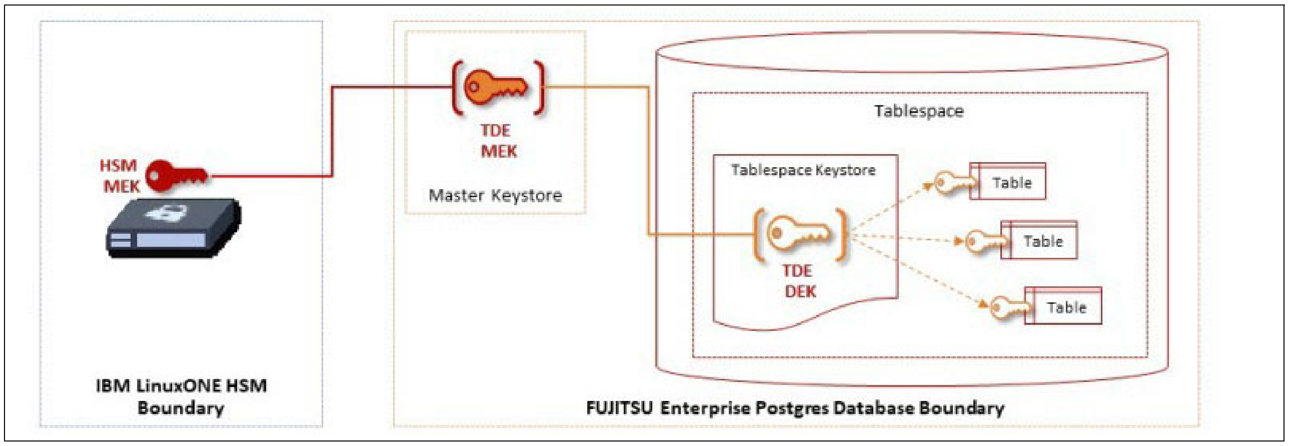


Figure 4-1 FUJITSU Enterprise Postgres Transparent Data Encryption on IBM LinuxONE

This chapter covers the following topics:

- IBM LinuxONE cryptography features
- Transparent Data Encryption
- FUJITSU Enterprise Postgres encryption on IBM LinuxONE with crypto cards
- Data Masking feature
- Audit Logging
- Authentication

4.2 IBM LinuxONE cryptography features

The IBM LinuxONE server includes a standard cryptography feature, the Central Processor Assist for Cryptographic Function (CPACF), and two optional features, the CryptoExpress feature card and Trusted Key Entry (TKE) workstation.

4.2.1 Central Processor Assist for Cryptographic Function

The CPACF delivers high-speed, on-chip symmetric cryptographic and hashing functions that enhance the encryption and decryption performance of clear key operations, including Secure Sockets Layer (SSL), virtual private networks (VPNs), and data-storing applications.

Every processor unit (core) of the IBM LinuxONE server has a dedicated co-processor that is responsible for CPACF cryptographic functions. The CPACF is a no-charge enablement feature of the IBM LinuxONE server.

4.2.2 CryptoExpress feature card

The CryptoExpress feature card is an optional, priced feature card of the IBM LinuxONE server that can be plugged into an I/O slot within the I/O drawers. Depending on the feature type, each feature card provides one or two HSMs to provide high-performance cryptographic functions to deploy cryptographically secure application environments.

The CryptoExpress feature card has built-in advanced tamper detection and response capabilities that are designed and certified at FIPS 140-2 Level 4, which is the highest cryptographic security level that is available.

The HSM on a CryptoExpress feature card can be configured to operate in one of the following three modes:

- Secure IBM CCA coprocessor

  The HSM on a CryptoExpress feature card supports IBM CCA, which is an architecture and a set of APIs to provide cryptographic algorithms, secure key management, and many functions that are required for high-data security use cases, such as those found in financial services or the defense industry.

  This mode also supports user-defined extension (UDX) services to implement cryptographic functions and algorithms in the HSM.

- Secure IBM Enterprise PKCS #11 (EP11) coprocessor

  The HSM on a CryptoExpress feature card supports the EP11 cryptography functions with a secure key.

  The EP11 library provides an interface like the industry standard PKCS #11 API. Existing applications that use PKCS #11 can benefit from using EP11 because they can be migrated easily to an IBM LinuxONE server and benefit from enhanced security by using secure key cryptography.

- Accelerator

  The HSM on a CryptoExpress feature card supports only RSA clear key and SSL acceleration with clear key.

For CryptoExpress feature cards with two HSMs, each HSM operates independently, so each can be configured for different operational modes if required.

The CPACF feature must be enabled for configuring and using CryptoExpress feature cards.

4.2.3 Trusted Key Entry workstation

A TKE workstation is an optional, priced feature that provides a secure, remote, and flexible method of management for keys and the CryptoExpress coprocessors (HSMs). TKE uses a highly secure logical channel through TCP/IP communication between the workstation and the CryptoExpress coprocessors that it manages to exchange sensitive data, such as coprocessor status information, and perform relevant HSM key management functions, such as master key creation or deletion.

The cryptographic functions on the TKE workstation are run by using a PCIe crypto card, which has the same built-in tamper detection and response capabilities as the HSM in a CryptoExpress feature card.

The TKE workstation can enable or disable specific elementary functions in CryptoExpress coprocessors by acting on the coprocessor access control points (ACPs).

A subset of the TKE management functions is available through the command-line interface (CLI) in Linux on IBM Z. These functions are necessary for day-to-day operations when the TKE workstation feature is not configured. However, the CLI does not benefit from the level of security that is achieved by using a TKE workstation. For example, entering the master key values by using the CLI is done in clear text.

Smart card readers with FIPS certified smart cards can be optionally added to a TKE workstation for the generation and loading of HSM MEKs.

A TKE workstation can be set up to manage CryptoExpress coprocessors in different IBM LinuxONE servers, even ones in different data centers. Multiple TKE workstations in different data centers can also be set up to manage CryptoExpress coprocessors in IBM LinuxONE servers that are deployed across different data centers for high availability (HA), disaster recovery (DR), and ease of management.

A TKE workstation with smart card readers and FIPS certified smart cards is mandatory when CryptoExpress coprocessors are configured as CCA in PCI-HSM mode or EP11 mode.

4.3 Transparent Data Encryption

In this section, we describe the steps that are used to configure and use encryption for database data on a FUJITSU Enterprise Postgres 12 server with Red Hat Enterprise Linux Server 8 on IBM LinuxONE.

In this section, we describe the following topics:

- Overview
- Architectural overview
- Configuring TDE
- Enabling encryption
- Managing keystore

4.3.1 Overview

TDE protects data that is stored in an OS file system from attackers who bypass the database server's authentication and access controls. Tablespace data is encrypted when it is stored to disk and decrypted when it is read from disk. This operation is handled transparently to users.

TDE is available by default with FUJITSU Enterprise Postgres. In FUJITSU Enterprise Postgres 12 running on IBM LinuxONE, TDE performance is enhanced by using the CPACF instructions set.

All user data, including tables, indexes, and Write-Ahead-Logs (WALs), that are spilled over temporary files of encrypted tablespace are encrypted by using the Advanced Encryption Standard (AES) algorithm.

For more information about FUJITSU Enterprise Postgres TDE, see Chapter 5. “Protecting Storage Data Using Transparent Data Encryption”, in *FUJITSU Enterprise Postgres 12 Operation Guide*.

4.3.2 Benefits of using TDE

Sensitive data can be protected by simply placing it in an encrypted tablespace. Data that is stored on a stolen disk is still protected regardless of login access to the database server. A passphrase is an extra piece of information that is needed to open the master keystore, which is not stored anywhere and must be remembered and entered by a database superuser.

Encryption/decryption is performed automatically and almost invisible to users. It does not require action from the user after the encryption function is enabled in the database.

All data transmission is encrypted, including backups and WALs that are involved in streaming replication from the primary server to the secondary server.

TDE helps enterprises to be compliant with PCI-DSS requirements by providing AES 256-bit encryption.

4.3.3 Architectural overview

TDE uses an envelope encryption strategy. All data within a specified tablespace is encrypted by TDE DEKs that are encrypted by a TDE master key. The TDE master key that is stored in the keystore is encrypted by a passphrase.

DEKs cannot be used without opening the master keystore where the TDE master key is. Deleting a TDE master key means crypto-shredding all encrypted data in the database cluster. Crypto-shredding means that the data is deleted or overwritten.

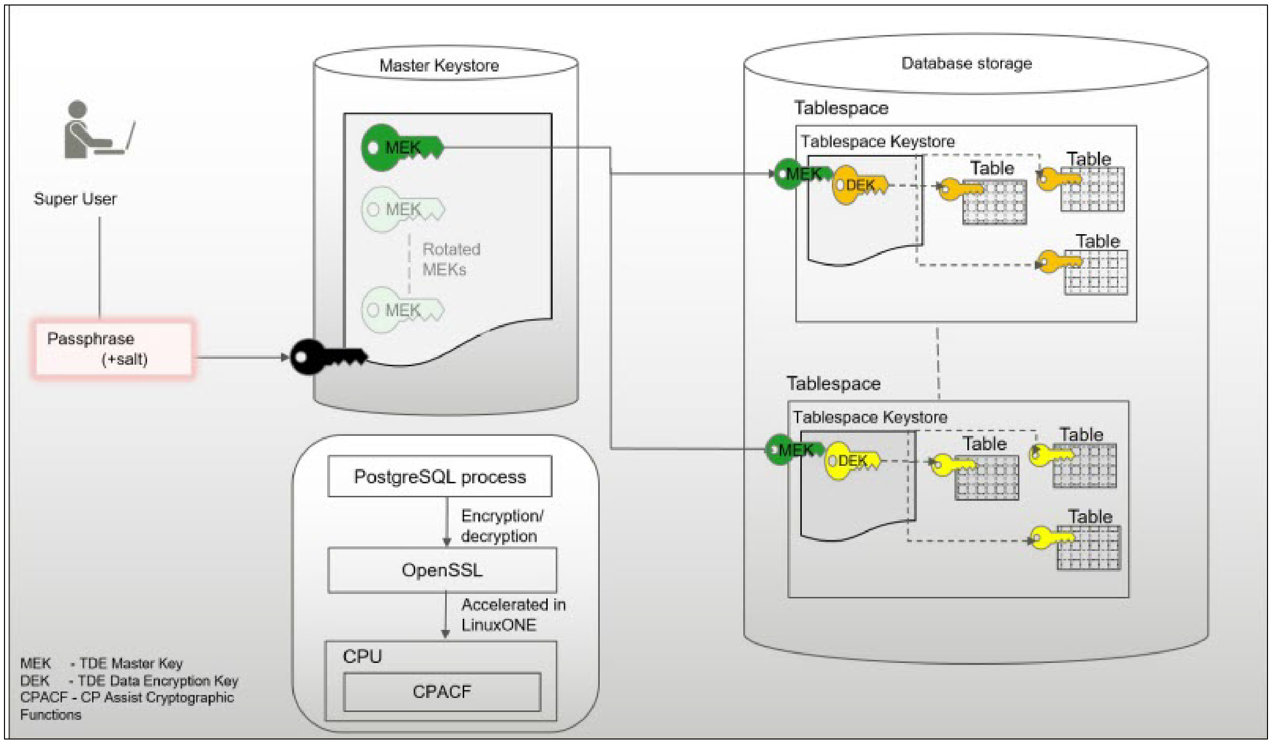


Figure 4-2 TDE architectural diagram

4.3.4 Configuring TDE

In this section, we describe the configuration parameters for TDE and provide steps about how to create a TDE master key.

Configuration parameters

To use TDE, a FUJITSU Enterprise Postgres Database server should be configured with the parameters that are shown in Table 4-1 in the configuration file `postgresql.conf`.

Table 4-1 Configuration parameters

Configuration parameter	Optional/Mandatory	Description
`keystore_location`	Mandatory	The file system directory to store master keystore file on. This location should be specific to the database cluster. Only users with a superuser privilege can configure this parameter.

Configuration parameter	Optional/Mandatory	Description
`tablespace_encryption_algorithm`	Optional	AES256, AES128, and none are the available encryption algorithm options. The user is expected to configure the encryption in the `postgresql.conf` file only when all further tablespaces by default should be encrypted. The user can also skip this configuration and configure encryption algorithm for each tablespace at tablespace creation time. Only users with the superuser privilege can configure this parameter.

For database servers that already are started, changes to the configurations become effective only after restarting the database server. Example 4-1 shows our configuration parameters as an example.

Example 4-1 Example settings of configuration parameters in the postgresql.conf file

```
keystore_location = ‘/home/fsepuser/keystore_dir’
tablespace_encryption_algorithm = ‘AES256’
```

Example 4-2 provides the command to restart the database server.

Example 4-2 Restarting the database server

```
[fsepuser@rdbkkpgr1 ~]$ pg_ctl -D ./testdb/ -l logfile restart
waiting for server to shut down.... done
server stopped
waiting for server to start.... done
server started
```

4.3.5 Creating a TDE master key

To start using encryption, a TDE master key must be created by a database superuser running the SQL command that is shown in Example 4-3 on page 121 against your database server.

The TDE master key is an internally generated encryption key that is not exposed anywhere in plain text format. The information is stored securely in a master keystore in encrypted format. The user must set a keystore passphrase to secure the master keystore during initial TDE master key creation. This passphrase is not stored anywhere and the onus is on the user to remember the passphrase. Loss of the passphrase means that the master keystore cannot be opened and hence decryption of secure data is not possible.

Example 4-3 Command to create the TDE master key

```
postgres=# select pgx_set_master_key('12345678');
pgx_set_master_key
-------------------

(One row)
```

Creation of the TDE master key command also opens the keystore so that the user can start creating encrypted tablespaces.

Note: The master keystore file is not included in a database backup and must be copied separately. The same passphrase that is used during backup should be used during recovery to open the keystore.

4.3.6 Enabling encryption

In this section, we demonstrate how to prepare a database for encryption/decryption. After every database server restart, the master keystore must be opened.

How to open the keystore

The master keystore (or simply called keystore) can be opened by two methods. Based on the preference and needs of the user, either of the following two options can be used:

- Command-line interface: Example 4-4 shows the command-line arguments that can be used to start a database server.

  To start the database server in encryption enabled mode, specify the passphrase to open the master keystore by using the **–keystore-passphrase** option that is specified in the **pg_ctl** command. This option prompts for user input when the database server starts.

Example 4-4 Command-line arguments to open a keystore during a server start

```
[fsepuser@rdbkkpgr1 ~]$ pg_ctl -D ./testdb/ -l logfile --keystore-passphrase start
Enter passphrase:
waiting for server to start.... done
server started
```

  You might need to use this option when the database server is starting in recovery mode and WAL records that are applied contain encrypted data. This option also means that user intervention is needed to enter the passphrase.

- SQL command: At any point when the database server is running, the keystore can be opened by running the SQL command that is shown in Example 4-5.

Example 4-5 SQL command to open the keystore

```
[fsepuser@rdbkkpgr1 ~]$ psql postgres
psql (12.1)
Type "help" for help.

postgres=# select pgx_open_keystore('12345678');
 pgx_open_keystore
-------------------

(One row)
```

The keystore can be opened only by users that have database superuser privileges.

Important: You must remember the passphrase because it cannot be recovered by any other means. The loss of the passphrase might mean crypto-shredding of your data.

Setting up the automatic opening of a keystore

Automatic opening of the keystore is required during DR and for deployments that require an auto-start of the database server with no manual intervention to enter the keystore passphrase. When this setup is in place, the database server restarts automatically in encryption-enabled mode without any needed user intervention to open the keystore.

The automatic opening of the keystore must be done by running the **`pgx_keystore`** command, as shown in Example 4-6.

Example 4-6 Automatic opening of the keystore

```
[fsepuser@rdbkkpgr1 ~]$ pgx_keystore -a -P ‘12345678’
/home/fsepuser/keystore_dir/keystore.ks
auto-open of the keystore has been enabled (14328)
```

Example 4-7 shows the command to check that the auto-open keystore (AKS) file was created. In our example, we see that the `keystore.aks` file was created successfully.

Example 4-7 Verifying the auto-open keystore command

```
[fsepuser@rdbkkpgr1 keystore_dir]$ ls -l
-rw-------. 1 fsepuser fsepuser 928 Oct 19 21:45 keystore.aks
-rw-------. 1 fsepuser fsepuser 928 Oct 19 21:08 keystore.ks
```

This command creates a stand-in keystore file (AKS file) with the extension `.aks` in the same directory where the master keystore file is stored, which is in a directory that is specified by the **`keystore_location`** configuration parameter.

AKS is encrypted, and the level of security should be considered only as intermediate and not as strong as a master keystore file.

AKS can be deleted at any point to stop automatic opening of a keystore. Also, AKS is valid only in the machine where it was created.

A MEK rotation process automatically updates the AKS file contents and no additional action is needed from the user.

Creating an encrypted tablespace

When creating a tablespace, the tablespace can be set as encrypted by assigning an encryption algorithm attribute to it. The keystore must be opened to create encrypted tablespaces. Here are the three different ways to assign an encryption algorithm:

- Configuration file: Set up a `tablespace_encryption_algorithm` in the `postgresql.conf` file, which is valid for all tablespace creations across all databases.
- Session variable: Use the **`SET`** command to assign an encryption algorithm, which is valid for all tablespace creations until the encryption algorithm is reset. This variable overrides any encryption algorithm assignment that was set by the configuration file, if any.
- Create tablespace command: Use the encryption algorithm attribute of the **`CREATE TABLESPACE SQL`** command, which is valid only for this single tablespace. The command overrides all other encryption algorithm assignments, if any.

Example 4-8 demonstrates the commands that:

1. Create a tablespace.
2. Set the algorithm to `none`. The value `none` is accepted in the encryption algorithm input, which means the tablespace is a non-encrypted tablespace.
3. Create a second tablespace.
4. Create a third tablespace and set the algorithm to `AES128`.

Example 4-8 Creating a tablespace without and with encryption

```
1 postgres=# create tablespace securetblspc1 location
'/home/fsepuser/securetbl_spc1_dir';
CREATE TABLESPACE
2 postgres=# set tablespace_encryption_algorithm=none;
SET
3 postgres=# create tablespace nomaltblspc location
'/home/fsepuser/nomaltblspc_dir';
CREATE TABLESPACE
4 postgres=# create tablespace securetblspc2 location
'/home/fsepuser/securetbl_spc2_dir' with
(tablespace_encryption_algorithm='AES128');
CREATE TABLESPACE
```

Two advanced encryption standards (AES 128 and AES 256) can be used to create encrypted tablespaces. Use the command that is shown in Example 4-9 to determine which encryption algorithm is being used.

Example 4-9 Determining which algorithm is being used for encryption

```
postgres=# show tablespace_encryption_algorithm;
 tablespace_encryption_algorithm
---------------------------------
AES256
(One row)
```

As shown in Example 4-8, data that is stored inside the created encrypted tablespaces `securetblspc1` and `securetblspc2` is stored in encrypted format.

It is also possible to alter an existing tablespace from non-encrypted to encrypted, but only when the tablespace is empty, as shown in Example 4-10.

Example 4-10 Altering a non-encrypted and empty tablespace to an encrypted tablespace

```
postgres=# alter tablespace tblspc set (tablespace_encryption_algorithm='AES256');
ALTER TABLESPACE
```

Viewing the list of encrypted tablespaces

Details about the encrypted tablespaces and the encryption algorithm that is used for each encrypted tablespace can be viewed by using the `pgx_tablespaces` catalog table, as shown in Example 4-11.

Example 4-11 Listing the encrypted tablespaces in a database

```
postgres=# select spcname,spctablespace,spcencalgo from pg_tablespace as
pgt,pgx_tablespaces as pgxt where pgt.oid=pgxt.spctablespace;
    spcname    | spctablespace | spcencalgo
---------------+---------------+-------------
 pg_default    |          1663 | none
 pg_global     |          1665 | none
 securetblspc1 |         16384 | AES256
 normaltblspc  |         16385 | none
 securetblspc2 |         16386 | AES128
(5 row)
```

4.3.7 Periodic operations

Certain periodic operations are required to maintain the security of database encryption and to comply with security standards such as PCI DSS.

Rotation of the master encryption key

Periodic rotation of the TDE MEK is a best practice of key management. NIST special publication 800-5 recommends changing the TDE master key once a year.

The TDE master key can be rotated by using the same SQL command that was used to create the TDE master key. When run the second time onwards, FUJITSU Enterprise Postgres internally rotates the TDE master key. All the tablespace keystore files are reencrypted by using a new TDE master key. Because the TDE keys are not changed, rotation of the TDE master key does not trigger massive amounts of data reencryption. The TDE master key rotation process is relatively fast, and it is based on the number of encrypted tablespaces that exist in the database.

The TDE master key rotation process also updates the auto-keystore, if one exists, with a new TDE master key.

In Example 4-12, we run a command before the TDE master key rotation to check the auto-keystore file.

Example 4-12 Checking the auto-keystore

```
[fsepuser@rdbkkpgr1 ~]$ ls -l  keystore_dir/
total 8
-rw-------. 1 fsepuser fsepuser 928 Oct 19 21:45 keystore.aks
-rw-------. 1 fsepuser fsepuser 928 Oct 20 19:57 keystore.ks
```

We issue the same SQL command that was used for creating the TDE master key to rotate it, as shown in Example 4-13.

Example 4-13 Rotating the MEK file

```
[fsepuser@rdbkkpgr1 ~]$ psql postgres
psql (12.1)
Type "help" for help.
```

```
postgres=# select pgx_set_master_key('12345678');
 pgx_set_master_key
--------------------
(One row)
```

After key rotation, we check that the master key was updated (Example 4-14). Note the different date and timestamp on the files.

Example 4-14 Checking a master keystore update

```
postgres=# \q
[fsepuser@rdbkkpgr1 ~]$ ls -l  keystore_dir/
total 8
-rw-------. 1 fsepuser fsepuser 928 Oct 20 20:00 keystore.aks
-rw-------. 1 fsepuser fsepuser 928 Oct 20 20:00 keystore.ks
```

What happens if the TDE master key rotation is interrupted

TDE master key rotation is a crucial process that is fenced in FUJITSU Enterprise Postgres. There is always a short amount of time between replacing a new TDE master key and reencrypted tablespace keystores and writing it to disk. If this process is interrupted during this window, the tablespace keystores, which are not reencrypted, are still valid and can use the old rotated TDE master key to decrypt its TDE key (see Figure 4-3).

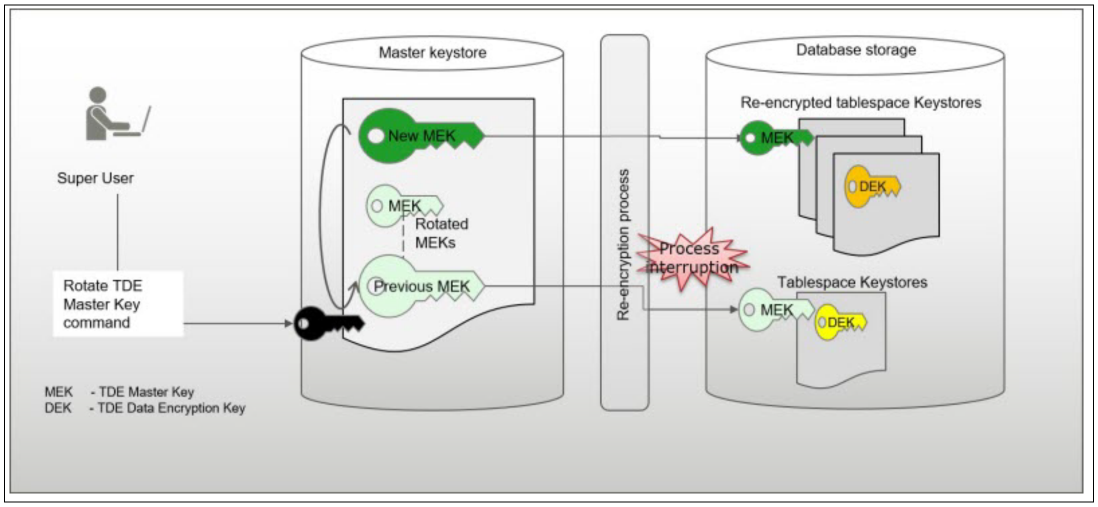


Figure 4-3 Interrupting the TDE master key rotation

4.3.8 Changing the keystore passphrase

A passphrase is a key element of protecting the TDE master key. As a best practice, change the passphrase periodically and whenever it is exposed to risks.

The keystore passphrase can be changed through an SQL command, as shown in Example 4-15.

Example 4-15 Command to change keystore passphrase

```
postgres=# select pgx_set_keystore_passphrase('12345678','87654321');
 pgx_set_keystore_passphrase
-----------------------------
```

```
(One row)
```

The first parameter of the SQL command requires the old passphrase, and the second parameter sets the new passphrase. The command updates the passphrase only when the correct old passphrase is passed.

Changing the keystore passphrase does not change the MEK, and only the master keystore is reencrypted by using the new passphrase. So, this command execution is faster than a MEK rotation.

4.3.9 Backing up the keystore

The master keystore is in a pre-configured keystore location and this file must be copied separately in addition to a database backup. The master keystore is not backed up as a standard part of database backup software. For more information about how to back up and restore a database, see Chapter 9, "Backup, recovery, and monitoring" on page 311.

- When is a keystore backup required?

  Whenever a database backup is taken.
- When is a keystore backup recommended?
  - When the TDE master key is first configured.
  - When the TDE master key is rotated.
  - When the keystore passphrase is changed.

4.3.10 Configuring the standby server to use TDE

When setting up a standby server, the master keystore should be copied from the primary server to the keystore location that is configured in the standby server. No other steps are required to use TDE with a standby server.

The standby server must be configured to use the same TDE master key as the primary server for the following basic reasons:

- A standby server that is configured from the backup of a primary server contains all encrypted tablespace data that is copied from the primary server. Decryption of such data requires the exact same TDE master key.
- After the standby server is configured, the primary server sends the WAL files/records that correspond to the encrypted tablespace to the standby server with encrypted data in it. The standby server, when reading this data, should decrypt it with the same TDE master key.

4.4 FUJITSU Enterprise Postgres encryption on IBM LinuxONE with crypto cards

In this section, we describe the steps that are used to configure and use encryption for database data on a FUJITSU Enterprise Postgres 12 server with Red Hat Enterprise Linux Server 8 on IBM LinuxONE with crypto cards available.

This section covers the following topics:

- Overview
- Architectural overview
- Extra hardware and software requirements/Pre-requisites
- Configuring TDE to use crypto cards
- Enabling encryption
- Managing the keystore
- Configuring the standby server to enable encryption by using crypto cards

4.4.1 Overview

TDE protects data that is stored in the OS file system from attackers who bypass the database server's authentication and access controls.

The IBM LinuxONE platform can offer more secure and performant *hardware-based* cryptography.

For more information about FUJITSU Enterprise Postgres TDE on IBM LinuxONE with crypto cards, see Protecting Storage Data Using Transparent Data Encryption.

4.4.2 Benefits of using TDE on IBM LinuxONE

All the benefits of using standard TDE that are listed in 4.3.2, "Benefits of using TDE" on page 118 also are applicable to TDE on IBM LinuxONE.

In addition, security is further enhanced by using IBM LinuxONE crypto hardware support. FUJITSU Enterprise Postgres uses the HSM of Crypto Express cards for TDE MEK management. The HSM is a special crypto hardware feature that is tamperproof and contains an unextractable secret master key.

FUJITSU Enterprise Postgres uses a secure key that is encrypted by the HSM MEK, which cannot be used inside the OS as the TDE master key for encryption and decryption of TDE DEKs that are managed in the database.

4.4.3 Architectural overview

TDE on IBM LinuxONE uses an envelope encryption strategy that encrypts plain text data with a data key and then encrypts the data key with another key. All data within a specified tablespace is encrypted by TDE DEKs, and the TDE DEKs are encrypted by the TDE master key. FUJITSU Enterprise Postgres stores the TDE MEK ID that points to an openCryptoki token directory object. The TDE master key that is used in the database is a *secure* key that is a TDE master key that is wrapped by the HSM MEK.

Encryption and decryption of a tablespace keystore is done through the Crypto Express card hardware unit.

TDE DEKs cannot be unwrapped without the TDE master key. Deleting the TDE master key means that all encrypted data in the database cluster will be crypto-shredded.

Figure 4-4 provides a diagram of this architecture.

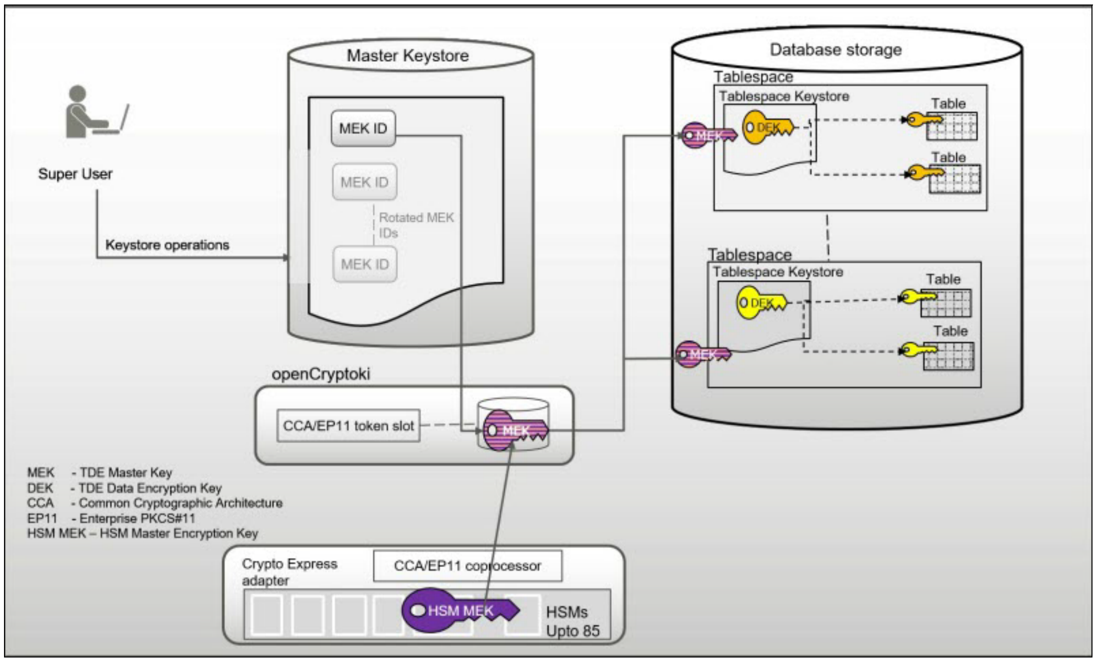


Figure 4-4 TDE on IBM LinuxONE architectural diagram

TDE on IBM LinuxONE uses the IBM Z crypto stack for TDE master key operations.

Figure 4-5 on page 129 shows the process communication between FUJITSU Enterprise Postgres and Crypto Express cards.

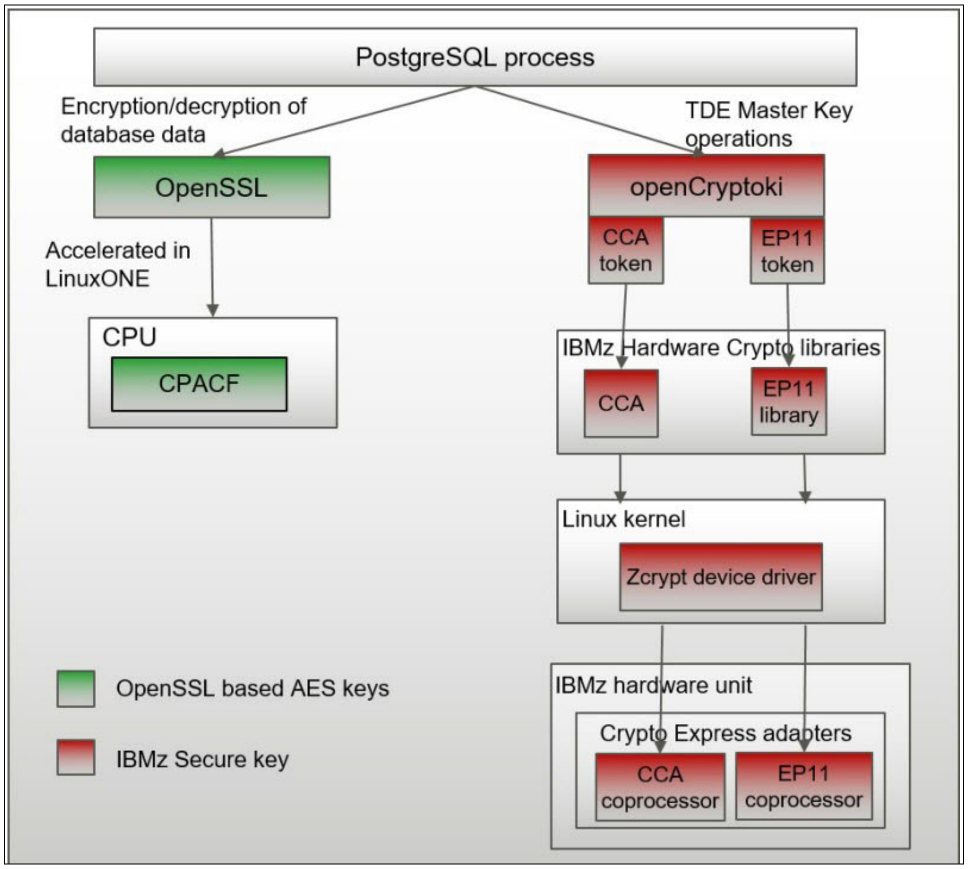


Figure 4-5 Process communication between FUJITSU Enterprise Postgres and Crypto Express

4.4.4 Prerequisites

FUJITSU Enterprise Postgres requires two prerequisite configurations:

1. Crypto Express card configuration
2. An openCryptoki configuration

FUJITSU Enterprise Postgres TDE on IBM LinuxONE uses the IBM Crypto Express card in CCA coprocessor or EP11 mode. Each Crypto Express card contains an array of HSMs, and FUJITSU Enterprise Postgres uses a single HSM unit in the Crypto Express card through openCryptoki software.

Before starting the database server, the Crypto Express cards should be configured in either CCA coprocessor mode or EP11 mode. For more information about Crypto Express cards, see IBM CryptoCards.

All the installed Crypto Express cards and HSM domains can be verified by using the **lszcrypt** command (see Example 4-16).

Example 4-16 Listing the Crypto Express cards

```
[fsepuser@rdbkpgr5 ~]$ lszcrypt -V
CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH FUNCTIONS  DRIVER
-----------------------------------------------------------------------------------
00          CEX7C CCA-Coproc  online        84        0     13     08 S--D--NF-  cex4card
00.0024     CEX7C CCA-Coproc  online        84        0     13     08 S--D--NF-  cex4queue
```

```
01          CEX7A Accelerator online        18        0     13     08 -MC-A-NF-  cex4card
01.0024     CEX7A Accelerator online        18        0     13     08 -MC-A-NF-  cex4queue
02          CEX7C CCA-Coproc  online        44        0     13     08 S--D--NF-  cex4card
02.0024     CEX7C CCA-Coproc  online        44        0     13     08 S--D--NF-  cex4queue
03          CEX7C CCA-Coproc  online        44        0     13     08 S--D--NF-  cex4card
03.0024     CEX7C CCA-Coproc  online        44        0     13     08 S--D--NF-  cex4queue
04          CEX7A Accelerator online        18        0     13     08 -MC-A-NF-  cex4card
04.0024     CEX7A Accelerator online        18        0     13     08 -MC-A-NF-  cex4queue
05          CEX7P EP11-Coproc online       404        0     13     08 -----XNF-  cex4card
05.0024     CEX7P EP11-Coproc online       404        0     13     08 -----XNF-  cex4queue
06          CEX7C CCA-Coproc  online        44        0     13     08 S--D--NF-  cex4card
06.0024     CEX7C CCA-Coproc  online        44        0     13     08 S--D--NF-  cex4queue
07          CEX7P EP11-Coproc online       332        0     13     08 -----XNF-  cex4card
07.0024     CEX7P EP11-Coproc online       332        0     13     08 -----XNF-  cex4queue
```

A Crypto Express card and domain ID pair is referred as an APQN. One APQN is used to generate a secure TDE master key for the FUJITSU Enterprise Postgres Database. This APQN should be configured in the openCryptoki software.

You can install the openCryptoki software by using supported OS repositories such as yum. For a more information about openCryptoki, see the EP11 crypto stack.

An openCryptoki configuration should be done by using the `openCryptoki.conf` file. This configuration file, by default, contains various slots for all available tokens in the system. Configure the slot with token details (CCA/EP11) for FUJITSU Enterprise Postgres. To configure the slot, you must pass the `STDLL` value that corresponds to the token type. The `STDLL` value is referred to in the set of slot details that are available by default in the configuration file. Additionally, set the `tokname` attribute of the slot.

Users must be in the `pkcs11` group to use openCryptoki. The FUJITSU Enterprise Postgres instance administrator should be added to the `pkcs11` group.

To configure an IBM CCA token, complete the following steps:

1. Edit the openCryptoki configuration file, `opencryptoki.conf`, to add the CCA token to a new slot, as shown in Example 4-17.

Example 4-17 Adding the CCA token to a new slot

```
[root@rdbkpgr4 fsepuser]# vi /etc/opencryptoki/opencryptoki.conf
slot 5
{
stdll = libpkcs11_cca.so
tokname = ccatoken_fep
}
[root@rdbkpgr4 fsepuser]# systemctl restart pkcsslotd.service
```

2. Initialize the token that you configured in slot 5 of the openCryptoki configuration file, as shown in Example 4-18.

Example 4-18 Initializing the token

```
[root@rdbkpgr4 fsepuser]# pkcsconf -I -c 5
Enter the SO PIN:
Enter a unique token label: cca_for_fep
```

3. Check the token status, as shown in Example 4-19 on page 131.

Example 4-19 Checking the token status

```
[root@rdbkpgr4 fsepuser]# pkcsconf –t
Token #5 Info:
        Label: cca_for_fep
        Manufacturer: IBM Corp.
        Model: IBM CCA Token
        Serial Number: 123
        Flags: 0x880445
(RNG|LOGIN_REQUIRED|CLOCK_ON_TOKEN|TOKEN_INITIALIZED|USER_PIN_TO_BE_CHANGED|SO_PIN
_TO_BE_CHANGED)
        Sessions: 0/18446744073709551614
        R/W Sessions: 18446744073709551615/18446744073709551614
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Hardware Version: 1.0
        Firmware Version: 1.0
        Time: 2020111521383700
```

4. Change the User PIN, as shown in Example 4-20.

Example 4-20 Changing the user PIN

```
[root@rdbkpgr4 fsepuser]# pkcsconf -u -c 5
Enter the SO PIN:
Enter the new user PIN:
Reenter the new user PIN:
```

5. Change the SO PIN, as shown in Example 4-21.

Example 4-21 Changing the SO PIN

```
[root@rdbkpgr4 fsepuser]# pkcsconf -P -c 5
Enter the SO PIN:
Enter the new SO PIN:
Reenter the new SO PIN:
Step-6: Check the token status
[root@rdbkpgr4 fsepuser]# pkcsconf –t
Token #5 Info:
        Label: cca_for_fep
        Manufacturer: IBM Corp.
        Model: IBM CCA Token
        Serial Number: 123
        Flags: 0x44D
(RNG|LOGIN_REQUIRED|USER_PIN_INITIALIZED|CLOCK_ON_TOKEN|TOKEN_INITIALIZED)
        Sessions: 0/18446744073709551614
        R/W Sessions: 18446744073709551615/18446744073709551614
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Hardware Version: 1.0
        Firmware Version: 1.0
        Time: 2020111521420300
```

For an EP11 token type, more configuration is required and can be passed through the configuration file that is assigned to the **confname** attribute of the slot. The EP11 configuration file should show the APQN number to use. The APQN number can be specified in decimal or hexadecimal format.

Example 4-22 provides an example configuration for the EP11 token type.

Example 4-22 EP11 configuration

```
[root@rdbkpgr5 ~]# vi /etc/opencryptoki/opencryptoki.conf
slot 5
{
stdll = libpkcs11_ep11.so
confname = ep11tok.conf
tokname = ep11token01
}
[root@rdbkpgr5 ~]# vi /etc/opencryptoki/ep11tok.conf
# Decimal representation of ‘lszcrypt –v’ command output CARD.DOMAIN ‘05.0024’
APQN_WHITELIST
5   36
END
[root@rdbkpgr5 ~]# systemctl restart  pkcsslotd.service
[root@rdbkpgr5 ~]#  pkcsconf -I -c 5
Enter the SO PIN:
Enter a unique token label: EP11_for_FEP
[root@rdbkpgr5 ~]#  pkcsconf -t
Token #5 Info:
        Label: EP11_for_FEP
        Manufacturer: IBM Corp.
        Model: IBM EP11Tok
        Serial Number: 93AABDYU38535022
        Flags: 0x880445
(RNG|LOGIN_REQUIRED|CLOCK_ON_TOKEN|TOKEN_INITIALIZED|USER_PIN_TO_BE_CHANGED|SO_PIN
_TO_BE_CHANGED)
        Sessions: 0/18446744073709551614
        R/W Sessions: 18446744073709551615/18446744073709551614
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Hardware Version: 7.22
        Firmware Version: 3.1
        Time: 2020111518251900
[root@rdbkpgr5 ~]#  pkcsconf -P -c 5
Enter the SO PIN:
Enter the new SO PIN:
Reenter the new SO PIN:
[root@rdbkpgr5 ~]#  pkcsconf -u -c 5
Enter the SO PIN:
Enter the new user PIN:
Reenter the new user PIN:
 [root@rdbkpgr5 fsepuser]# usermod -G pkcs11 fsepuser
[fsepuser@rdbkpgr5 ~]$  pkcsconf –t
Token #5 Info:
        Label: EP11_for_FEP
        Manufacturer: IBM Corp.
        Model: IBM EP11Tok
        Serial Number: 93AABDYU38535022
        Flags: 0x44D
(RNG|LOGIN_REQUIRED|USER_PIN_INITIALIZED|CLOCK_ON_TOKEN|TOKEN_INITIALIZED)
        Sessions: 0/18446744073709551614
        R/W Sessions: 18446744073709551615/18446744073709551614
        PIN Length: 4-8
        Public Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Private Memory: 0xFFFFFFFFFFFFFFFF/0xFFFFFFFFFFFFFFFF
        Hardware Version: 7.22
        Firmware Version: 3.1
        Time: 2020111521452200
```

The EP11 configuration file should be in the same place as the openCryptoki configuration file.

If required, the EP11 configuration file can be stored in a different directory location, but you must set the environment variable **`OCK_EP11_TOKEN_DIR`** to point to the directory location where the EP11 configuration files are stored.

For more information about configuring openCryptoki for the EP11 token type, see Configuring openCryptoki for EP11 support.

4.4.5 Configuring TDE on IBM LinuxONE

In this section, we describe the configuration parameters that are needed to use TDE on IBM LinuxONE with FUJITSU Enterprise Postgres. We also describe the steps to create a TDE master key.

Configuration parameters

To use TDE on IBM LinuxONE, the FUJITSU Enterprise Postgres Database server should be configured with the parameters that are listed in Table 4-2. These parameters are found in the configuration file `postgresql.conf`.

Table 4-2 TDE on IBM LinuxONE configuration parameters

Configuration parameter	Optional/Mandatory	Description
`shared_preload_libraries`	Mandatory	Specify `tde_z` in the **`shared_preload_libraries`** configuration.
`tde_z.SLOT_ID`	Mandatory	Specify which openCryptoki slot to use for TDE on IBM LinuxONE.
`tde_z.USER_PIN`	Optional	Specify `userPIN` to log in to the token that is configured in the openCryptoki slot for TDE on IBM LinuxONE. This parameter is optional, and it enables the system to recover automatically in encryption/decryption enabled mode. There are other ways to specify `userPIN` for users who prefer more security. For more information, see “How to open the keystore” on page 136.

Configuration parameter	Optional/Mandatory	Description
tde_z.IBM_CCA_CSU_DEFAULT_ADAPTER	Optional	Specify this parameter to set the CCA environment variable **CSU_DEFAULT_ADAPTER**. This parameter is used to specify the adapter number to use for the CCA token. It accepts a decimal value as the adapter number. The **lszcrypt** command shows the CCA card configuration in hexadecimal notation. Convert it to decimal plus 1 to set the adapter number.
,tde_z.IBM_CCA_CSU_DEFAULT_DOMAIN	Optional	Specify this parameter to set the IBM LinuxONE environment variable **CSU_DEFAULT_DOMAIN**. This parameter is used to specify the domain number to use for the CCA token. It accepts a decimal value as the domain number. For more information about **CS_DEFAULT_ADPATER** and **CSU_DEFAULT_DOMAIN**, see CCA of Version 6.0.
keystore_location	Mandatory	File system directory to store the master keystore file. This location should be specific to a database cluster.
tablespace_encryption_algorithm	Optional	Configure this parameter in postgresql.conf file only when all further tablespaces by default should be encrypted. AES256, AES128, and none are the available encryption algorithm options. You can also configure an encryption algorithm for each tablespace at tablespace creation time.

Example 4-23 provides sample settings for CCA token-specific configuration parameters.

Example 4-23   Settings of CCA token-specific configuration parameters

```
tde_z.IBM_CCA_CSU_DEFAULT_ADAPTER = 'CRP01'
# PKCS11 environment variable CSU_DEFAULT_ADAPTER for TDEz of CCA configuration
#
tde_z.IBM_CCA_CSU_DEFAULT_DOMAIN = '34'
# PKCS11 environment variable CSU_DEFAULT_DOMAIN for TDEz of CCA configuration
```

For database servers that already are started, changes to configurations become effective only after restarting the database server.

Example 4-24 provides an example of the configuration parameters set in the postgresql.conf file.

Example 4-24   Settings of the configuration parameters

```
# - Shared Library Preloading -
shared_preload_libraries = 'tde_z'      # (change requires restart)
```

```
keystore_location = '/home/fsepuser/keystoredir'
# TDEz parameters are for crypto card secure key support on IBM Z platform
# shared_preload_libraries must include 'tde_z'
#
tde_z.SLOT_ID = 5
# PKCS11 A Slot ID as defined in opencryptoki.conf
#
tde_z.USER_PIN = '1234'
```

Use the command that is shown in Example 4-25 to restart the database server.

Example 4-25 Command to restart the database server

```
[fsepuser@rdbkpgr5 ~]$ pg_ctl -D ./testdb/ -l logfile restart
waiting for server to shut down.... done
server stopped
waiting for server to start.... done
server started
```

4.4.6 Creating a TDE master key

To start using encryption, you must create a TDE master key. This task must be performed by a database superuser running an SQL command against the database server.

The TDE master key is an internally generated *secure key* that is not stored anywhere in plain text format. The key object is stored in the openCryptoki token directory, and the ID to the key object is stored in the master keystore of the FUJITSU Enterprise Postgres server.

You must pass the `userPIN` argument when creating the TDE master key regardless whether this value already was passed to the FUJITSU Enterprise Postgres server (Example 4-26).

Example 4-26 Creating the TDE master key

```
[fsepuser@rdbkpgr5 ~]$ psql postgres
psql (12.1)
Type "help" for help.
postgres=# select pgx_set_master_key('1234');
 pgx_set_master_key
--------------------
(One row)
```

The TDE master key creation command also opens the keystore so that you can start creating encrypted tablespaces.

Note: The master keystore file is not included in a database backup and must be copied separately. In addition, the openCryptoki token directory should also be copied. The same `userPIN` that is effective during a backup should be used during recovery to open the keystore.

4.4.7 Enabling encryption

To prepare a database for encryption and decryption, the master keystore must be opened after every database server restart. We describe how to do this task in this section.

How to open the keystore

The master keystore (keystore) can be opened through various methods. Based on the preference and needs of the user, any of the following options can be used:

- Configuration settings

  You can specify `tde_z.USER_PIN` in the `postgresql.conf` configuration file. When this setting is in place, then the database automatically starts in encryption enabled mode. You can start using encryption or decryption when the database server is started without any extra commands.

- Command-line argument in database server start

  To start the database server in encryption/decryption enabled mode, you can specify `userPIN` to open the master keystore by using the **`–user-pin`** option that is specified in the **`pg_ctl`** command. This option prompts you for input when starting the database server. Example 4-27 shows the command-line arguments that can be used to start a database server.

Example 4-27 Command-line arguments to open a keystore during a server start

```
[fsepuser@rdbkpgr5 ~]$ pg_ctl -D ./testdb/ -l logfile --user-pin start
Enter User PIN:
waiting for server to start.... done
server started
```

  You might use this option when the database server is starting in recovery mode and WAL records that are getting applied contain encrypted data.

  To start the database server in encryption or decryption enabled mode, specify `userPIN` to open the master keystore by using the **`--user-pin`** option in the **`pg_ctl`** command. This option prompts for user input when starting the database server. This option also means that user intervention is needed to enter the `userPIN`.

- SQL command

  While the database server is running, the keystore can be opened by running an SQL command. Example 4-28 shows the command to open a keystore.

Example 4-28 SQL command to open a keystore

```
postgres=# select pgx_open_keystore('1234');
 pgx_open_keystore
------------------
(One row)
```

Keystores can be opened only by users that have the database superuser privilege.

Important: You must remember the `userPIN` because it cannot be recovered by any other means.

4.4.8 Creating encrypted tablespaces

When creating a tablespace, the tablespace can be set as encrypted by assigning an encryption algorithm attribute to it. The keystore must be opened to create encrypted tablespaces. Here are the three different ways to assign an encryption algorithm, which are identical to the ones that are described in “Creating an encrypted tablespace” on page 122:

- Configuration file: Set `tablespace_encryption_algorithm` in the `postgresql.conf` file. Valid for all tablespace creations across all databases.
- Session variable: Use the **`SET`** command to assign an encryption algorithm. Valid for all tablespace creations until the encryption algorithm is reset. Overrides encryption algorithm assignment that is set through the configuration file, if any.
- Create tablespace command: Use the encryption algorithm attribute of the **`CREATE TABLESPACE`** SQL command. Valid only for this single tablespace. Overrides all other encryption algorithm assignments, if any. This command is identical to the one in Example 4-8 on page 123, and is demonstrated in Example 4-29 with a different tablespace name and directory, and using the AES128 encryption standard. After running the command, any data that is stored in this tablespace is stored in an encrypted fashion.

Example 4-29 Creating an encrypted tablespace

```
postgres=# create tablespace securetblspc location '/home/fsepuser/tblspcdir' with
(tablespace_encryption_algorithm='AES128');
CREATE TABLESPACE
```

4.4.9 Viewing the list of encrypted tablespaces

Details about the encrypted tablespaces and the encryption algorithm that is used for each encrypted tablespace can be viewed by using the `pgx_tablespaces` catalog table. Possible values of encryption algorithm are `none`, `AES128`, and `AES256`. To view the tablespaces and their encryption values, run the command that is shown in Example 4-30.

Example 4-30 Listing the encrypted tablespaces in a database

```
postgres=# select spcname, spctablespace, spcencalgo from pg_tablespace as
pgt,pgx_tablespaces as pgxt where pgt.oid=pgxt.spctablespace;
   spcname    | spctablespace | spcencalgo
--------------+---------------+------------
 pg_default   |          1663 | none
 pg_global    |          1664 | none
 securetblspc |         16385 | AES256
 normaltblspc |         16386 | AES256
(4 row)
```

4.4.10 Managing the keystore

In this section, we describe necessary periodic operations, changing the keystore passphrase, and how to back up a keystore.

Periodic operations

Certain periodic operations are required to maintain the security of database encryption and comply with security standards such as PCI DSS.

Rotating the TDE master key

This operation is identical to in "Rotation of the master encryption key" on page 124. When using TDE on IBM LinuxONE, the user must pass the `userPIN` value to the **`pgx_set_master_key`** command. Example 4-31 provides an example of the sequence of commands that is used to rotate the master encryption file and check the master keystore update.

Example 4-31 Commands to rotate the master encryption file

```
[fsepuser@rdbkpgr5 ~]$ ls -l keystoredir/
total 4
rw-------. 1 fsepuser fsepuser 928 Nov 15 21:55 keystore.ks
[fsepuser@rdbkpgr5 ~]$ psql postgres
psql (12.1)
Type "help" for help.
postgres=# select pgx_set_master_key('1234');
 pgx_set_master_key
-------------------
(One row)
postgres=# \q
[fsepuser@rdbkpgr5 ~]$ ls -l keystoredir/
total 4
rw-------. 1 fsepuser fsepuser 928 Nov 15 22:02 keystore.ks
```

Changing the userPIN

The `userPIN` is a key element of protecting MEK storage. It is a best practice that you change the `userPIN` whenever it is exposed. The `userPIN` should be changed by running the openCryptoki **`pkcsconf`** command. Any previously opened database sessions can still run and are not interrupted by a `userPIN` change. However, for any new sessions, the new `userPIN` should be used to log in to an openCryptoki token. So, after changing the `userPIN`, it is a best practice to reopen the keystore in FUJITSU Enterprise Postgres to use the new `userPIN`.

The new `userPIN` can be passed to FUJITSU Enterprise Postgres through the **`pgx_open_kesytore`** SQL command, as shown in Example 4-32.

Example 4-32 Changing userPIN and passing it to the database server

```
[fsepuser@rdbkpgr5 ~]$ pkcsconf -p -c 5
Enter user PIN:
Enter the new user PIN:
Reenter the new user PIN:
[fsepuser@rdbkpgr5 ~]$ psql postgres
psql (12.1)
Type "help" for help.
postgres=# select pgx_open_keystore('12345');
 pgx_open_keystore
------------------
(One row)
```

Changing the `userPIN` does not change the TDE master key. This command execution is faster than performing a TDE master key rotation.

Rotating the HSM master encryption key

There can be periodic rotation of the HSM MEK depending on your company's policies. For more information about how to migrate to a new HSM MEK on a Crypto Express adapter, see Migrating master keys with the pkcsep11_migrate tool.

Impact on FUJITSU Enterprise Postgres

In both CCA and EP11 HSM MEK migrations, all the process that are using openCryptoki should be stopped before the migration process. The FUJITSU Enterprise Postgres server process should also be stopped during this entire migration process. You should also take a backup of the openCryptoki token directory before migration.

The openCryptoki token directory is created with its name stored in the `tokname` attribute of the openCryptoki slot configuration. Example 4-33 shows the command to back up an openCryptoki token directory.

Example 4-33 Command to back up an openCryptoki token directory

```
[fsepuser@rdbkpgr5 opencryptoki]$ vi /etc/opencryptoki/opencryptoki.conf
..
slot 5
{
stdll = libpkcs11_ep11.so
confname = ep11tok.conf
tokname = ep11token01
}
[fsepuser@rdbkpgr5 ~]$ tar -cf token_directory_Nov162020_ep11token01.tar
/var/lib/opencryptoki/ep11token01/
```

Upon completion of the HSM MEK rotation process, FUJITSU Enterprise Postgres can start a backup with no extra steps required.

Extra backup requirements

When using TDE on IBM LinuxONE, in addition to standard backup files that are taken through backup commands such as **`pgx_dmpall`** and **`pg_basebackup`**, the following storage should also be copied separately:

- Master keystore: Located in the configured keystore location. This file must be copied in separately and in addition to the database backup.
- The openCryptoki token directory: Located in the openCryptoki token storage with the directory name stored in the `tokname` attribute of the openCryptoki slot configuration.

For more information about how to back up and restore databases, see Chapter 9, "Backup, recovery, and monitoring" on page 311.

In addition, consider the following items:

- When are a keystore and openCryptoki token directory backup required?

  You should back up the keystore and openCryptoki token directory whenever a database backup is performed.
- When are the keystore and openCryptoki token directory backup recommended?
  - When the TDE master key is first configured.
  - Whenever the TDE master key is rotated.

– When the userPIN of the openCryptoki token is changed.
– Whenever the HSM MEK is rotated.

Important: There is no way to recover the `userPIN` that is configured for the corresponding openCryptoki token directory backup, so the user *must* remember this `userPIN`.

4.4.11 Configuring a standby server to use TDE on IBM LinuxONE

Here are the prerequisites when setting up the standby (also known as the secondary) server to use TDE on IBM LinuxONE:

1. The HSM MEKs that are configured to a standby server should be the same as the HSM MEK that is used on the primary server. There might be extra hardware or software requirements to load the same master key parts into the domains of the Crypto Express adapters that are assigned to FUJITSU Enterprise Postgres server usage.
2. Copy the master keystore file from the primary server to the secondary server. The location should point to the `tokname` attribute of the secondary server's `postgresql.conf` file.
3. Back up the openCryptoki token directory corresponding to the slot that is used by FUJITSU Enterprise Postgres.
4. On the secondary server, perform the followings steps that are related to an openCryptoki configuration:
   a. Update the `openCryptoki.conf` file with a new slot with the `tokname` attribute to use with FUJITSU Enterprise Postgres. This slot should point to the type of token (either CCA or EP11). It must be the same as the primary server. The adapter and domain mapping of the token should be to the HSM that is loaded with the same master key parts as the primary server.
   b. Restart the `pkcsslotd` daemon process of openCryptoki to enable the openCryptoki configuration changes.
   c. Place the openCryptoki token directory backup that was taken inside the secondary server's openCryptoki token directory that corresponds to the slot that is configured for FUJITSU Enterprise Postgres. A new directory with `tokname` is listed in the openCryptoki token directory and is the location to place the copy of backup contents. The PIN and `userPIN` are the same as the ones that were configured in the primary server.
   d. Remove any shared memory files in `/dev/shm` that are associated with the new `tokname` if any exists, and restart the `pkcsslotd` daemon process.

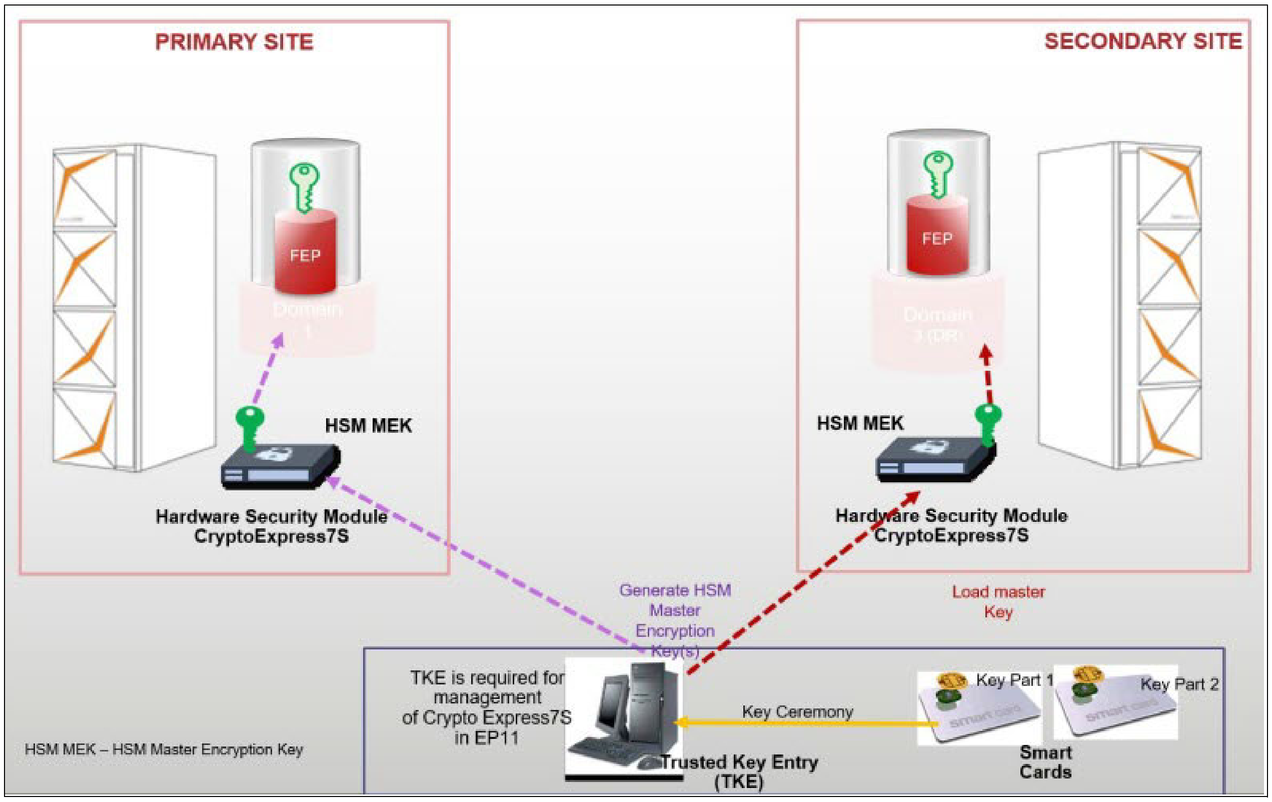


Figure 4-6 Configuring the standby server with the same HSM MEK

The standby server must be configured to use the same TDE master key as the primary server for the following reasons:

- The standby server that is configured from a backup of a primary server contains all the encrypted tablespace data that is copied from the primary server. Decryption of this data requires the exact same TDE master key.
- After the standby server is configured, the primary server sends the WAL files and records that correspond to the encrypted tablespace to the standby server with the encrypted data in it. The standby server, when reading this data, should decrypt it with same TDE master key.

4.5 Data Masking feature

In this section, we provide the steps to configure and use the Data Masking feature to mask sensitive data on a FUJITSU Enterprise Postgres 12 server with Red Hat Enterprise Linux Server 8 on IBM LinuxONE.

This section covers the following topics:

- Overview
- Architectural overview
- Configuring data masking
- How to use data masking

4.5.1 Overview

FUJITSU Enterprise Postgres Data Masking enables sensitive data that stored in the database to be extracted and redacted before sending it to any application. Users can govern access to sensitive data by using the embedded Data Masking feature.

The Data Masking feature is available by default with FUJITSU Enterprise Postgres as an extension.

4.5.2 Benefits of using data masking

Sensitive data such as credit card numbers or salaries of employees that are stored in database tables can be protected selectively by creating a masking policy. Data masking can restrict access to sensitive original data to certain user groups by obfuscating the original data. Authorized users can still view the original data. It provides dynamic masking of sensitive data.

- Data masking provides facilities for protecting both online and offline data.
- It takes a flexible, easy to use policy-based approach.
- Data masking helps minimize storage costs because your enterprise can use the same set of data for many purposes, such as testing or running *ad hoc* analytical queries without compromising the security of sensitive data. Data masking allows you to share only the information that is required for applications.

4.5.3 Architectural overview

Data masking obfuscates data in a runtime. During SQL processing, based on the masking policies that are applicable to querying resultant data, the original data is redacted and sent as output while the referential integrity of the data is maintained. An overview of the architecture of the Data Masking feature is provided in Figure 4-7.

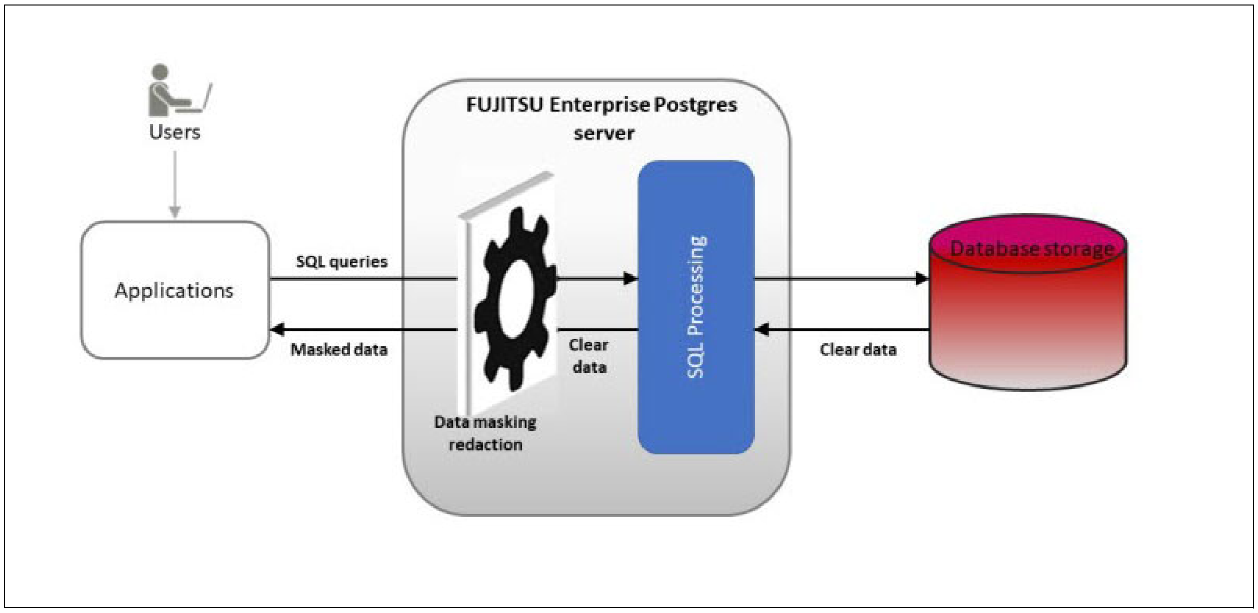


Figure 4-7 Data Masking architecture

4.5.4 Configuration parameters for the Data Masking feature

To use Data Masking, the FUJITSU Enterprise Postgres Database server should be configured with the parameter that is shown in Example 4-34 in the configuration file `postgresql.conf`. In **`shared_preload_libraries`**, '`pgx_datamasking`' must be the first member of this list. Only users with superuser privileges can configure this parameter.

Example 4-34 Shared library preloading

```
# - Shared Library Preloading -
shared_preload_libraries = 'pgx_datamasking'     # (change requires restart)
```

For database servers that already are started, this change becomes effective only after restarting the database server. Example 4-35 demonstrates the command to restart the database server.

Example 4-35 Command to restart the database server

```
[fsepuser@rdbkpgr1 ~]$ ]$ pg_ctl -D ./testdb/ -l logfile restart
waiting for server to shut down..... done
server stopped
waiting for server to start..... done
server started
```

4.5.5 Creating the Data Masking extension

To start using the Data Masking feature, the `pgx_datamasking` extension must be created. This task must be performed by a database superuser by running the SQL command that is shown in Example 4-36 against the database server.

Example 4-36 Command to create the Data Masking extension

```
postgres=# create extension pgx_datamasking;
CREATE EXTENSION
```

4.5.6 Enabling and disabling the Data Masking feature

By default, the Data Masking feature is enabled after creating the extension. At any PiT, the Data Masking feature can be disabled and enabled by using the configuration setting **`pgx_datamasking.enable`**. This setting can be modified in the `postgresql.conf` file or by running an SQL command, as shown in Example 4-37. When this task is done through an SQL command, as in all configuration settings, it applies only to the current session.

This setting can be modified only by a superuser, and if it is disabled, Data Masking policies are stopped from being applied and the original data becomes visible.

Example 4-37 Commands to disable Data Masking

```
postgres=# show pgx_datamasking.enable;
 pgx_databaseking.enable
-------------------------
 on
(One row)

postgres=# set pgx_datamasking.enable=false;
```

```
SET
postgres=# show pgx_datamasking.enable;
------------------------
 off
(One row)
```

4.5.7 How to use Data Masking

Creating the Data Masking extension allows you to start creating masking policies for individual tables. To mask sensitive data, you must create a masking policy on the table and specify the column details that are to be masked and how you want them to be masked.

4.5.8 How to create Data Masking policies

Users can mask sensitive data by creating a masking policy on the table. A single policy can be created for any table and any number of columns can be added to the same policy. Example 4-38 demonstrates a command that creates a data mask on a table that is named `fulldemo`. From this example, we can see that a masking policy, `maskfulldemo`, was applied to a table named `fulldemo`.

Example 4-38 Data masking on a table

```
postgres=# select pgx_create_confidential_policy(table_name :=
'fulldemo',policy_name := 'maskfulldemo',enable := 't', expression := '1=1');
 pgx_create_confidential_policy
-------------------------------
 t
(One row)
```

A Data Masking policy is used to define:

- What to mask

  The column details of the data to be masked.

- When to mask

  The condition that is associated with the policy decides whether masking should be applied or not for the query results. Users can specify the masking condition through the '`expression`' attribute of policy during policy creation. This expression runs every query execution time, and masking is performed only when the expression results in a true condition. For example, a '`1=1`' expression always returns true and the masking condition is considered to be satisfied.

- How to mask

  Users can choose how to mask the sensitive data through the **`function_type`** attribute of the create or alter Data Masking policy commands. Acceptable values are `Full`, `Partial`, or `regex`. If this attribute is not specified, then by default the `Full` Data Masking type is applied.

  – Full

    All the data in a column is replaced by an alternative value. Replacement values come by default for each supported data type and can be viewed in the table `pgx_confidential_values`.

    Example 4-39 on page 145 shows the command to view the default replacement values.

Example 4-39 Viewing the default replacement values

```
postgres=# select * from pgx_confidential_values;
 number_value | char_value | varchar_value | date_value |      ts_value
--------------+------------+---------------+------------+-----------------
            0 |            |               | 1970-01-01 | 1970-01-01 00:00:00
(One row)
```

Values can also be modified according to your needs, as shown in Example 4-40.

Example 4-40 Modifying the default replacement values

```
postgres=# select pgx_confidential_values(number_value := 9,char_value
:='x',varchar_value := 'X',date_value := 'Jan-01-2000',ts_value := '1984-01-01
00:00:00');
-------------------------------
 t
(One row)
```

Example 4-41 provides a sample command demonstrating how to mask sensitive data by using the `Full` Data Masking type.

Example 4-41 Full Data Masking type

```
postgres=# select pgx_create_confidential _policy(table_name := 'demo',policy_name
:= 'policy_on_demo',enable := 't', expression := '1=1', column_name := 'annual
_salary');
pgx_create_confidential _policy
-------------------------------
t
(One row)



postgres=# select name,age,annual _salary from demo;
name  | age | annual_salary
------+-----+---------------
Bob   |  35 |             0
Nick  |  40 |             0
James |  45 |             0
(3 row)
```

– Partial

A portion of the column data is replaced by a replacement string. Start and end points of the data and the replacement character can be masked while creating the masking policy.

Example 4-42 shows the command to mask sensitive numeric data by using the `Partial` Data Masking type.

Example 4-42 Partial Data Masking type of numeric data

```
postgres=# select pgx_alter_confidential _policy(table_name := 'demo',policy_name
:= 'policy_on_demo',action := 'modify_column',column_name :=
'annual_salary',function_type := 'partial', function_parameters := '9,1,3');
pgx_alter_confidential _policy
------------------------------
 t
```

```
(One row)

postgres=# select name,age,annual_salary from demo;
name  | age | annual _salary
------+-----+----------------
Bob   |  35 |          99900
Nick  |  40 |          99900
James |  45 |         999900
(3 row)
```

Example 4-43 shows the command to mask sensitive character data by using the `Partial` Data Masking type.

Example 4-43 Partial Data Masking type of character data

```
postgres=# select pgx_al ter_confidential _policy(table_name := 'demo',policy_name
:= 'policy_on_demo',acti on := 'add_col umn',column_name :=
'credit_card',function_type := 'partial', function_parameters
.-'VVVVFVVVVFVVVVFVVVV , VVVV-VVVV-VVVV-VVVV, *, 5, 16');
pgx_alter_confidential_policy
-------------------------
 t
(One row)

postgres=# select * from demo;
name   | age |     credit_card     |annual _salary
-------+-----+---------------------+---------------
 Bob   |  35 | 1234-****-****-**** |         99900
 Nick  |  40 | 4567-****-****-**** |         99900
 James |  45 | 6767-****-****-**** |        999000
(3 row)
```

Example 4-44 shows the command to mask sensitive date/timestamp data by using the `Partial` Data Masking type.

Example 4-44 Partial Data Masking type of date/timestamp data

```
postgres=# select pgx_alter_confidential_policy(table_name :=
'fulldemo',policy_name := 'maskfulldemo',action := 'add_column',column_name :=
't',function_type := 'partial',function_parameters := 'mldly2000h0m0s0');
pgx_alter_confidential_policy
--------------------------------
t
(One row)

postgres=# select cv as company_name,t as annual_day from fulldemo;
company_name |     annual_ day
-------------+----------------------
 FUJITSU     |  2000-01-01 00 :00 :00
(One row)
```

– Regular expression

The column data is modified based on regular expression (`regex`) results. Users can specify the regex pattern, replacement character, and flags when creating a masking policy.

Example 4-45 shows the command to mask sensitive data by using the `regex` Data Masking type.

Example 4-45 Regex Data Masking type of sensitive data

```
postgres=# select pgx_alter_confidential_policy(schema_name := 'public',table_name
:= 'demo',policy_name := 'policy_on_demo',action := 'ADD_COLUMN',column_name :=
'project',function_type := 'REGEXP',regexp_pattern := 'Fuji ... ', regexp_
replacement := '***',regexp_ f ags := 'g' );
pgx_alter_confidential_policy
--------------------------------
t
(One row)

postgres=# select name,project from demo;
name   |           project
-------+------------------------------
Bob    |    *** Enterprise PostgreSQL
Nick   |    *** Enterprise PostgreSQL
James  |    *** Enterprise PostgreSQL
(3 row)
```

Data types that are supported by Data Masking types

Table 4-3 provides a list of data types that are supported by Data Masking types.

Table 4-3 Data types supported by Data Masking type

Category	Data type	Data Masking type		
		Full masking	Partial masking	Regular expression masking
Numeric type	smallint	Y	Y	N
	integer	Y	Y	N
	bigint	Y	Y	N
	decimal	Y	Y	N
	numeric	Y	Y	N
	float	Y	Y	N
	real	Y	Y	N
	double precision	Y	Y	N
Character type	character varying(*n*)	Y	Y	Y
	varchar(*n*)	Y	Y	Y
	character(*n*)	Y	Y	Y
	char(*n*)	Y	Y	Y
Date/timestamp type	date	Y	Y	N
	timestamp	Y	Y	N

Commands that are supported by the Data Masking feature

Here is the list of commands that can apply data masking and change the resultant data based on the masking policy:

- `SELECT`
- `COPY`
- `Pg_dump`
- `Pg_dumpall`

Note:

1. If a masking target is specified to `INSERT...SELECT` target columns, processing is performed by using data before the change.
2. If a masking target other than the `SELECT` target columns is specified, processing is performed by using data before the change.
3. If a masking target is specified in a function where the data type will be converted, an error occurs.

For more details about Data Masking formats, see, Chapter 6, "Data Masking", of the *FUJITSU Enterprise Postgres 12 on IBM LinuxONE Operation Guide*.

4.5.9 How to alter Data Masking policies

You can alter a Data Masking policy for the following tasks:

1. Addition of columns to the masking policy: You can include any number of table columns to mask in the same policy that is created for a table. At any point, one column can be added to the policy with masking type details.

   Example 4-46 demonstrates the command to add more columns to a Data Masking policy.

Example 4-46 Adding more columns to a Data Masking policy

```
postgres=# select pgx_alter_confidential_policy(table_name :=
'fulldemo',policy_name := 'maskfulldemo ',action := 'ADD_COLUMN',column_name :=
't',function_type := 'partial',function_parameters := 'm1d1y2000h0m0s0');
pgx_alter_confidential_policy
--------------------------------
t
(One row)
```

2. Removal of columns from masking policy: You can remove a column from the Data Masking policy. Column data previously returned as masked values is no longer masked, and clear data is returned.

   Example 4-47 demonstrates the command to remove a column from a Data Masking policy.

Example 4-47 Removing a column from the Data Masking policy

```
postgres=# select pgx_alter_confidential_policy(schema_name := 'public',table_name
:= 'fulldemo',policy_name := 'policy_on_demo',action := 'DROP_COLUMN',column_name
:= 'project');
pgx_alter_confidential_policy
--------------------------------
t
(One row)
```

3. Modify the masking condition: You can modify the **expression** attribute of the policy to set a new masking condition.

   Example 4-48 shows the command that is used to modify the Data Masking condition of a policy.

Example 4-48 Modifying the Data Masking condition of a policy

```
postgres=# select pgx_alter_confidential_policy(schema_name := 'public',table_name
:= 'demo',policy_name := ‘policy_on_demo',action := 'MODIFY_EXPRESSION',expression
:= $$current_user='postgres'$$);
pgx_alter_confidential_policy
--------------------------------
t
(One row)
```

4. Modify masking type of column: You can set different Data Masking types for a previously masked column. Then, data in the column can be masked based on the new settings.

   Example 4-49 shows the command to modify the Data Masking type of a column in policy.

Example 4-49 Modifying the Data Masking type of a column

```
postgres=# select pgx_alter_confidential_policy(table_name := 'demo',policy_name
:= ‘policy_on_demo',action := 'modify_column',column_name :=
'annual_salary',function_type := 'partial', function_parameters := '9,1,3');
pgx_alter_confidential_policy
--------------------------------
t
(One row)

postgres=# select name,age,annual_salary from demo;
name  | age | annual _salary
------+-----+---------------
Bob   |  35 |         99900
Nick  |  40 |         99900
James |  45 |        999000
(3 row)
```

5. Set policy or column description: You can include a short description about the policy or confidential columns in the policy.

Example 4-50 shows the command that is used to set a policy and column description in the policy.

Example 4-50 Setting a policy or column description

```
postgres=# select pgx_alter_confidential_policy(schema_name := 'public',table_name
:= 'demo',policy_name := ‘policy_on_demo',action :=
'SET_POLICY_DESCRIPTION',policy_description := 'Masking policy on demo table');
pgx_alter_confidential_policy
--------------------------------
t
(One row)

postgres=# select pgx_alter_confidential_policy(schema_name := 'public',table_name
:= 'demo',policy_name := ‘policy_on_demo',action :=
'SET_COLUMN_DESCRIPTION',column_name := ‘project’, column_description := 'To
demonstrate usage of regex’);
```

```
pgx_alter_confidential_policy
--------------------------------
t
(One row)
```

4.5.10 How to delete Data Masking policies

Data Masking policies can be deleted at any point by running the SQL command that is shown in Example 4-51. Only users with the superuser privilege can delete Data Masking policies.

Example 4-51 Command to delete a Data Masking policy

```
postgres=# select pgx_drop_confidential_policy(table_name := 'demo',policy_name := 
'policy_on_demo');
 pgx_drop_confidential_policy
--------------------------------
t
(One row)
```

You can also check the effect of a deleted Data Masking policy, as shown in Example 4-52.

Example 4-52 Checking the effect of a deleted Data Masking policy

```
postgres=# select pgx_drop_confidential_policy
 schema_name | table_name | policy_name       | expression |enable 
|policy_description
-------------+------------+-------------------+------------+-------+--------------
----
     public  |   fulldemo |      maskfulldemo | 1=1        | t     |
(One row)
```

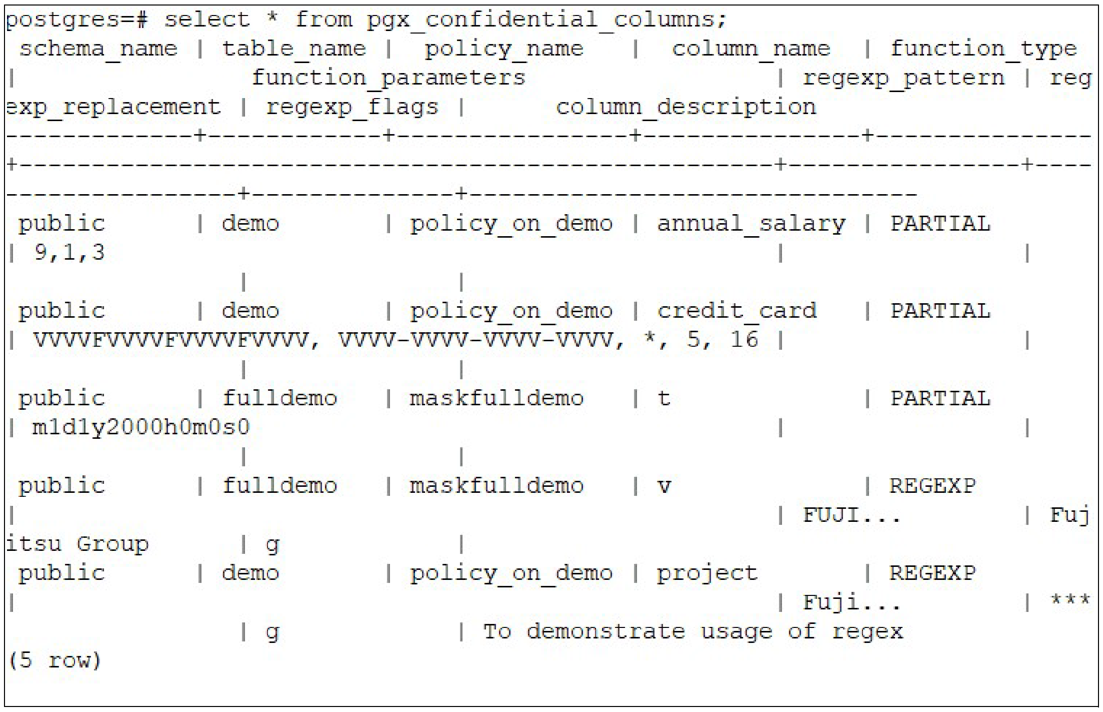
4.5.11 Viewing the list of Data Masking policies in a database

All the Data Masking policies that are created in a database can be viewed from the `pgx_confidential_policies` table by issuing the command that is shown in Example 4-53.

Example 4-53 Command to view all policies of the database

```
postgres=# select pgx_drop_confidential_policy
 schema_name | table_name | policy_name       | expression |enable 
|policy_description
-------------+------------+-------------------+------------+-------+--------------
----
     public  |   fulldemo |  maskfulldemo     | 1=1        | t     |
     public  |   demo     |  policy_on_demo   | 1=1        | t     |
(2 row)
```

Details about all the masked columns and how they are masked can be viewed in the `pgx_confidential_columns` table, as shown in Figure 4-8 on page 151.



```
postgres=# select * from pgx_confidential_columns;
 schema_name | table_name |  policy_name   |  column_name  | function_type
|                 function_parameters                  | regexp_pattern | reg
exp_replacement | regexp_flags |      column_description
-------------+------------+----------------+---------------+---------------
+----------------------------------------------------+----------------+----
----------------+--------------+------------------------------
 public      | demo       | policy_on_demo | annual_salary | PARTIAL
| 9,1,3                                              |                |
                |              |
 public      | demo       | policy_on_demo | credit_card   | PARTIAL
| VVVVFVVVVFVVVVFVVVV, VVVV-VVVV-VVVV-VVVV, *, 5, 16 |                |
                |              |
 public      | fulldemo   | maskfulldemo   | t             | PARTIAL
| m1d1y2000h0m0s0                                    |                |
                |              |
 public      | fulldemo   | maskfulldemo   | v             | REGEXP
|                                                    | FUJI...        | Fuj
itsu Group      | g            |
 public      | demo       | policy_on_demo | project       | REGEXP
|                                                    | Fuji...        | ***
                | g            | To demonstrate usage of regex
(5 row)
```

Figure 4-8   Command to view all masked columns of the database

4.5.12 Data Masking security notes

Logical replication allows publisher and subscriber databases to have their own or the same Data Masking policies.

In this scenario, the user must disable Data Masking on the publisher database whenever a subscription is created. This action ensures that subscribers can obtain the original data (initial copy) instead of the masked version. Then, it is the user's responsibility to set masking policies to each subscribed database.

Be careful when publishing Data Masking confidential tables (`pgx_confidential_policies` and `pgx_confidential_columns`) unless you are publishing all tables of the database and want to apply the same Data Masking policies on the subscribed database for all of them.

Otherwise, because these confidential tables contain the masking policies for all tables of the database, the confidential policies of unpublished tables might be unintentionally published. Additionally, it is not possible to apply different Data Masking policies on the subscriber database.

Note: A database dump taken by using the **`COPY`** command has obfuscated data for the masked columns. Doing a backup by using file copy strategy will have the original data.

4.6 Audit Logging

In this section, we describe the steps to configure and use the Audit Logging feature to generate audit logs that are required to comply with government, financial, or ISO certifications on a FUJITSU Enterprise Postgres 12 server with Red Hat Enterprise Linux Server 8 on IBM LinuxONE.

4.6.1 Overview

FUJITSU Enterprise Postgres Audit Logging enables organizations to trace and audit the usage of sensitive data and connection attempts of the database. Audit Logging provides a clear picture of data access by logging what data is accessed, when the data is accessed, how the data is accessed, and who accessed the data.

Administrators can configure the type of operations and objects to be logged based on organizations requirements. These logs can also be stored in dedicated audit log files that are separate from the server log for easy access.

The Audit Logging feature is available by default as an extension (`pgaudit`) with FUJITSU Enterprise PostgreSQL.

There are two types of Audit Logging to configure:

1. Session audit logging: Logs information about connection attempts (request, authentication, and disconnection); backup events; server starts and promotion of standby to primary events; maintenance events such as vacuum and checkpoint; and read/write events on database objects. An administrator can configure these events, the timeframe within which the events are logged, and the objects and database to be logged.
2. Object audit logging: Enables an administrator to log data manipulation language (DML) operations that are carried out in specific objects. Object logging allows a more granular level of logging, for example, an administrator can configure to log a selected operation that is handled in single column inside the table or index. Also, what you log can be configured inside the database session, so restarting the database server is not needed.

Administrators can also configure the destination location for the audit logs. There are two types of log destinations to configure:

1. Server log: All the audit log messages are directed to same log file that is used for server logging, that is, the standard FUJITSU Enterprise PostgreSQL log file that is inside the `pg_log` directory is used to store audit logs too. Log-file-related configurations that are made in the `postgresql.conf` file are applicable.
2. Dedicated audit log: A separate log file to store all audit logs. The Audit Log directory, log file name preference, permissions of the log file, log file size, log file rotation time, and log file truncation behavior can be configured through a `pgaudit`-specific configuration file.

For more information about FUJITSU Enterprise Postgres Audit Logging and all possible configurations, see Chapter 6, “Audit Log Feature”, of *FUJITSU Enterprise Postgres 12 on IBM LinuxONE Security Operation Guide*.

4.6.2 Benefits of using Audit Logging

The Audit Logging feature of FUJITSU Enterprise PostgreSQL is one of the security features that enables organizations to counter security threats like spoofing, unauthorized access, application manipulation, and misuse of privileges.

Audit Logging helps organizations to comply with all PCI DSS requirements that are related to audit logs.

Table 4-4 PCI DSS requirements

PCI DSS requirement	Database security measure
10	Track and monitor all access to network resources and cardholder data.
10.1	Implement audit trails to link all access to system components to each individual user.
10.2	Implement automated audit trails for all system components to reconstruct the following events: ► All individual user accesses to cardholder data (PCI DSS requirement 10.2.1) ► All actions that are taken by any individual with root or administrative privileges (PCI DSS requirement 10.2.2) ► Access to all audit trails (PCI DSS requirement 10.2.3) ► Invalid logical access attempts (PCI DSS requirement 10.2.4) ► Use of and changes to identification and authentication mechanisms, including but not limited to the creation of accounts and elevation of privileges, and all changes, additions, or deletions to accounts with root or administrative privileges (PCI DSS requirement 10.2.5) ► Initialization, stopping, or pausing of the audit logs (PCI DSS requirement 10.2.6) ► Creation and deletion of system-level objects (PCI DSS requirement 10.2.7)
10.3	Record at least the following audit trail entries for all system components for each event: ► User identification (PCI DSS requirement 10.3.1) ► Type of event (PCI DSS requirement 10.3.2) ► Date and time (PCI DSS requirement 10.3.3) ► Success or failure indication (PCI DSS requirement 10.3.4) ► Origination of event (PCI DSS requirement 10.3.5) ► Identity or name of affected data, system component, or resource (PCI DSS requirement 10.3.6)
10.4	Using time-synchronization technology, synchronize all critical system clocks and times and ensure that the following is implemented for acquiring, distributing, and storing time: Critical systems have the correct and consistent time
10.5	Secure audit trails so they cannot be altered.
10.5.1	Limit viewing of audit trails to those users with a job-related need.
10.5.2	Protect audit trail files from unauthorized modifications.
10.5.3	Promptly back up audit trail files to a centralized log server or media that is difficult to alter.
10.5.5	Use file-integrity monitoring or change-detection software on logs to ensure that existing log data cannot be changed without generating alerts.
10.6	Review logs and security events for all system components to identify anomalies or suspicious activity. Log harvesting, parsing, and alerting tools may be used to meet this requirement.
10.7	Retain an audit trail history for at least one year, with a minimum of three months immediately available for analysis (for example, online, archived, or restorable from backup).

4.6.3 How to use audit logging

In this section, we describe the prerequisites and configuration parameters for Audit Logging. We also provide to examples of using Audit Logging.

Prerequisites

The set of files that are related to Audit Logging are in the installation directory of FUJITSU Enterprise Postgres in the `OSS/pgaudit` subdirectory. Copy the `pgaudit` files to your installation directory by using the following command.

```
[root@rdbkpgr4 fsepuser]# cp -r /opt/fsepv12server64/OSS/pgaudit/*
/opt/fsepv12server64
```

Configuration parameters

To use Audit Logging, the FUJITSU Enterprise Postgres Database server should be configured with the following parameters through the configuration file `postgresql.conf`:

- **`shared_preload_libraries`**: Specify `pgaudit` in the list.
- **`pgaudit.config_file`**: The `pgaudit`-specific configuration file to use. Where to log, what to log, and when to log are all configured through **`pgaudit.config_file`**.
- **`log_replication_commands`**: Must be set to `on`.
- **`log_min_messages`**: `Error` or a higher value should be configured.

There are other optional parameters that are based on the log destination that is configured: **`serverlog`** and **`auditlog`**. For more information about these parameters, see 6.2, "Setup" in *FUJITSU Enterprise Postgres 12 on IBM LinuxONE Security Operation Guide*.

The `pgaudit` configuration file that is set by **`pgaudit.config_file`** should contain `logger`, where you specify `auditlog` to use a dedicated log file for the audit log or specify `serverlog` to use an existing database server's log file.

There are other configurations that are specific to the `auditlog` choice that can be configured in the `pgaudit` configuration file. For more information about such parameters, see 6.3, "pgaudit Configuration File", in *FUJITSU Enterprise Postgres 12 on IBM LinuxONE Security Operation Guide*.

Example use of object audit logging

Object logging can be made simple by assigning a dedicated role for auditing and all the permissions that are assigned to the specific role are logged automatically.

Example 4-54 provides an example of the `postgresql.conf` file settings.

Example 4-54 The postgresq.conf file settings

```
shared_preload_libraries = 'pgaudit' (change requires restart)
pgaudit.config_file = '/home/fsepuser/conf/audit.conf'
log_min_messages = warning
log_replication_commands = on
```

Example 4-55 provides the command and expected output to create the `pgaudit` module.

Example 4-55 Command to create the pgaudit module

```
postgres=# create extension pgaudit;
CREATE EXTENSION
```

Example 4-56 provides an example of the `pgaudit` configuration file that is named `audit.conf`.

Example 4-56 Command to create the pgaudit module

```
[fsepuser@rdbkpgr4 ~]$ cat conf/audit.conf
[output]
logger = 'auditlog'
[option]
role = 'auditor'
```

Example 4-57 provides the commands to generate audit logs.

Example 4-57 Commands to generate audit logs

```
[fsepuser@rdbkpgr4 ~]$ pg_ctl -D ./testdb -l logfile restart
waiting for server to shut down.... done
server stopped
waiting for server to start.... done
server started
[fsepuser@rdbkpgr4 ~]$ psql postgres
psql (12.1)
Type "help" for help.
postgres=# CREATE USER auditor NOSUPERUSER LOGIN;
CREATE ROLE
postgres=# create table account(id integer, name text, password text, description
text);
CREATE TABLE
postgres=# grant select(password), update(name,password) on table account to
auditor;
GRANT
postgres=# grant insert on table account to auditor;
GRANT
postgres=# insert into account values(101,'user1','pwd1','test user');
INSERT 0 1
postgres=# update account set password='pwd2' where password='pwd1';
UPDATE 1
postgres=# update account set description='test user for object log';
UPDATE 1
postgres=# select name,description from account;
 name  |       description
------+--------------------------
 user1 | test user for object log
(One row)
postgres=# select * from account;
 id  | name  | password |       description
----+-------+----------+--------------------------
 101 | user1 | pwd2     | test user for object log
(One row)
postgres=# create table account_noaudit(id integer, name text, password text,
description text);
CREATE TABLE
postgres=# insert into account_noaudit values(101,'user1','pwd1','test user');
INSERT 0 1
postgres=#  select * from account_noaudit;
 id  | name  | password | description
```

```
----+-------+----------+-------------
 101 | user1 | pwd1     | test user
(One row)
```

Example 4-58 shows a sample log output.

Example 4-58 Sample log output

```
[fsepuser@rdbkpgr4 pgaudit_log]$ pwd
/home/fsepuser/testdb/pgaudit_log
[fsepuser@rdbkpgr4 pgaudit_log]$ cat pgaudit-2020-11-08_212019.log
AUDIT: OBJECT,1,1,WRITE,INSERT,TABLE,public.account,"insert into account
values(101,'user1','pwd1','test user');",<not logged>
AUDIT: OBJECT,2,1,WRITE,UPDATE,TABLE,public.account,update account set
password='pwd2' where password='pwd1';,<not logged>
AUDIT: OBJECT,3,1,READ,SELECT,TABLE,public.account,select * from account;,<not
logged>
```

Example use of session audit logging

A session audit can have the same `postgresql.conf` file settings and retain session audit-related settings in the audit log configuration file.

Example 4-59 provides a sample of a `pgaudit` configuration file that is named `audit.conf`. The `rule` section defines session logging.

Example 4-59 The pgaudit configuration file

```
[fsepuser@rdbkpgr4 ~]$ cat conf/audit.conf
[output]
logger = 'auditlog'
[option]
role = 'auditor'
[rule]
class = 'READ,WRITE'
object_name = 'public.account'
```

Example 4-60 provides the SQL command to generate audit logs.

Example 4-60 SQL command to generate audit logs

```
[fsepuser@rdbkpgr4 ~]$ pg_ctl -D ./testdb -l logfile restart
waiting for server to shut down.... done
server stopped
waiting for server to start.... done
server started
[fsepuser@rdbkpgr4 ~]$ psql postgres
psql (12.1)
Type "help" for help.
postgres=# insert into account values(102,'user2','pwd102','test user2');
INSERT 0 1
postgres=# select * from account;
 id  | name  | password |       description
----+-------+----------+-------------------------
 101 | user1 | pwd2     | test user for object log
 102 | user2 | pwd102   | test user2
(2 row)
```

Example 4-61 provides a sample log output.

Example 4-61 Audit Log output

```
[fsepuser@rdbkpgr4 pgaudit_log]$ pwd
/home/fsepuser/testdb/pgaudit_log
[fsepuser@rdbkpgr4 pgaudit_log]$ cat pgaudit-2020-11-08_220044.log
AUDIT: OBJECT,1,1,WRITE,INSERT,TABLE,public.account,"insert into account
values(102,'user2','pwd102','test user2');",<not logged>
AUDIT: SESSION,WRITE,2020-11-08 22:01:19
EST,[local],13456,psql,fsepuser,postgres,3/2,1,1,INSERT,,TABLE,public.account,,"in
sert into account values(102,'user2','pwd102','test user2');",<not logged>
AUDIT: OBJECT,2,1,READ,SELECT,TABLE,public.account,select * from account;,<not
logged>
AUDIT: SESSION,READ,2020-11-08 22:01:23
EST,[local],13456,psql,fsepuser,postgres,3/3,2,1,SELECT,,TABLE,public.account,,sel
ect * from account;,<not logged>
```

4.6.4 Transforming the output to SQL tables

Audit Logging output, when set up with dedicated log files, follows the standard CSV format so that log file contents can be converted into SQL tables. This transformation requires an extra module, `file_fdw`, which is available by default with the FUJITSU Enterprise Postgres server. This module can be installed when needed without disturbing a running database server.

By transforming log output to SQL, you can issue queries and filter required details quickly.

Separating the session and object audit logs

When both object and session logs are enabled, then the audit log file contains a mix of logs in all log files. A best practice is to separate the session and object logs before loading them into SQL tables.

The command that is shown in Example 4-62 parses all the files in the working directory and gathers session and object audit logs in to separate log files.

Example 4-62 Example commands to separate audit and session logs

```
[fsepuser@rdbkpgr4 pgaudit_log]$ grep '^AUDIT: OBJECT' * | sed -r
's/.*?\:AUDIT/AUDIT/g' > object_audit_log.log
[fsepuser@rdbkpgr4 pgaudit_log]$ grep '^AUDIT: SESSION' * | sed -r
's/.*?\:AUDIT/AUDIT/g' > session_audit_log.log
```

Transforming session audit logs

To load session logs in to a table, use the command that is shown in Example 4-63.

Example 4-63 Commands to load session logs in to a table

```
[fsepuser@rdbkpgr4 ~]$ psql postgres
psql (12.1)
Type "help" for help.
postgres=# create extension file_fdw;
CREATE EXTENSION
postgres=# create server auditlog foreign data wrapper file_fdw;
CREATE SERVER
```

```
postgres=# create foreign table auditlog1(header text, class text, ql_start_time
timestamp with time zone,remote_host_name text, backend_process_id text,
application_name text, session_user_name text, database_name text,
virtual_transaction_id text, statement_id text, substatement_id text, command_tag
text, sqlstate text, object_type text, object_name text, error_message text, sql
text, parameter text) server auditlog options(filename
'/home/fsepuser/testdb/pgaudit_log/session_audit_log.log', format 'csv');
CREATE FOREIGN TABLE
postgres=# select * from auditlog1;
     header     | class |     ql_start_time      | remote_host_name |
backend_process_id | application_name | session_user_name | database_name |
virtual_transaction_id | statement_id | sub
statement_id | command_tag | sqlstate | object_type |  object_name   |
error_message |                              sql                              |
parameter
---------------+-------+------------------------+------------------+--------------
------+------------------+-------------------+---------------+--------------------
----+--------------+----
------------+-------------+----------+-------------+----------------+-------------
--+----------------------------------------------------------------+--------------
 AUDIT: SESSION | WRITE | 2020-11-08 22:01:19-05 | [local]        | 13456
| psql             | fsepuser          | postgres      | 3/2                    |
1            | 1
            | INSERT      |          | TABLE       | public.account |
| insert into account values(102,'user2','pwd102','test user2'); | <not logged>
 AUDIT: SESSION | READ  | 2020-11-08 22:01:23-05 | [local]        | 13456
| psql             | fsepuser          | postgres      | 3/3                    |
2            | 1
            | SELECT      |          | TABLE       | public.account |
| select * from account;                                         | <not logged>
(2 row)
```

Transforming object audit logs

Use the command that is shown in Example 4-64 to load object audit logs in to a table.

Example 4-64 Commands to load object logs in to a table

```
postgres=#  create foreign table auditlog2(header text, statement_id text,
substatement_id text, class text,  command_tag text, object_type text, object_name
text, sql text, parameter text) server auditlog options(filename
'/home/fsepuser/testdb/pgaudit_log/object_audit_log.log', format 'csv');
CREATE FOREIGN TABLE
postgres=# select * from auditlog2;
    header     | statement_id | substatement_id | class | command_tag |
object_type |  object_name   |                         sql
|  parameter
--------------+--------------+-----------------+-------+-------------+------------
-+----------------+---------------------------------------------------------------
-+--------------
 AUDIT: OBJECT | 1            | 1               | WRITE | INSERT      | TABLE
| public.account | insert into account values(101,'user1','pwd1','test user');
| <not logged>
 AUDIT: OBJECT | 2            | 1               | WRITE | UPDATE      | TABLE
| public.account | update account set password='pwd2' where password='pwd1';
| <not logged>
```

```
 AUDIT: OBJECT | 3           | 1             | READ  | SELECT      | TABLE
| public.account | select * from account;
| <not logged>
 AUDIT: OBJECT | 1           | 1             | WRITE | INSERT      | TABLE
| public.account | insert into account values(102,'user2','pwd102','test user2');
| <not logged>
 AUDIT: OBJECT | 2           | 1             | READ  | SELECT      | TABLE
| public.account | select * from account;
| <not logged>
(5 row)
```

Note: Details in the log files differ based on the configuration settings. Create the foreign table based on the log output.

4.7 Authentication

FUJITSU Enterprise Postgres provides a flexible approach to client authentication by allowing a configuration where you can use several authentication methods for different combinations of connection requests that are differentiated by connection type, connection source, and user.

This approach can be useful in allocating an authentication method that is appropriate to a specific area of the network or group of users. This section describes the recommended authentication methods that should be used by an enterprise requiring a secure approach.

4.7.1 Host-based authentication

FUJITSU Enterprise Postgres employs host-based authentication. A client connection request is examined and then matched to an entry in a configuration file that is called `pg_hba.conf` to determine the type of authentication that is used for that connection request.

Matching considers the type of connection, the database being connected to, the user, and the host or subnet that the request comes from.

The product documentation provides more information about how to configure host-based authentication. The purpose of this section is to provide you with information that enables you to select the most appropriate and secure authentication method for your situation.

FUJITSU Enterprise Postgres provides many different authentication methods, many of which are fine for development and test environments but are not suitable for production systems requiring a high level of security.

Here are the recommended authentication methods for production systems that are covered in this section:

- Password-based (Salted Challenge Response Authentication Mechanism (SCRAM))
- Kerberos (Generic Security Standard Application Programming Interface (GSSAPI))
- SSL certificates (CERT)
- Socket-based (PEER)

4.7.2 SCRAM

SCRAM is password-based. Although password-based authentication methods are generally considered to have many shortcomings, SCRAM uses AES-256 bit encryption and does not require the password to be passed to the server.

The challenge response part works like the MD5 authentication method that it replaces:

1. The client sends a random nonce (a random byte sequence of varying length that is used only once) to the FUJITSU Enterprise Server. The server sends the client a random nonce that is appended to the one that the client sent along with salt (for salting the client's password) and iteration count.
2. The client sends its proof that it knows the password, which is calculated in accordance with the guidelines that are provided in RFC 5802 and RFC 7677.
3. The server sends back proof that it also knows the password without either sending the password across the network.

Configuring FUJITSU Enterprise Postgres requires three actions:

1. The server must know which method to use to encrypt/hash passwords. To do so, set the **`password_encryption`** setting in the `postgresql.conf` file to `'scram-sha-256'`.
2. Creating a user with its password encrypted by the server. The **`createuser`** command can accomplish this task.
3. In the host-based authentication configuration file `pg_hba.conf`, you must specify the `'scram-sha-256'` authentication type for the combination of user, database, method, and host that should be using this method.

4.7.3 GSSAPI

The GSSAPI method of authentication provides a framework of generic security services that the client can use to use any of the several security technologies in it. Kerberos is one such technology that you can use for authentication with FUJITSU Enterprise Postgres.

Kerberos is an industry standard authentication protocol that is found in many products, such as OpenLDAP, Active Directory, Chrome, and FUJITSU Enterprise Postgres.

Kerberos implements a Key Distribution Center (KDC) containing a repository of all principles (shared secrets of users and services) and two servers that used during the authentication process:

- Authentication server
- Ticket granting server

This method of authentication works by the FUJITSU Enterprise Client using a symmetric encryption key to carry out client authentication with the server.

Using Kerberos means that the client and FUJITSU Enterprise Server can mutually authenticate each other and do not directly exchange encryption keys or passwords across the network.

Figure 4-9 gives an overview of the authentication process, which is detailed in the following list:

- A simplified authentication process begins with the client requesting a connection to the FUJITSU Enterprise Postgres server (1), which then looks up which authentication method to use from the `pg_hba.conf` configuration file. If the combination of connection method, user, host, and database matches an entry that specifies GSSAPI, the connection is routed to the KDC server.
- The client request to connect to FUJITSU Enterprise Postgres is received by the authentication service (2), which responds to the known user (a principle that is cached in the KDC) with a couple of messages. The first is a Ticket Granting Ticket (TGT), followed by a message containing a temporary session key that is encrypted by using the client's secret key (3), which the KDC already cached as the client principle.
- The client decrypts the temporary session key with its client secret. The client sends the TGT along with some more messages that are encrypted with the temporary session key to the Ticket Granting Service (4).
- The Ticket Granting Service decrypts the messages validating that the client can access the FUJITSU Enterprise Postgres service and sends a Service Ticket along with two messages containing a communications key: One encrypted with the temporary session key, and the other encrypted with the service secret key (5).
- The server secret encrypted communications key is forwarded by the client to the FUJITSU Enterprise Postgres server (it does not need to decrypt it, so it passes it on) (6). Both the client (using the temporary session key) and the server (using its server secret key) can exchange authentication information through symmetrically encrypted communication by using the communications key that is issued by the Ticket Granting Server (7).

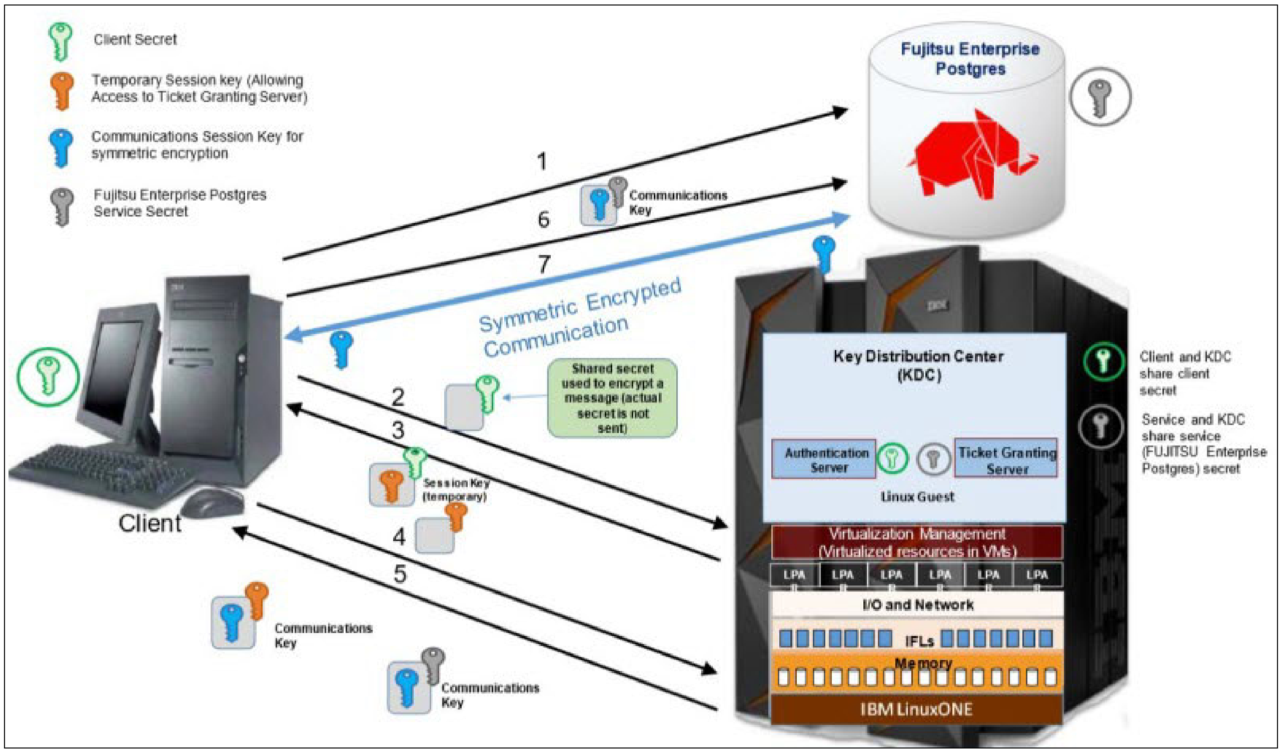

Client Secret
Temporary Session key (Allowing Access to Ticket Granting Server)
Communications Session Key for symmetric encryption
Fujitsu Enterprise Postgres Service Secret
Fujitsu Enterprise Postgres
1
6
7
Communications Key
Symmetric Encrypted Communication
Shared secret used to encrypt a message (actual secret is not sent)
2
3
4
5
Session Key (temporary)
Communications Key
Communications Key
Client
Key Distribution Center (KDC)
Authentication Server
Ticket Granting Server
Linux Guest
Virtualization Management (Virtualized resources in VMs)
LPA
I/O and Network
IFLs
Memory
IBM LinuxONE
Client and KDC share client secret
Service and KDC share service (FUJITSU Enterprise Postgres) secret

Figure 4-9 Authentication process

The following steps should be followed to configure FUJITSU Enterprise Postgres for GSSAPI authentication:

1. Install and configure Kerberos KDC. This task can be different depending on the OS. If you use Red Hat Enterprise Linux, as a best practice, use the Identity Management tool.
   a. Install the KDC (on Red Hat Enterprise Linux by installing `krb5-server`, `krb5-libs`, and `krb5-workstation`).
   b. Edit the `krb5.conf` and `kdc.conf` configuration files to contain the realm name and domain to realm mappings.
   c. Set up the principals to have access to the Kerberos database through `kadm5.acl` and then create the Kerberos database.
   d. Start the Kerberos database and add the principals.
   e. Use **`ktadd`** to add the host to the keytab files.
   f. Test whether KDC works by trying to get a ticket by using **`kinit`** and **`klist`**.
2. Install and configure the PostgreSQL Server:
   a. Install the Kerberos client libraries on the FUJITSU Postgres Server and configure the `krb5.conf` configuration files by using the same details that are entered for the Kerberos server.
   b. Add a `fsepuser` principal and a `fsepserver/`*`realm name`* service principal by using **`kadmin`** on the KDC.
   c. Extract the newly created service principal to a keytab file by using **`ktutil`** (this principal will be used in the configuration of the FUJITSU Enterprise Postgres Server).
   d. Add a client principal, for example, the user that is used by an application to access the database, like `appuser` on the KDC.
   e. Test that these new principals work with **`kinit`** and **`klist`** (**`kinit fsepserver/`*`realm name`***). The `appuser` principal should be checked on the machine that the application (database client) is running on.
   f. Copy the extracted keytab file and place it in a folder in the FUJITSU Enterprise Postgres data directory. Then, edit the `postgres.conf` file and set **`krb_server_keyfile`** to the full path name of the keytab file.
   g. Add entries to the `pg_hba.conf` configuration file with a connection type of `hostgssenc` to indicate a TCP connection request that is encrypted with GSSAPI. The `hostgssenc` connection type allows GSS to be specified as the authentication method on the same line. An **`include_realm`** setting can be set to the value of `0`, which automatically removes the realm name from the user.
      - Use the `pg_ident.conf` file to map the principal users to FUJITSU Enterprise Postgres users if required.
      - Use **`kinit`** to initialize the user and then try connecting to the FUJITSU Enterprise Postgres Database instance.

4.7.4 CERT

Certificate authentication (CERT) can be used for SSL connection types. The client SSL certificate is used to authenticate the user.

This type of authentication means that the user does not have to supply a password when logging in. The Common Name (CN) that is contained in the client certificate is examined by the FUJITSU Enterprise Server and if it matches a valid database username and the certificate is valid, then the connection is deemed authenticated successfully.

The following steps describe what is involved in setting up CERT:

- Server setup:
  a. Obtain or generate a server certificate. You can generate one by using the **openssl** utility. There are plenty of tutorials on the internet explaining how to generate a key, remove the password, and then use the key to create a certificate.
  b. Set the server.key file so that it can be read only by the user that is the owner of the data directory.
  c. Obtain a certificate authority (CA) certificate. If in a testing environment, the server certificate can be used.
  d. Place the server and CA certificate in either:
     i. The data directory so that FUJITSU Enterprise Postgres can pick them up automatically.
     ii. A directory that you choose, where you set the **ssl_cert_file** and **ssl_ca_file** settings in the `postgresql.conf` configuration file.
  e. Configure the FUJITSU Enterprise Postgres settings for SSL:
     i. `ssl=on`
     ii. `ssl_cert_file=<location of server certificate>`
     iii. `ssl_key_file=<location of server key>`
     iv. `ssl_ca_file=<location of the CA certificate>`
  f. Configure the host-based authentication configuration file (`pg_hba.conf`) for the required connection request combination (connection type, user, database, and host address) to use certificate authentication. The type should be `hostssl`, and the method should be set to **'**`cert clientcert=1`**'**.
- Client setup:
  a. Obtain or create a certificate for each role that the client connects as. Again, you can use **opennssl**.
  b. Sign the client certificates by using the key of the CA certificate and server key by setting the CN to the role name.
  c. The `key`, `csr`, and `crt` files must be placed into the `.postgresql` folder under the home directory with read only permissions for that role.

Once configured, try connecting to the FUJITSU Enterprise Postgres server from the client by using SSL and specifying `sslmode=require` in the connection string.

4.7.5 PEER

Peer authentication (PEER) relies on OS kernel authentication when the user logs in to the host. FUJITSU Enterprise Postgres requests the username from the kernel with the understanding that the kernel already authenticated that user.

Because we are asking the kernel who the user is, this method of authentication can be used only for local, socket-based connections.

OS users can be mapped to a FUJITSU Enterprise Postgres user by using the `pg_ident.conf` file.

To map an OS username to a database username, add an entry to `pg_ident` to provide a mapping name, database username, and OS username. Then, edit the `pg_hba.conf` file and set the authentication method to `peer` for the appropriate entry along with a mapping attribute called **`map`** with a value of the mapping name that is specified in the `pg_hba.conf` file. An example of this configuration is shown in Example 4-65.

Example 4-65 Mapping an OS username to a database username

```
pg_ident.conf
#MAPNAME                SYSTEM-USERNNAME       PG-USERNAME
deptx                   john546                 jsmith
deptx                   sarah327                sbrown

pg_hba.conf
#TYPE   DATABASE        USER        ADDRESS      METHOD
Localall all                      peer map=deptx
```

5

High availability and high reliability architectures

IBM LinuxONE is a perfect choice for installations where high availability (HA) is a key customer criterion in the hardware infrastructure selection for database deployments. In this chapter, we show how the highly regarded HA characteristics of IBM LinuxONE are an ideal complement for installations that are running the FUJITSU Enterprise Postgres Database, and we use the unique FUJITSU Enterprise Postgres Database HA feature options that are available as a standard feature.

We also provide examples of some of the deployment scenarios that showcase how the FUJITSU Enterprise Postgres Database running on IBM LinuxONE can provide an excellent HA enterprise-ready environment solution. For assistance with designing a more tailored HA deployment architecture to suit your specific business requirements, contact your Fujitsu and IBM client representatives.

This chapter includes the following topics:

- FUJITSU Enterprise Postgres Database HA options
- HA building blocks for FUJITSU Enterprise Postgres Database on Linux running on IBM LinuxONE
- IBM LinuxONE with FUJITSU Enterprise Postgres Database: Considerations for high availability
- IBM LinuxONE with FUJITSU Enterprise Postgres Database: Examples for HA
- IBM LinuxONE with IBM z/VM live guest relocation for FUJITSU Enterprise Postgres Database

5.1 FUJITSU Enterprise Postgres Database HA options

The topic of HA is important for installations that are running databases that underpin critical applications for the organizations that they serve. This subject affects almost every data center or cloud environment in the IT Industry.

FUJITSU Enterprise Postgres provides a comprehensive and integrated set of HA technologies that enable rapid recovery from failures and minimize the impact of unplanned downtimes. The focus of this section is on HA options that are related to the FUJITSU Enterprise Postgres Database.

The available FUJITSU Enterprise Postgres HA options are listed in Table 5-1.

Note: HA in PostgreSQL depends on the Streaming Replication operation.

Table 5-1 FUJITSU Enterprise Postgres Advanced Edition HA software components

HA/disaster recovery (DR) feature	**Benefit**	**Cost**	**Level of availability**
Mirroring Controller (MC)	► Constantly monitor operating system (OS), server, disk, network, and database, and send notifications when a resource is not available. ► When the primary server fails, performs an automatic switch to the standby server to ensure operational continuity, and the primary server is disconnected. ► Detects streaming replication issues (log transfer network and Write-Ahead-Log (WAL) send/receive processes) by periodically accessing the FUJITSU Enterprise Postgres. system views.	No additional cost over the annual subscription license.	HA with minimal disruption (in seconds).
Server Assistant	► Provides quorum between primary and standby servers. ► Objectively checks the status of database servers as a third party and isolates (fences) unstable servers in such cases through anomaly detection.	No additional cost over the annual subscription license.	HA with minimal disruption (in seconds).
Connection Manager	► Detects kernel panics between the server running the client and the server running the FUJITSU Enterprise Postgres instance, physical server failures, and inter-server network link downs, and notifies the client or instance. ► Provides transparent connection support between an application and the HA FUJITSU Enterprise Postgres.	No additional cost over the annual subscription license.	HA with transparency in connectivity between Primary/Standby.

HA/disaster recovery (DR) feature	Benefit	Cost	Level of availability
WebAdmin	► Used for FUJITSU Enterprise Postgres setup; creating and monitoring a streaming replication cluster; database backups; and for recovery. ► GUI-based tool for rapid HA solutions as needed. ► Provides simple one-click backup and recovery operations.	No additional cost over the annual subscription license.	Scalability, HA, and backup and recovery.

5.2 HA building blocks for FUJITSU Enterprise Postgres Database on Linux running on IBM LinuxONE

This section describes how FUJITSU Enterprise Postgres HA options interact with Linux running on IBM LinuxONE. The FUJITSU Enterprise Postgres HA options are an ideal complement for the superb HA features that are provided by IBM LinuxONE.

Figure 5-11 shows a complete HA scenario that consists of the following major components:

- ► IBM LinuxONE system hardware-provided HA
- ► OS-provided HA
- ► FUJITSU Enterprise Postgres Database provided HA

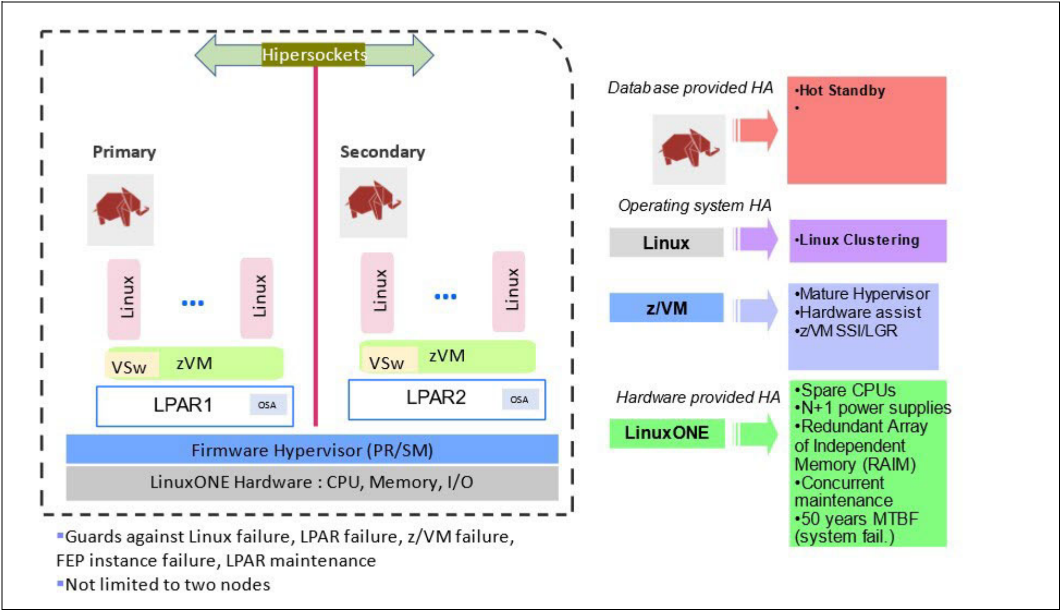


Figure 5-1 FUJITSU Enterprise Postgres on IBM LinuxONE: High availability

5.2.1 IBM LinuxONE Hardware provided HA

IBM LinuxONE has HA and reliability that is embedded through various redundant hardware options, providing a mean time between failures (MBTF) of around 50 years.

Examples of how hardware level HA is implemented on the IBM LinuxONE include:

- Transparent CPU sparing: If a CPU encounters a hardware failure, a spare CPU can be quickly brought online without affecting the applications or database.
- Redundant Array of Independent Memory (RAIM) memory modules: RAIM memory prevents a memory module from affecting the availability of a system or application.

In addition, IBM LinuxONE includes numerous other RAS features. The following hardware-provided HA features are standard offerings:

- Concurrent maintenance: At a minimum of two processor books, hardware parts of the processor can be replaced without a Power on Reset with the normal continuation of operations.
- Chip sparing in memory: An error detection and correction mechanism on IBM LinuxONE allows error detection during instruction execution and transparent error correction of spare processors that were configured to the system.
- N+1 Power supplies: The ability to have two sets of redundant power supplies. Each set of the power supplies has its individual power cords or pair of power cords, depending on the number of Bulk Power Regulator (BPR) pairs that are installed.

5.2.2 Operating system-provided high availability

Several Linux based OSs are supported on IBM LinuxONE. However, in this section, we focus on those OSs that are also supported by FUJITSU Enterprise Postgres Database whose OS level HA features can be leveraged to enhance the HA position.

Linux clustering

Red Hat Enterprise Linux Server and SUSE Enterprise Linux Server are the two Linux distributions that are certified for FUJITSU Enterprise Postgres on IBM LinuxONE at the time of writing. Several Linux OS-based solutions can also be used to provide HA for databases and applications.

The SUSE Advanced High Availability Extension is included with every subscription. It offers you a clustered file system for IBM LinuxONE and delivers all the essential monitoring, messaging, and cluster resource management features in an integrated suite of open source technologies.

The Red Hat Enterprise Linux High Availability Add-On is a collection of software that is deployed on top of the Red Hat Enterprise Linux OS. Policies, requirements, and limitations that are applicable to Red Hat Enterprise Linux are also applicable to Red Hat Enterprise Linux HA. Red Hat supports HA clusters containing 2 -4 running as IBM LinuxONE members.

These Linux based HA solutions work in a similar manner, where FUJITSU Enterprise Postgres Database is configured to run on a Linux guest. If planned maintenance or an unplanned system event occurs, the HA solution fails over to another idle Linux guest.

The Linux HA solution shuts down the source system database and unmounts the FUJITSU Enterprise Postgres file systems on the source system. Next, the Linux HA solution mounts the necessary FUJITSU Enterprise Postgres file systems, starts the application VIPs on the server, and then starts the FUJITSU Enterprise Postgres listener on this VIP IP address on the failover server. The last step is to start the database on the failover Linux guest such that the applications can now connect to the failover server.

Depending on activity load in the database and other factors, a failover can be as quick as a few seconds to the new server, or up to several minutes if many insert or update operations require rolling back.

5.2.3 Hypervisor (IBM z/VM)

z/VM offers a base to use IBM virtualization technology on IBM LinuxONE. Fully tested and deployed in FUJITSU Enterprise Postgres development environments, FUJITSU Enterprise Postgres products are certified with z/VM.

z/VM provides a highly secure and scalable enterprise cloud infrastructure. It also provides an environment for efficiently running multiple diverse critical applications on IBM LinuxONE with support for more virtual servers than any other platform in a single footprint.

z/VM virtualization technology is designed to allow for the capability to run hundreds to thousands of Linux servers on a single IBM LinuxONE footprint.

5.2.4 FUJITSU Enterprise Postgres provided high availability

There are different types of HA modes that can be achieved with FUJITSU Enterprise Postgres Advanced Edition. As described in Chapter 1, “Customer value” on page 1, in general a complete HA solution for databases should be highly resilient and have instant availability through automatic failover for business continuity. There are four major requirements for which a robust and fault-tolerant HA implementation is required:

- DR
- Load sharing and load balancing
- Instant failover
- Controlled switch-over

These HA requirements are provided by various data replication modes that are available with FUJITSU Enterprise Postgres.

Note: For more information about PostgresSQL replication, see Streaming Replication.

High availability terminology

Before we describe how to set up different HA modes, let us briefly understand the various HA modes terminologies and key differences between each other.

Let us now go through some of the nomenclature that is commonly and interchangeably used across different types of HA solutions:

- Primary (also know as master or active) server: The database servers or instances on which the data can be modified by the application through SQL data manipulation language (DML). They are also called *read/write servers* or *read/write databases instances* because the database instances accept read and write operations.
- Secondary (also know as standby, subordinate, or passive server): The database servers that receive changes from the primary database server through log shipping or streaming replication of PostgreSQL. They are also called *read only* or *read replica* servers or database instances because PostgreSQL inherently does not support multi-primary replication, so only `SELECT` SQL statements s are processed by the standby servers or database instances.

There are different types of standby operations that are characterized based on the data replication methods that are used between the primary and standby servers and the ability to be available for a connected application with either manual or automated switchover or failover processes:

- Cold standby
- Warm standby
- Hot standby

The main difference between these three methods is the replication technology that is used between the primary and secondary database servers and how the switchover or failover happens between the primary and secondary servers. Switchover and failover are processes in which a standby-only server is promoted to a primary server either through an automated process or through controlled semi-automated or manual process.

Table 5-2 describes these modes in more detail.

Table 5-2 PostgreSQL high availability modes

HA modes	Description
Cold standby	► The standby database server is in dormant state and brought into the active state when there is a major failure at the primary site due to hardware, software, power failure, or natural calamities. ► This mode of operation requires the standby database server to be initiated in recovery mode with all the transaction logs applied to the current point after the nightly backups of the primary database are applied on the standby database server. ► The oldest DR solution, which is outdated because it requires a long time for a database to be brought online.
Warm standby	► Warm standby supersedes cold standby because in the warm-standby mode of operation, the standby server is online and not in a dormant state. ► The transaction logs are continuously applied by using a replication method that is known as *log shipping* through which the transaction logs from the primary database server are copied to the standby server and applied to the standby server as a business as usual process. ► If there is a failure of the primary, the application must be redirected to the standby server, so the application is not connected to the standby server in normal mode of operation, and the updates on the standby database server are done only through log shipping replication. ► A DR setup with a better recovery time objective (RTO) than cold standby. The major drawback of warm standby is that the application is not connected to the standby database server, so even the read only SQL queries cannot be run on the standby database server in business as usual scenario. So, the high-compute capacity is not used efficiently. This limitation is overcome by using the hot-standby mode.

HA modes	Description
Hot standby	► In the hot-standby mode of operation, the transactional logs are continuously applied on the standby database server like warm standby. But unlike warm standby, which uses the log shipping replication method for copying transaction logs from primary database server to the standby database server, the hot-standby setup uses the streaming replication protocol. In addition, the standby database server is also available for the read-only SQL queries. ► This HA mode satisfies the following HA requirements: – DR – Load balancing – Replication to DR ► With FUJITSU Enterprise Postgres MC and Server Assistant, the hot-standby configuration can perform the following tasks: – Instant automatic failover from primary to standby – Controlled switchover from primary to standby

Figure 5-2 shows the differences between the cold-standby, warm-standby, and hot-standby HADR configurations that are possible with PostgreSQL replication.

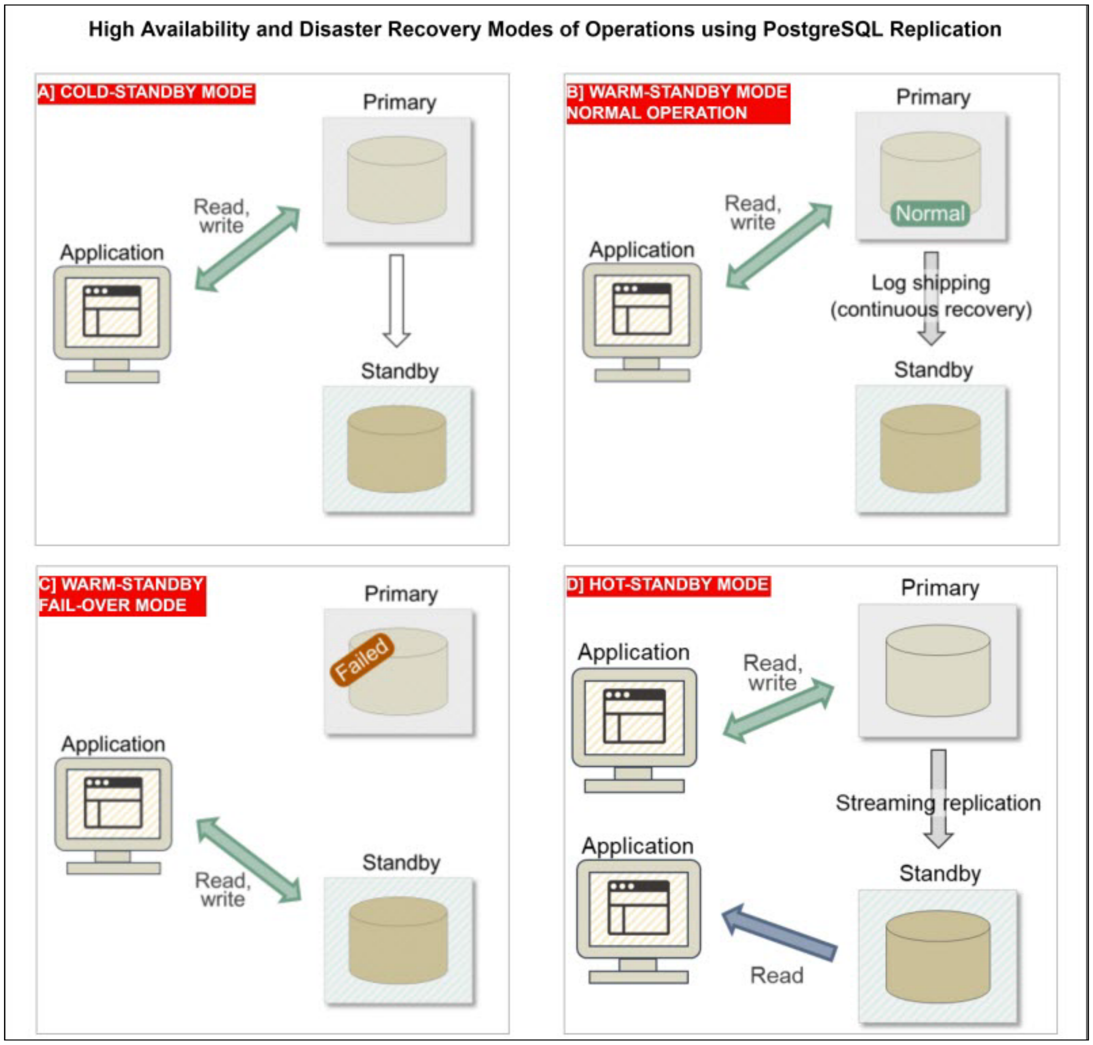


Figure 5-2 Primary/secondary HA modes of database operations that use log shipping and streaming replication

Note: In the context of HADR database setups, the Primary, Active, Master, and read/write terms are used interchangeably throughout the book. Similarly, the secondary, passive, subordinate, standby, and read only terms are used interchangeably throughout the book.

FUJITSU Enterprise Postgres provides enterprise-grade, hot-standby capability through MC, Server Assistant, and Connection Manager for uneventful automated failover from primary to secondary servers in the event of a disaster.

In hot-standby HA mode, the streaming replication can be either *asynchronous* or *synchronous*. The main difference between the two is in the way the handshake occurs between the primary and the secondary database instance. When primary and secondary databases are connected through asynchronous streaming replication, the data is streamed in an intermittent manner. In synchronous streaming replication from a primary database to a secondary database (standby) instance, the master database instance (also called the primary database instance) waits for an acknowledgment of each committed transaction from the subordinate database instance (also called secondary database (standby) instance) before the master database instance continues processing the next transactions.

Figure 5-3 shows the comparison of two modes of streaming replication.

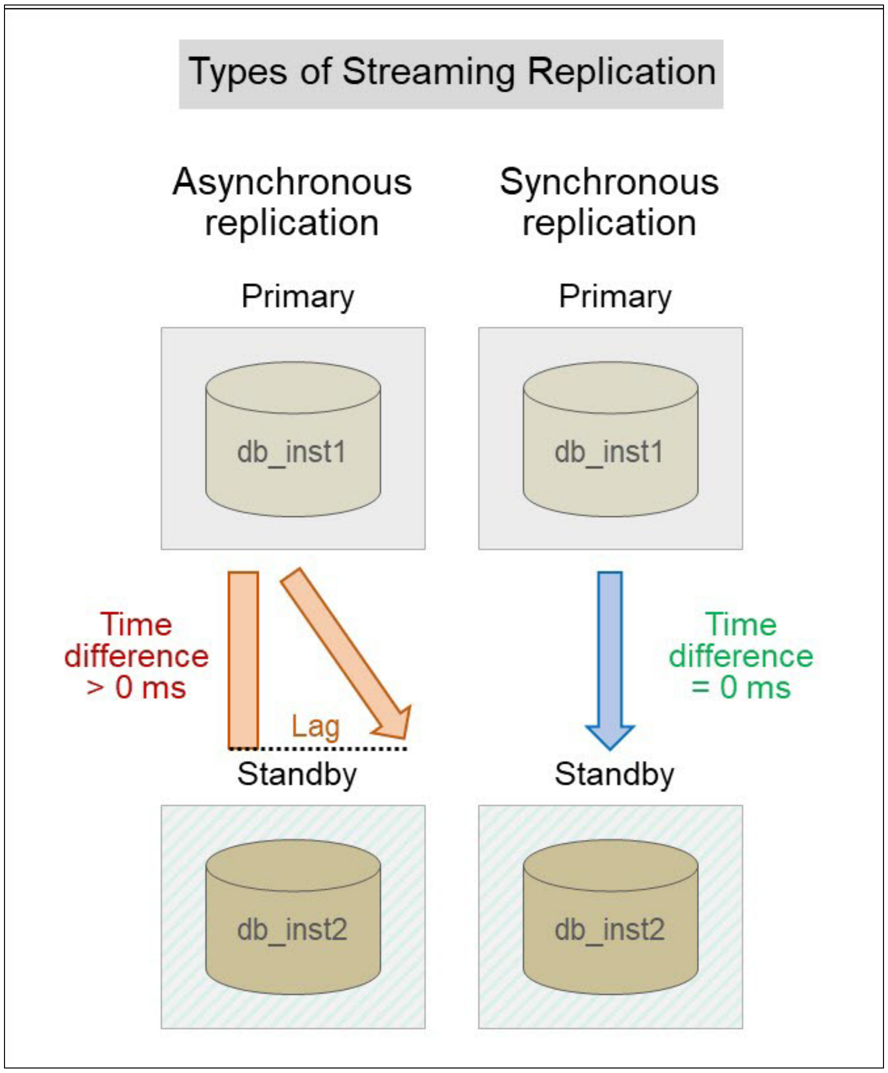


Figure 5-3 Asynchronous and synchronous streaming replication

Although streaming replication places a complete replica of a primary database instance into a secondary database instance, there might be reasons that you might not want to have a complete replica of the primary database instance. In this scenario, the PostgreSQL native logical replication replicates data from a selective list of databases or tables from the primary database instance to the secondary database instance. In native logical replication, the primary instance sends each transaction to the secondary instance through replication slots, which provides a publisher-subscriber mechanism between the master and the subordinate database instances.

Note: Unlike streaming replication, native logical replication supports implementation between different major versions of master and standby servers. For more information about native logical replication, see PostgresSQL.

High availability implementation types

Having described various concepts of HA modes, now we describe the HA implementations. There are two types of HA implementations for databases:

- Active-passive HA / Single-master HA:
  - In this implementation, there is only one master database instance with one or more subordinates that are either connected to the master or connected to the subordinate in a cascaded pattern.
  - Under this implementation, the application writes only on the master and reads from both master and subordinate database instances.
- Active-active HA / Multi-master HA
  - In this implementation, there is more than one master database instance, and the application can read and write data to more than one master.
  - In this setup, the data that is modified on the first master is sent to the second master before the modified data is committed on the first master, which requires a complex mechanism of locking the database objects through the life of the transaction between commits.

Note: PostgreSQL inherently does not support active-active HA architectures, although various solutions are available to implement active-active HA architectures in PostgreSQL.

HA implementation types that use FUJITSU Enterprise Postgres

The following active-passive HA implementations are possible with FUJITSU Enterprise Postgres:

- Single master-single subordinate
- Single master-multiple subordinates
- Single master-multiple subordinates that are connected through cascaded read replicas

Figure 5-4 shows the three active-passive HA architectures.

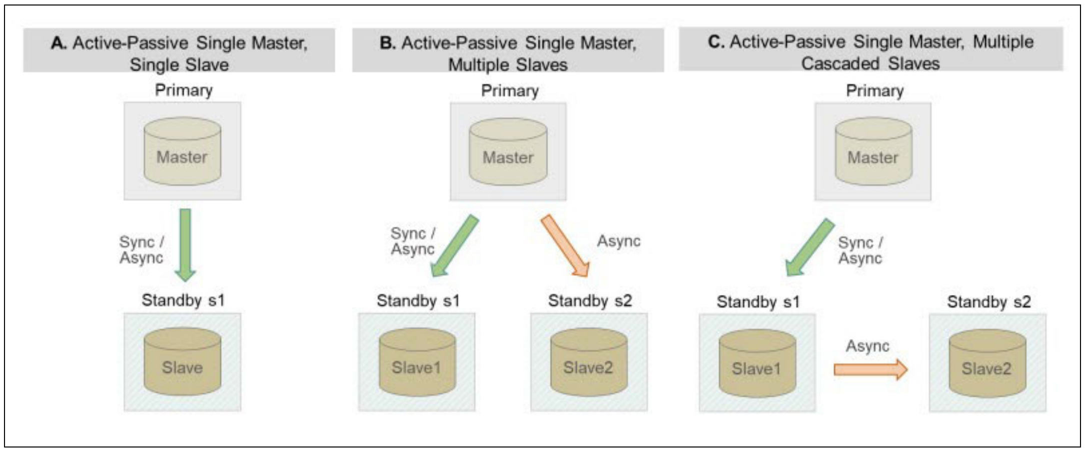


Figure 5-4 Active-passive high availability implementations

Depending upon your requirements, you can create a FUJITSU Enterprise Postgres cluster in various ways. Figure 5-5 shows a simple FUJITSU Enterprise Postgres HA architecture with four database instances for HA, disaster preparedness, and load-balancing. The quorum services are provided by the FUJITSU Server Assistant component of the FUJITSU Enterprise Postgres that is installed on the arbitration server.

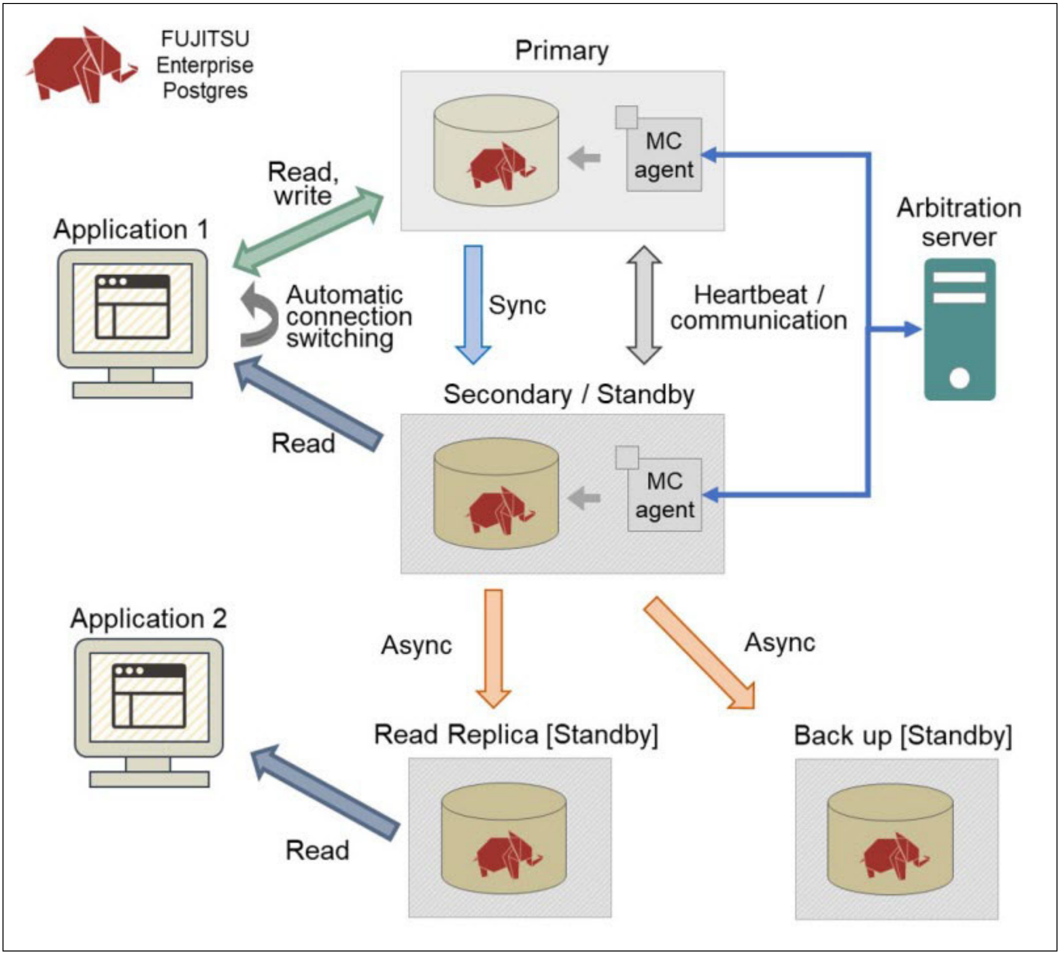


Figure 5-5 FUJITSU Enterprise Postgres HADR implementation example

Note: Several open source software (OSS) data management utilities come packaged with FUJITSU Enterprise Postgres. One of the important utilities is *Pgpool-II*. Pgpool-II is a connection pooling open source software that also can provide load-balancing and connection switchover. For simplicity, Pgpool-II is considered as part of the database installation, although Pgpool might require separate servers to provide a HA connection pooling layer between the application and the database layer. For more information about Pgpool-II, see Chapter 6, “Connection pooling and load balancing with Pgpool-II” on page 233.

Figure 5-5 on page 174 meets the following three architectural requirements:

- HA by using primary and secondary FUJITSU Enterprise Postgres instances with the capability of automated failover that is enabled by using MC and Server Assistant (Arbitration Server). In this scenario, the primary and secondary database (standby) instances are connected through synchronous streaming replication in hot-standby mode.
- A backup instance is connected to the secondary database (standby) instance through asynchronous streaming replication to provide a secure backup capability at an offsite location.
- Load balancing through the secondary database (standby) instance and the read replica database instance that is connected to the secondary database (standby) instance by using asynchronous streaming replication. Here an application (application 2) that needs only to query the data for reference or reporting purposes is connected to the read replica instance. In this way, the read replica can reduce the load on the primary database instance for any read only queries. At the same time, the secondary instance can process read only queries from application 1.

In this section, we briefly described a simple HA architecture that uses FUJITSU Enterprise Postgres and how it looks like at the database layer. When we consider implementing this configuration on IBM LinuxONE, we have various possibilities of creating HA, load-balancing, and DR architectures on a single IBM LinuxONE machine or on two IBM LinuxONE machines. In the next section, we explore implementing the above example on two IBM LinuxONE machines.

5.3 IBM LinuxONE with FUJITSU Enterprise Postgres Database: Considerations for high availability

In this section, we provide some considerations for implementing FUJITSU Enterprise Postgres in an IBM LinuxONE environment for HA. In Figure 5-5 on page 174, we considered a simple HA implementation that uses four FUJITSU Enterprise Postgres instances for HADR, load distribution, and backup. In this section, we describe the same simple FUJITSU Enterprise Postgres HADR implementation considerations on IBM LinuxONE.

5.3.1 Networking

IBM LinuxONE has several options for designing the network between logical partitions (LPARs) and other servers on a network.

An Open System Adapter (OSA) is a physical network card in IBM LinuxONE. An OSA can be dedicated to an FUJITSU Enterprise Postgres Linux guest, which is shared among multiple Linux guests, or configured in a virtual switch (VSWITCH) configuration. Virtual switches are beneficial for systems with many Linux guests sharing a network infrastructure.

IBM HiperSockets is a technology that provides high-speed TCP/IP connectivity within a central processor complex (CPC). It eliminates the need for any physical cabling or external networking connection between servers running in different LPARs.

The communication is through the system memory of the processor, so servers are connected to form an "internal LAN."

IBM HiperSockets are advantageous for workloads in which a large amount of data must be transferred quickly between LPARs if there is enough CPU capacity to support the network transfers.

Figure 5-6 shows some of these network concepts to consider in a network design for development, test, and production environments.

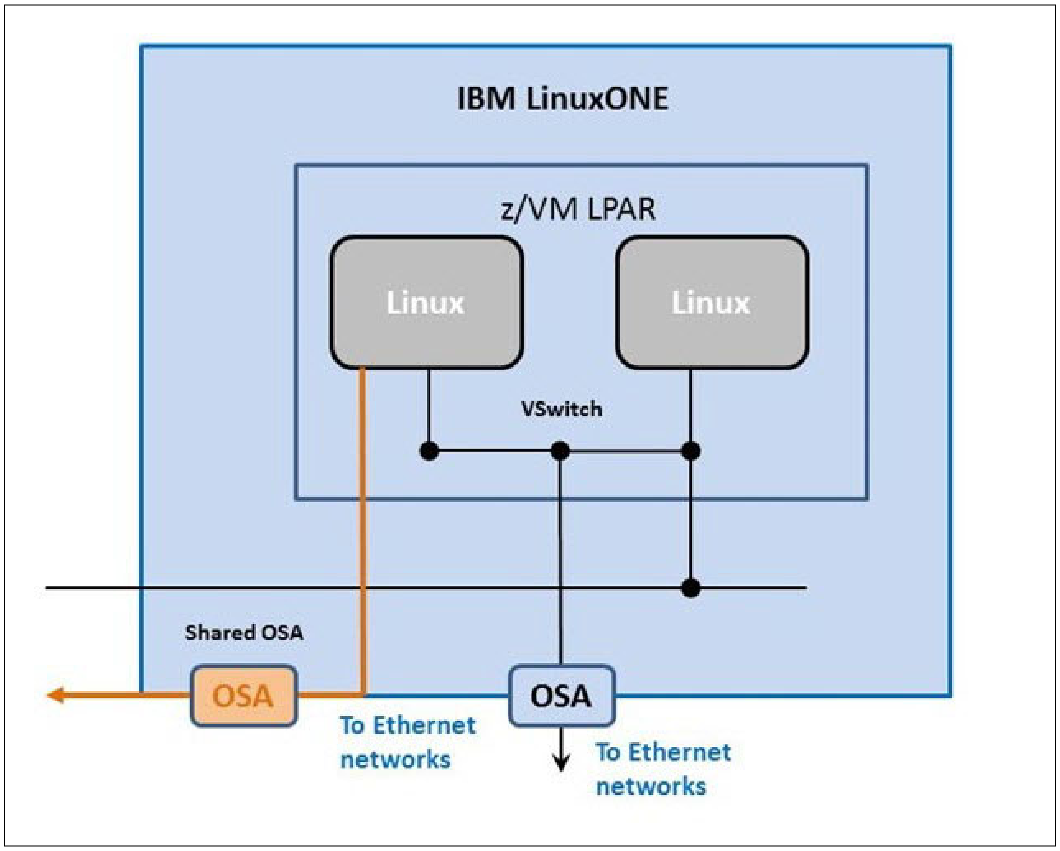


Figure 5-6 Networking concepts

5.3.2 FUJITSU Enterprise Postgres HA networking options

IBM LinuxONE includes the following options for supporting the FUJITSU Enterprise Postgres interconnection between FUJITSU Enterprise Postgres nodes in a FUJITSU Enterprise Postgres real application cluster or FUJITSU Enterprise Postgres RAC OneNode cluster configuration:

- Link Aggregation (active-active): Allows up to eight OSA-Express adapters to be aggregated when using a VSWITCH type of configuration.
- Linux bonding: Linux provides HA by using two or more Linux interfaces in an active-backup or active-active configuration. For example, Linux interfaces "eth1" and "eth2" are configured to create a HA bonded interface that is named "bond0".

5.3.3 I/O channel failover considerations

IBM LinuxONE supports multiple logical channel subsystem (LCSS) images that are mapped onto a physical channel subsystem (see Figure 5-7).

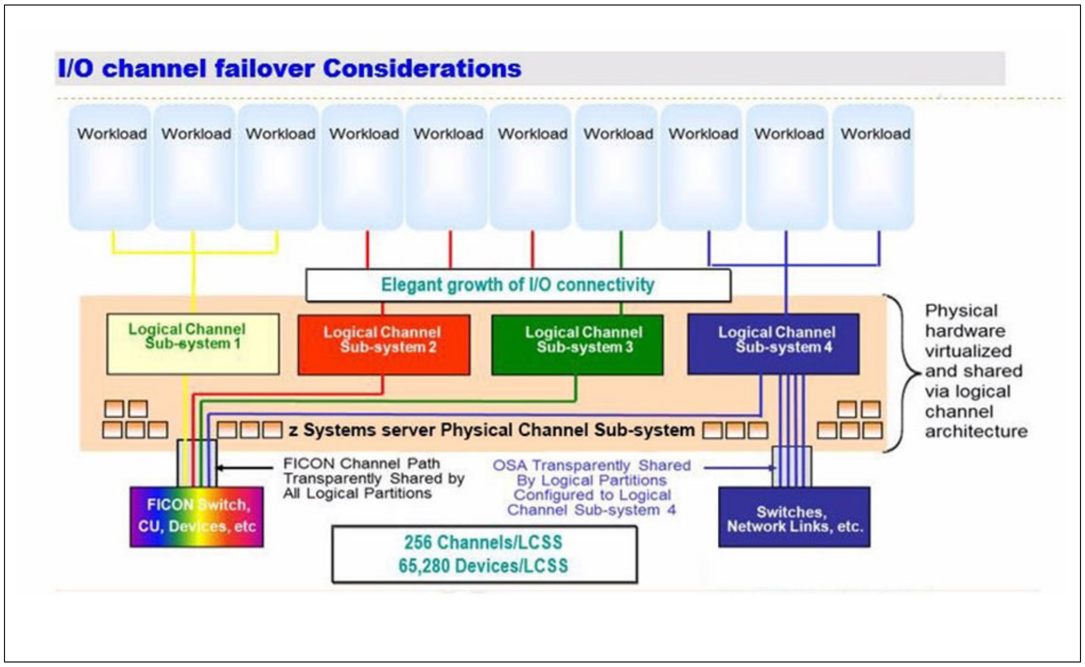

I/O channel failover Considerations
Workload
Workload
Workload
Workload
Workload
Workload
Workload
Workload
Workload
Workload
Elegant growth of I/O connectivity
Logical Channel Sub-system 1
Logical Channel Sub-system 2
Logical Channel Sub-system 3
Logical Channel Sub-system 4
Physical hardware virtualized and shared via logical channel architecture
z Systems server Physical Channel Sub-system
FICON Channel Path Transparently Shared by All Logical Partitions
OSA Transparently Shared By Logical Partitions Configured to Logical Channel Sub-system 4
FICON Switch, CU, Devices, etc
Switches, Network Links, etc.
256 Channels/LCSS
65,280 Devices/LCSS

Figure 5-7   Example of an I/O channel failover architecture

Consider the following points:

- The physical hardware, such as IBM FICON® channels and the OSA cards, are shared by using the logical channel architecture.
- The processors that run the channel subsystem are called the System Assist Processors. More than one System Assist Processor can be running the channel subsystem. The System Assist Processor drives the IBM LinuxONE server's I/O channel subsystem, which serves a collection of more than 1,000 high-speed buses.
- The System Assist Processor relieves the OS and the general-purpose CPs of much of the work that is required to run I/O operations.

  Specifically, the System Assist Processor schedules and starts each I/O request, that is, it finds an available channel path to the requested I/O device and starts the I/O operation. The System Assist Processor does not handle the actual data movement between central storage and the channel.
- Channels are the communication path from the CSS to the control units and I/O devices. They are represented by the black rectangles. In Figure 5-7, we show both FICON and OSA.

  A Channel Path Identifier (CHPID) is a value that is assigned to each channel path that uniquely identifies that path. You can have a maximum of 256 in an LCSS.

Within the physical I/O subsystem, channel paths and I/O adapters (for example, OSA Ethernet cards) can be shared, potentially by all logical subsystems. As shown in Figure 5-7, the FICON channel is shared by all four LCSSs (indicated by the yellow, red, green, and purple lines), and thus all partitions.

If necessary (for example, for performance reasons), a path or an adapter can be restricted to one LCSS or a subset of the LCSSs. Figure 5-7 on page 177 shows where the OSA on the right and in purple can be used only by LCSS 4 or the blue LPARs that are shown at the top.

5.3.4 More information about the LCSS

In this section, we provide more information about the LCSS; its maximums in terms of LPARs, channels, and physical devices it supports; and why we must create the LCSS concept to map to the physical channel subsystem.

IBM LinuxONE has a unique channel architecture that provides powerful and flexible support for the most demanding I/O performance and high-volume workloads. This channel architecture is managed by LCSS. Each IBM LinuxONE server can have up to four LCSSs, and each one can support 15 LPARs. Each LPAR can address 256 data channels and 64,000 I/O devices. Therefore, a single IBM LinuxONE system can handle over 1,000 data channels and a quarter of million I/O devices.

The LCSS architecture is further enhanced by the IBM LinuxONE Multiple Image Facility. This Multiple Image Facility allows all 15 LPARs that share a common LCSS to directly access each I/O device without having to forward the request through an intermediate partition, as is the case with UNIX architectures. The direct result of a shorter path length is improved I/O performance.

IBM LinuxONE Channel Spanning allows each device (disk or tape units that are attached by using Fibre Channel technology) to appear to be on any LCSS. The result is that the device may be accessed by any partition on IBM LinuxONE. Therefore, greater I/O flexibility and simplified operational management can be achieved.

IBM LinuxONE III supports the following components:

- Up to six LCSSs
- Up to 15 LPARs per LCSS
- Up to 256 channels per LCSS
- Up to 256 CHPIDs per LCSS
- Up to 64 K physical devices per LCSS
- Up to 1536 CHPIDs made available for the entire system
- Up to 1536 physical channels for the entire system

Usually in computing, virtualization allows more logical resources than physical. In this case, the physical resources exceed the logical channel-addressing ability of the LCSS.

The architecturally defined channel-path identification number that is called the CHPID must be maintained without change. The CHPID value is defined as an 8-bit binary number that results in a range of unique CHPID values 0 - 255.

Since the inception of the precursor S/370 XA channel-subsystem architecture in the late 1970s, this 8-bit CHPID was maintained without change because of its pervasive use in the IBM z/VM OS.

For example, the CHPID value, which is maintained in many internal programming control blocks, is displayed in various operator messages, and it is the object of various system commands, programming interfaces, and so on, all of which must be redesigned if the CHPID value was increased to more than an 8-bit number to accommodate more than 256 channel paths.

Therefore, another level of channel path addressing indirection was created because of the I/O subsystem that allows more than 256 physical channel paths to be installed and uniquely identified without changing the former 8-bit CHPID value and the corresponding programming dependencies on the CHPID.

The new CHPID value, called the physical channel identifier (PCHID), is a 16-bit binary number 0 - 65,279, which uniquely identifies each physically installed channel path.

With the current IBM LinuxONE servers, a maximum of 1024 external channel paths out of 65 K (for example, ESCON, FICON, and OSA) and 48 internal channel paths (for example, internal coupling and IQDIO hyperlink) are each assigned a unique PCHID value of 0 - 2,047.

5.4 IBM LinuxONE with FUJITSU Enterprise Postgres Database: Examples for HA

In this section, we show some of the possible HA configurations that can be adopted for FUJITSU Enterprise Postgres Database on IBM LinuxONE. For assistance with designing a more tailored HA deployment architecture that meets your specific business and operational requirements, contact your Fujitsu and IBM client representatives.

At the time of writing, FUJITSU Enterprise Postgres Database configurations are limited to active-passive, but when deploying on IBM LinuxONE, three potential deployment architectures may be adopted.

- An active-passive DB on a two-LPAR system within a single IBM LinuxONE server.
- An active-passive DB on two IBM LinuxONE servers within one data center.
- An active-passive DB on two IBM LinuxONE servers that are distributed between two data centers.

5.4.1 FUJITSU Enterprise Postgres active-passive on a two-LPAR system

This configuration relies on the granular partitioning capabilities of IBM LinuxONE, which can deliver Evaluation Assurance Level (EAL) 5+ isolation between the virtual servers, which is equivalent to two air-gapped physical servers that are deployed in independent racks.

This particular configuration delivers up to 99.99% availability because there is still the potential for network or application outages that might affect availability. In this configuration, FUJITSU Enterprise Postgres is configured in an active-passive mode where the passive database instance can function in read-only mode.

The arbitration server, which monitors the availability heartbeat between the active and passive database instances, is deployed on the secondary LPAR.

This configuration is shown in Figure 5-8.

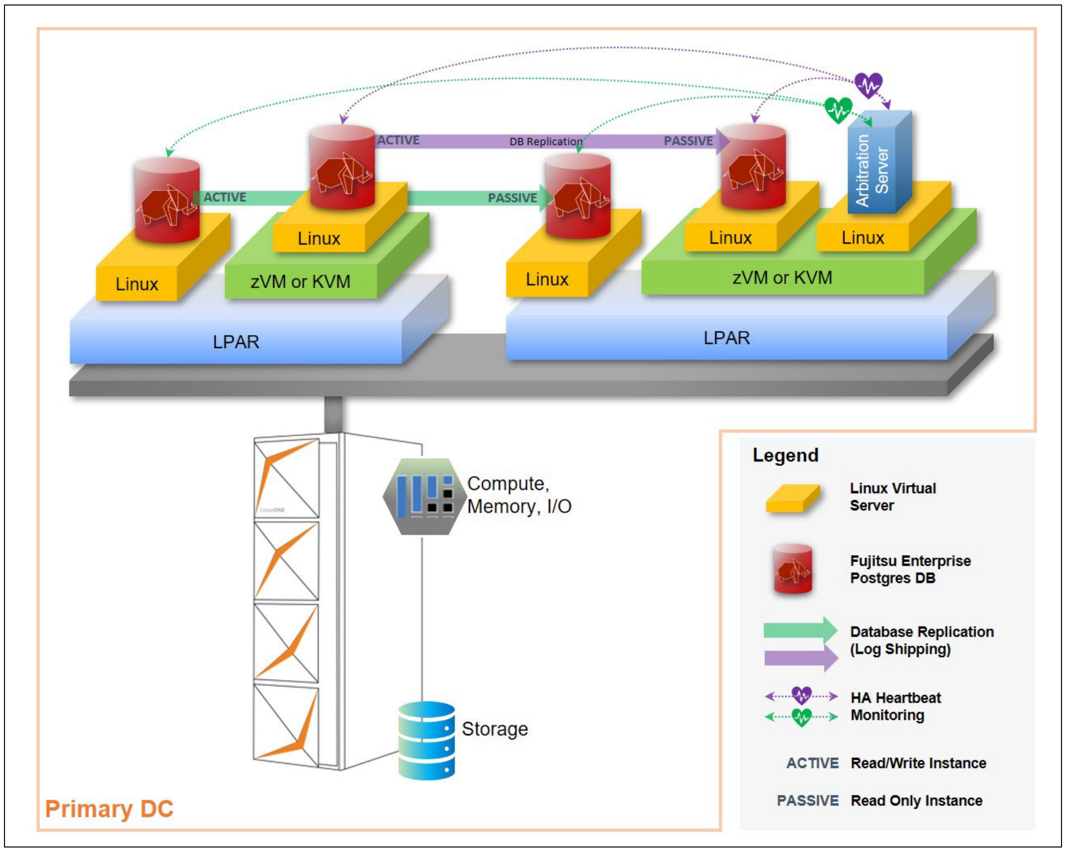


Figure 5-8 Active-passive mode on a two-LPAR system

5.4.2 FUJITSU Enterprise Postgres active-passive mode on two IBM LinuxONE systems within the same data center

This configuration leverages the granular EAL5+ partitioning and isolation capabilities of IBM LinuxONE and further improves on the 99.99% availability posture for mission-critical applications by deploying the FUJITSU Enterprise Postgres Database across independent physical servers within the same data center.

There is still the potential for a data center network that might affect both servers, but the impact of outages from server hardware failures is minimized. In this configuration, FUJITSU Enterprise Postgres is configured in an active-passive mode where the passive database instance can function in read-only mode. This configuration is shown in Figure 5-9 on page 181.

The arbitration server, which monitors the availability heartbeat between the active and passive database instances, is deployed on the secondary IBM LinuxONE server.

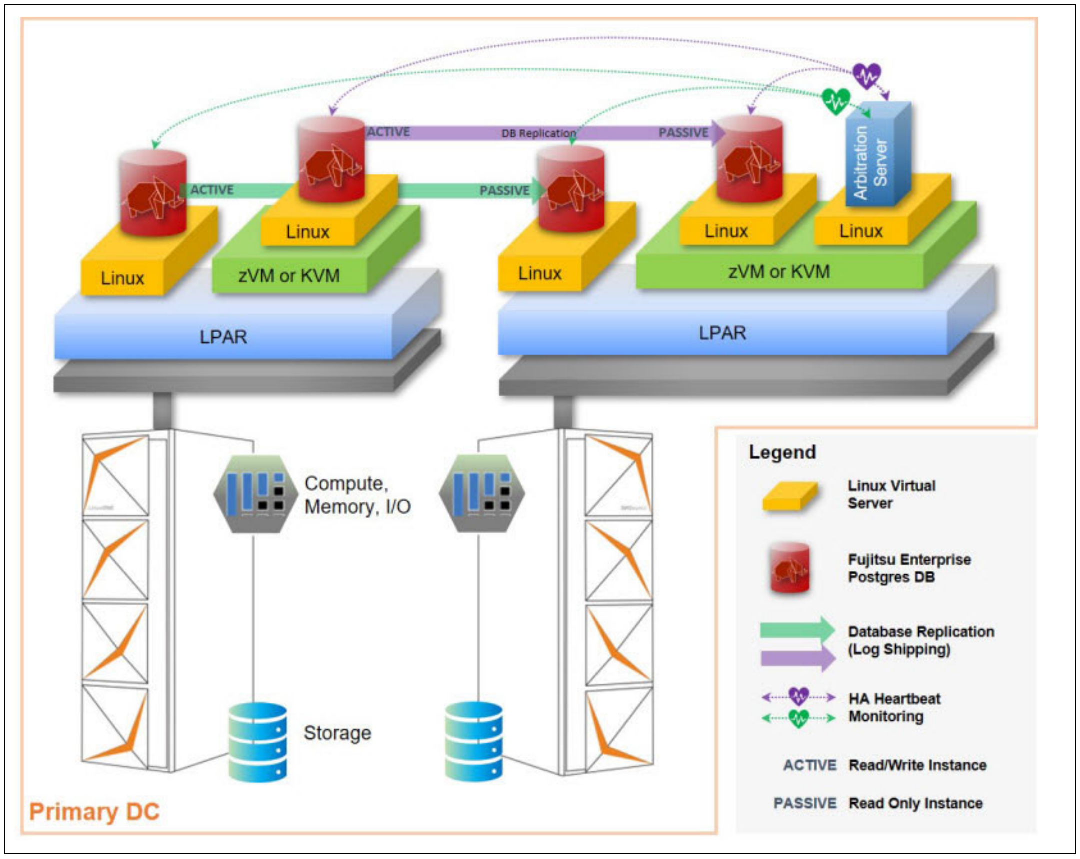


Figure 5-9 Active-passive mode on two IBM LinuxONE systems within the same data center

5.4.3 FUJITSU Enterprise Postgres active/passive mode on two IBM LinuxONE systems distributed across two data centers

The configuration that is shown in Figure 5-10 on page 182 leverages the granular EAL5+ partitioning and isolation capabilities of IBM LinuxONE and delivers the highest availability posture for mission-critical applications by deploying the FUJITSU Enterprise Postgres Database across independent physical servers across two data centers.

This configuration protects against primary data center failures and primary server hardware failures. In this configuration, FUJITSU Enterprise Postgres is configured in an active-passive mode where the passive database instance can function in read-only mode.

The arbitration server, which monitors the availability heartbeat between the active and passive database instances, is deployed on the secondary IBM LinuxONE server in the secondary data center.

In this configuration, the physical proximity and the available network bandwidth between the two data centers are key constraints to ensure that both data synchronization between with primary and stand-by databases and heartbeat monitoring by the arbitration server are at acceptable levels to ensure an effective HA configuration. This configuration also lends itself to achieving an effective DR posture where the standby database can be replicated to a tertiary database or backed up or within the secondary data center.

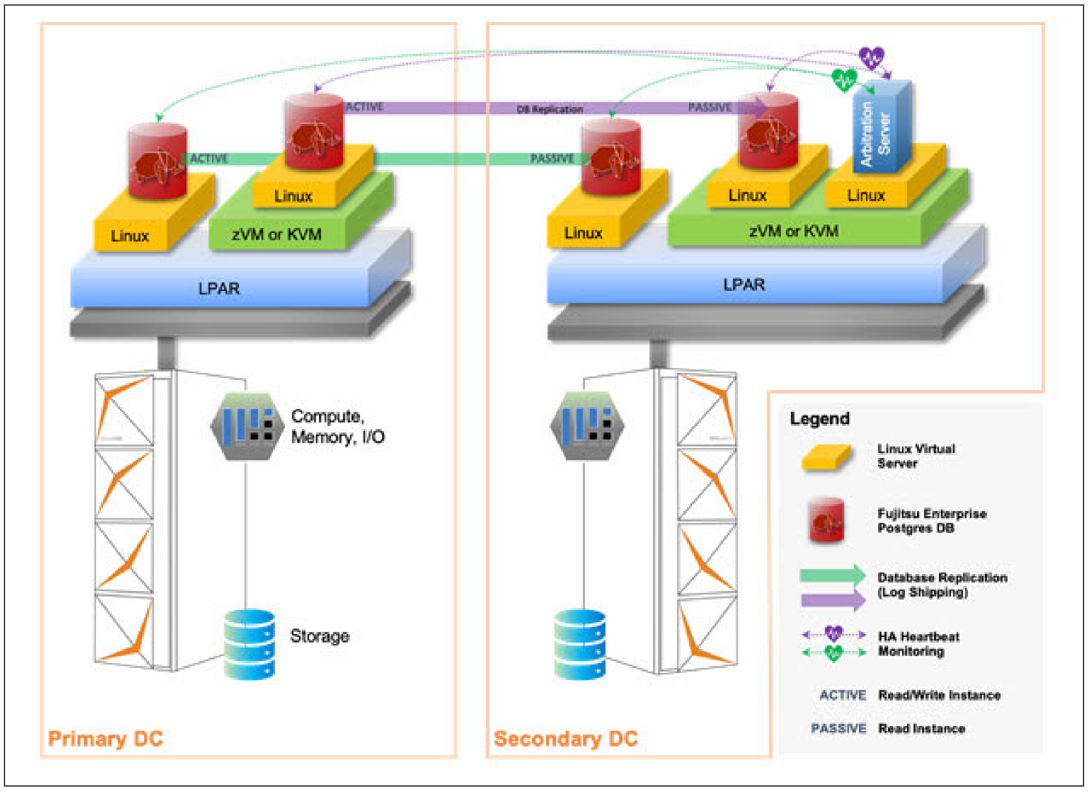


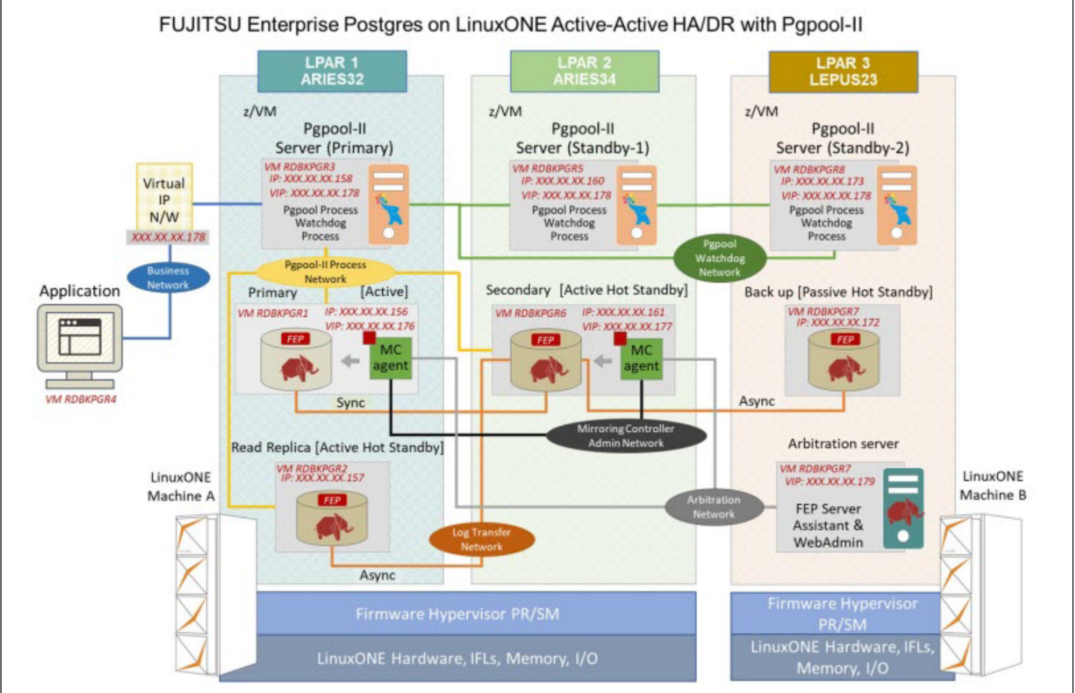
Figure 5-10 Active-passive mode on two IBM LinuxONE systems distributed across two data centers

5.4.4 Implementing an active/passive HA architecture when using a command-line interface

In this section, we show the steps to create the HA architecture on IBM LinuxONE that was briefly described in 5.2.3, “Hypervisor (IBM z/VM)” on page 169. In this example, we use two IBM LinuxONE machines with three LPARs for the installation of FUJITSU Enterprise Postgres Advanced Edition; Pgpool-II for connection pooling and load balancing; MC for heartbeat monitoring between the primary database instance and secondary database (standby) instance; and the Server Assistant on the arbitration server and the application server.

Figure 5-11 on page 183 provides the high-level architecture that is used in this section to show the implementation steps.

Note: To simplify the example implementation, Pgpool is installed on the same z/VM as the primary, secondary, and backup database instances. You might consider installing Pgpool-II on separate LPARs per your requirements.


FUJITSU Enterprise Postgres on LinuxONE Active-Active HA/DR with Pgpool-II
LPAR 1 ARIES32
LPAR 2 ARIES34
LPAR 3 LEPUS23
z/VM
Pgpool-II Server (Primary)
VM RDBKPGR3 IP: XXX.XX.XX.158 VIP: XXX.XX.XX.178
Pgpool Process Watchdog Process
Pgpool-II Server (Standby-1)
VM RDBKPGR5 IP: XXX.XX.XX.160 VIP: XXX.XX.XX.178
Pgpool-II Server (Standby-2)
VM RDBKPGR8 IP: XXX.XX.XX.173 VIP: XXX.XX.XX.178
Virtual IP N/W
XXX.XX.XX.178
Business Network
Application
VM RDBKPGR4
Pgpool-II Process Network
Pgpool Watchdog Network
Primary [Active]
VM RDBKPGR1 IP: XXX.XX.XX.156 VIP: XXX.XX.XX.176
FEP
MC agent
Sync
Secondary [Active Hot Standby]
VM RDBKPGR6 IP: XXX.XX.XX.161 VIP: XXX.XX.XX.177
Back up [Passive Hot Standby]
VM RDBKPGR7 IP: XXX.XX.XX.172
Async
Mirroring Controller Admin Network
Read Replica [Active Hot Standby]
VM RDBKPGR2 IP: XXX.XX.XX.157
Arbitration server
VM RDBKPGR7 VIP: XXX.XX.XX.179
FEP Server Assistant & WebAdmin
Arbitration Network
Log Transfer Network
LinuxONE Machine A
LinuxONE Machine B
Firmware Hypervisor PR/SM
LinuxONE Hardware, IFLs, Memory, I/O

Business N/W	🟦	Uses Virtual IP: XXX.XX.XX.178	Application connects to Pgpool-II Virtual IP
Pgpool-II Watchdog N/W	🟩	Uses IP: XXX.XX.XX.158/160/173	High Availability of Pgpool-II using Watchdog
Pgpool-II Process N/W	🟨	Use IP: XXX.XX.XX.156/161/157	Pgpool-II Process for Load Balancing, Load Distribution and Connection Pooling
Arbitration N/W	⬜	Uses VIP: XXX.XX.XX.176/177/179	Quorum Service by Server Assistant
Log Transfer using Replication N/W	🟧	Uses IP: XXX.XX.XX.156/161/157/172	Streaming Replication b/w database instances
Mirroring Controller Admin N/W	⬛	Uses IP: XXX.XX.XX.156/161	High Availability of FUJITSU Enterprise Postgres

Figure 5-11 HADR example of FUJITSU Enterprise Postgres with Pgpool-II on IBM LinuxONE

Note: Figure 5-11 shows a sample architecture that depicts all the components that enable HA, disaster preparedness, and load balancing by using FUJITSU Enterprise Postgres on IBM LinuxONE. Use discretion for designing appropriate HADR architecture per your business requirements. All the servers that are shown in Figure 5-11 are running as virtual machines (VMs).

To start implementing this architecture, ensure that you have met the prerequisites for storage, networking, Linux guests, and z/VM. For more information about the hardware and software prerequisites on IBM LinuxONE, see *the FUJITSU Enterprise Postgres Installation and Setup Guide for Server*, *FUJITSU Enterprise Postgres Installation and Setup Guide for Client*, and *FUJITSU Enterprise Postgres Installation and Setup Guide for Server Assistant,* which can be found at *FUJITSU Enterprise Postgres 12 on IBM LinuxONE*.

The flow of the implementation has the following high-level steps:

1. Start with the simple implementation of a stand-alone FUJITSU Enterprise Postgres Advanced Edition on a Red Hat Enterprise Linux/SUSE Linux Enterprise Server VM running on LPAR1.
2. Install and configure the FUJITSU Enterprise Postgres Server Assistant component on the arbitration server on LPAR3.

3. Install and configure a FUJITSU Enterprise Postgres Secondary (Standby) instance in a Red Hat Enterprise Linux/SUSE Linux Enterprise Server VM on LPAR2.
4. Install and configure a FUJITSU Enterprise Postgres read-replica (standby) instance in a Red Hat Enterprise Linux or SUSE Linux Enterprise Server VM on LPAR1. The purpose of this read-replica is to enable scaling out of FUJITSU Enterprise Postgres and reducing the load on the primary database instance.
5. Install and configure a backup (standby) instance in a Red Hat Enterprise Linux or SUSE Linux Enterprise Server VM on LPAR3. In this installation, LPAR3 is on IBM LinuxONE machine B.

Installing and configuring a primary database instance on an IBM LinuxONE guest

To install and configure a primary database instance on an IBM LinuxONE guest, complete the following steps:

1. Run the following commands to install the international components for Unicode libraries through `libicu`. Also, install the library `libnsl`, which provides a transport layer interface to the networking services.

   ```
   # yum install libnsl
   # yum install libicu
   ```

2. Set up `JAVA_HOME` by running the following command. In our example, we used OpenJDK V1.8.0.

   ```
   #export
   JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre"
   ```

3. Mount the FUJITSU Enterprise Postgres 12 Advanced Edition ISO and start the installation, as shown in Example 5-1.

Example 5-1 Active-passive on a two IBM LinuxONE systems distributed across two data centers

```
#mount -t iso9660 -r -o loop /fep12iso/fsep12_ae_linux64.iso /mnt2
#cd /mnt2/SERVER/Linux/packages/r80s390x
# rpm -ivh FJSVfsep-SV-12-1200-0.el8.s390x.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:FJSVfsep-SV-12-1200-0.el8        ################################# [100%]
```

4. Install the WebAdmin component of the FUJITSU Enterprise Postgres, as shown in Example 5-2. WebAdmin provides an easy way to create and manage FUJITSU Enterprise Postgres clusters by using a GUI.

Example 5-2 Installing WebAdmin

```
#mount -t iso9660 -r -o loop /fep12iso/fsep12_ae_linux64.iso /mnt2
#cd /mnt2/WEBADMIN/Linux/packages/r80s390x
# rpm -ivh FJSVfsep-WAD-12-1200-0.el8.s390x.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:FJSVfsep-WAD-12-1200-0.el8       ################################# [100%]
```

5. Install the MC on the primary instance by using the commands that are shown in Example 5-3 on page 185.

Example 5-3   Installing Mirroring Controller

```
#/opt/fsepv12server64/bin/mc_update_jre_env
INFO:
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre/lib/s390x/server/l
ibjvm.so is in use.
```

6. Create the database administrator user ID `fsepuser` and set the password by using the following commands:

   ```
   #useradd -m fsepuser
   #passwd fsepuser
   ```

7. After the user ID is created, prepare the directories to store the FUJITSU Enterprise Postgres binary files, transaction logs, and backups, as shown in Example 5-4.

Example 5-4   Creating the directories to store binary files, transaction logs, and backups

```
# mkdir -p /database/inst1
# chown -R fsepuser:fsepuser /database
# chmod 700 /database/inst1
# mkdir -p /transaction/inst1
# chown -R fsepuser:fsepuser /transaction
# chmod 700 /transaction/inst1
# mkdir -p /backup/inst1
# chown -R fsepuser:fsepuser /backup
# chmod 700 /backup/inst1
```

   For more information about the role of each directory this is created in this step, see “Preparing Directories for Resource Deployment” in *FUJITSU Enterprise Postgres 12 Installation and Setup Guide for Server*.

8. To enable communication, grant access to ports 27500, 27515, 27516, 27540, and 27541 by running the commands that are shown in Example 5-5. Port 27500 is used by FUJITSU Enterprise Postgres, ports 27515 and 27516 are used by WebAdmin, and ports 27540 and 27541 are used by the MC.

Example 5-5   Granting access to ports

```
# vim /etc/firewalld/services/fsep.xml
# cat /etc/firewalld/services/fsep.xml
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>FSEP12</short>
  <description>FUJITSU Enterprise Postgres Database Server</description>
  <port protocol="tcp" port="27500"/>
  <port protocol="tcp" port="27515"/>
  <port protocol="tcp" port="27516"/>
  <port protocol="tcp" port="27540"/>
  <port protocol="tcp" port="27541"/>
</service>
# firewall-cmd --reload
# firewall-cmd --get-services | grep fsep
# firewall-cmd --permanent --zone=public --add-service=fsep
# firewall-cmd --reload
# firewall-cmd --list-all | grep fsep
```

9. Configure the environment variables by using the commands that are shown in Example 5-6.

Example 5-6 Configuring the environment variables

```
# su - fsepuser
$ PATH=/opt/fsepv12server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

10. Create the database instance `inst1`, as shown in Example 5-7.

Example 5-7 Creating a database instance

```
$ initdb -D /database/inst1 --waldir=/transaction/inst1 --lc-collate="C"
--lc-ctype="C" --encoding=UTF8
The files belonging to this database system will be owned by user "fsepuser".
This user must also own the server process.
The database cluster will be initialized with locales
  COLLATE:  C
  CTYPE:    C
  MESSAGES: en_US.UTF-8
  MONETARY: en_US.UTF-8
  NUMERIC:  en_US.UTF-8
  TIME:     en_US.UTF-8
The default text search configuration will be set to "english". (15541)
Data page checksums are disabled. (18153)
fixing permissions on existing directory /database/inst1 ... ok (15516)
fixing permissions on existing directory /transaction/inst1 ... ok (15516)
creating subdirectories ... ok (15516)
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ...  (19842)America/New_York
creating configuration files ... ok (15516)
running bootstrap script ... ok (15516)
performing post-bootstrap initialization ... ok (15516)
syncing data to disk ... ok (15516)
initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.
Success. You can now start the database server using the following command:
    pg_ctl -D /database/inst1 -l logfile start
```

11. After the first database instance is created, configure the instance to accept remote connections (Example 5-8 on page 187). By default, FUJITSU Enterprise Postgres allows connections only from the local machine on which the database server is installed. Also, turn on the logging collector to capture the log messages into the log files. This parameter can be set only at server start. For more information about the parameters in the `postgresql.conf` file, see the official PostgreSQL documentation.

Example 5-8 Configuration to accept remote connections

```
$ vim /database/inst1/postgresql.conf
listen_addresses = '*'
port = 27500
logging_collector = on
```

12. Configure the host-based authentication by setting up the `pg_hba.conf` file. The parameters in pg_hba.conf control the authentication process in PostgreSQL and FUJITSU Enterprise Postgres (Example 5-9).

Example 5-9 Configuring the authentication

```
$ vim /database/inst1/pg_hba.conf
host    all             all             XXX.XX.XX.0/24     trust
```

After completing the configuration, all the connections originating from the XXX.XX.XX.0 network can access the primary database instance inst1. Ensure that you provide the accurate network range in place of XXX.XX.XX.0/24.

Note: FUJITSU Enterprise Postgres supports several authentication methods, including TRUST, IDENT, PEER, Generic Security Standard Application Programming Interface (GSSAPI), LDAP, Salted Challenge Response Authentication Mechanism (SCRAM), SHA-256, MD5, and password. For more information about host-based authentication methods, see Chapter 4, “Data security with Transparent Data Encryption, Data Masking, and Audit Log” on page 113 and the FUJITSU Enterprise Postgres and PostgreSQL documentation.

13. Start the database instance `inst1` on the primary database server (Example 5-10).

Example 5-10 Starting the database instance

```
$ pg_ctl start -D /database/inst1
waiting for server to start....2020-12-25 00:19:37.058 EST [2154304] LOG:
starting PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (GCC) 8.3.1
20190507 (Red Hat 8.3.1-4), 64-bit
2020-12-25 00:19:37.060 EST [2154304] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2020-12-25 00:19:37.060 EST [2154304] LOG:  listening on IPv6 address "::", port
27500
2020-12-25 00:19:37.061 EST [2154304] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2020-12-25 00:19:37.067 EST [2154304] LOG:  redirecting log output to logging
collector process
2020-12-25 00:19:37.067 EST [2154304] HINT:  Future log output will appear in
directory "log".
 done
server started
```

14. After the database instance starts, verify the connection by running the **psql** command (see Example 5-11) from the primary database server. Enter the IP address of the primary server in place of XXX.XX.XX.156 in the **psql** command. Press the Enter key and wait for the `postgres=#` prompt.

Example 5-11 Testing connection by using psql

```
$ psql -h <XXX.XX.XX.156> -p 27500 -d postgres -U fsepuser
psql (12.1)
Type "help" for help.
postgres=#
```

15. Stop the primary database instance and wait for the `server stopped` message, as shown in Example 5-12.

Example 5-12 Stopping the database instance

```
$ pg_ctl stop -D /database/inst1
waiting for server to shut down.... done
server stopped
```

16. On the arbitration server, set up `JAVA_HOME` for OpenJDK by running the following command:

```
#export
JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre"
```

Installing and configuring the arbitration server on an IBM LinuxONE guest

To install and configure an arbitration server on a Linux guest, complete the following steps:

1. Onthe arbitration server, install the Server Assistant software, which provides the quorum service that prevents a split-brain scenario between the primary database instance and the secondary (standby) database instance, as shown in Example 5-13.

Example 5-13 Installing Server Assistant on the arbitration server

```
# mount -t iso9660 -r -o loop /fep12iso/fsep12_assist.iso /mnt2
# cd /mnt2/ARBITER/Linux/packages/r80s390x
# rpm -ivh FJSVfsep-ARB-12-1200-0.el8.s390x.rpm
Verifying...                       ################################# [100%]
Preparing...                       ################################# [100%]
Updating / installing...
   1:FJSVfsep-ARB-12-1200-0.el8    ################################# [100%]
```

2. Configure the MC by running the command that is shown in Example 5-14.

Example 5-14 Configuring Mirroring Controller on the arbitration server

```
# /opt/fsepv12assistant/bin/mc_update_jre_env
INFO:
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre/lib/s390x/server/l
ibjvm.so is in use.
```

3. Create the administrator user ID `fsepuser` and set the password by running the following commands:

```
#useradd -m fsepuser
#passwd fsepuser
```

4. Prepare the `arbiter/inst1` directory and change the ownership to `fsepuser` by running the following commands:

   ```
   # mkdir /arbiter/inst1
   # chown -R fsepuser:fsepuser /arbiter
   # chmod 700 /arbiter/inst1
   ```

5. Grant access to port 27541, as shown in Example 5-15.

Example 5-15 Granting access to port 27541

```
# vim /etc/firewalld/services/fsep.xml
# cat /etc/firewalld/services/fsep.xml
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>FSEP12</short>
  <description>FUJITSU Enterprise Postgres Assistant Server</description>
  <port protocol="tcp" port="27541"/>
</service>
# firewall-cmd --reload
# firewall-cmd --get-services | grep fsep
# firewall-cmd --permanent --zone=public --add-service=fsep
# firewall-cmd --reload
# firewall-cmd --list-all | grep fsep
```

6. Configure the environment variables, as shown in Example 5-16.

Example 5-16 Configuring environment variables

```
# su - fsepuser
$ PATH=/opt/fsepv12assistant/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12assistant/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12assistant/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

7. Configure the `network.conf` file by updating the identifiers for primary and secondary database instances with IP addresses and the port number of the MC on the primary and secondary database servers. In Example 5-17, our primary database server is named `inst1p` and the secondary database server is named `inst1s`. Replace xxx.xx.xx.xxx with the IP addresses of the primary and secondary database guests in your environment.

Example 5-17 Configuring the network.conf file

```
$ cp /opt/fsepv12assistant/share/mcarb_network.conf.sample
/arbiter/inst1/network.conf
$ vim /arbiter/inst1/network.conf
inst1p XXX.XX.XX.176 27541
inst1s XXX.XX.XX.177 27541
```

8. Configure the arbitration configuration file (`arbitration.conf`) and define the information that is related to controlling the arbitration process. You can refer to the sample `arbitration.conf` file that is available in the installation directory `->share->` `mcarb_arbitration.conf.sample`. Replace the xxx.xx.xx.xxx in the `my_address` parameter with the IP address of your arbitration server, as shown in Example 5-18.

Example 5-18 Configuring the arbitration.conf file

```
$ cp /opt/fsepv12assistant/share/mcarb_arbitration.conf.sample
/arbiter/inst1/arbitration.conf
$ vim /arbiter/inst1/arbitration.conf
port=27541
my_address='XXX.XX.XX.179'
fencing_command='/arbiter/inst1/execute_fencing.sh'
```

Note: For more information about the parameters of the `arbitration.conf` file, see the *FUJITSU Enterprise Postgres Cluster Operations Guide*.

9. Copy the sample fencing command and configure it for the arbitration server, as shown in Example 5-19.

Example 5-19 Copying the sample database server fencing script

```
$ cp /opt/fsepv12assistant/share/mcarb_execute_fencing.sh.sample
/arbiter/inst1/execute_fencing.sh
$ vim /arbiter/inst1/execute_fencing.sh
```

10. Change the file permissions by running the following command:

```
$ chmod -R 700 /arbiter/inst1
```

11. Start the arbitration process, as shown in Example 5-20.

Example 5-20 Starting the arbitration process

```
$ mc_arb start -M /arbiter/inst1
starting Mirroring Controller Arbitration process (MCR00045)
Mirroring Controller Arbitration process started (MCR00046)
```

12. Check the status of the MC on `inst1p` and `inst1s` (primary and secondary database instances). The expected status is `offline` because the MC on the primary and the secondary database instances will be updated with the information about the arbitration server and Server Assistant that is installed on arbitration server (Example 5-21).

Example 5-21 Check the Mirroring Controller status on the primary and secondary (standby) instances

```
$ mc_arb status -M /arbiter/inst1
server_id  host           status
---------------------------------------
inst1p    XXX.XX.XX.176  offline
inst1s    XXX.XX.XX.177  offline
```

13. Because the status that is shown in Example 5-21 was expected, update the `pg_hba.conf` file with an entry for the replication on the primary database server (Example 5-22 on page 191).

Example 5-22 Setting up authentication by updating the pg_hba.conf file

```
$ vim /database/inst1/pg_hba.conf
host    all            all            XXX.XX.XX.0/24      trust
host    replication    all            XXX.XX.XX.0/24      trust
```

14. Continuing with the primary database instance, set up `postgresql.conf` to configure the primary database instance for replication. The parameter **`synchronous_standby_names`** now has the name of the secondary database instance `inst1s` (Example 5-23).

Example 5-23 Configuring replication

```
$ vim /database/inst1/postgresql.conf
listen_addresses = '*'
port = 27500
logging_collector = on
wal_level=replica
max_wal_senders=10
synchronous_standby_names='inst1s'
hot_standby=on
wal_log_hints=on
restart_after_crash=off
recovery_target_timeline = 'latest'
```

Note: The parameter **`wal_log_hints`** is required to use **`pg_rewind`** to recover instance switching. The parameter **`restart_after_crash`** is set to `off` so that the MC can control the database instance. For more information about these parameters, see the official PostgreSQL documentation.

15. On the primary database server, create the MC directory and change the ownership to `fsepuser` user ID, as shown in Example 5-24.

Example 5-24 Creating the Mirroring Controller directory on the primary database server

```
# mkdir -p /mcdir/inst1
# chown -R fsepuser:fsepuser /mcdir
# chmod 700 /mcdir/inst1
```

16. Configure the environment variables on the primary database server, as shown in Example 5-25.

Example 5-25 Configuring environment variables

```
# su - fsepuser
$ PATH=/opt/fsepv12server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

17. Update the `network.conf` file on the primary database server with the MC IP address, port numbers from the secondary database (standby) instance, and the arbitration server. (The secondary database (standby) instance is not installed and configured yet.) See Example 5-26.

Example 5-26 Updating the network.conf file on the primary database server

```
$ cp -p /opt/fsepv12server64/share/mc_network.conf.sample
/mcdir/inst1/network.conf
$ vim /mcdir/inst1/network.conf
inst1p XXX.XX.XX.156,XXX.XX.XX.176 27540,27541 server
inst1s XXX.XX.XX.161,XXX.XX.XX.177 27540,27541 server
arbiter XXX.XX.XX.179 27541 arbiter
```

Note: Ensure that the port numbers set for the primary server, standby server, and arbitration server do not conflict with other software. In addition, when the arbitration server is used for automatic degradation, use a network in which the arbitration network is not affected by a network failure in the admin network.

18. In the primary database server, configure the `server-identifier` configuration file to define the information that is related to MC monitoring and control. This file ensures that appropriate automated action is taken when a heartbeat abnormality occurs between the primary database instance and the secondary database (standby) instance.

    Because we want the arbitration server to perform automatic degradation, we set the parameter **`hearbeat_error_action`** to `arbitration`. In Example 5-27, we copy the sample server identifier file `mc_server.conf.sample` to the MC directory of the primary database instance `inst1`, and name the copied file `inst1p.conf`. We use `inst1p` because it is the server identifier name that was used when we configured the `network.conf` file in step 17.

Example 5-27 Configuring the server identifier configuration file

```
$ cp /opt/fsepv12server64/share/mc_server.conf.sample /mcdir/inst1/inst1p.conf
$ vim /mcdir/inst1/inst1p.conf
db_instance='/database/inst1'
heartbeat_error_action = arbitration
```

Note: There are more than 30 parameters in the server identifier configuration file. For more information about these parameters, see Appendix A., “Server Configuration File for the Database Servers” in *FUJITSU Enterprise Postgres 12 Cluster Operations Guide*.

19. Change the file permission by running the following command:

```
$ chmod -R 700 /mcdir/inst1
```

20. Start the MC process on the primary database server, as shown in Example 5-28.

Example 5-28 Starting the Mirroring Controller process

```
$ mc_ctl start -M /mcdir/inst1
starting Mirroring Controller (MCA00038)
requesting arbitration server "arbiter" to connect (MCA00149)
succeeded in connection with arbitration server "arbiter" (MCA00151)
enabled failover  target server:"inst1p" (MCA00046)
Mirroring Controller started (MCA00039)
```

21. Monitor the MC status on the primary database server, as shown in Example 5-29.

Example 5-29 Checking the Mirroring Controller status

```
$ mc_ctl status --arbiter -M /mcdir/inst1
arbiter_id  host          status
---------------------------------------
arbiter     XXX.XX.XX.179  online
$ mc_ctl status -M /mcdir/inst1
mirroring status
-----------------
not-switchable
server_id  host_role  host          host_status  db_proc_status      disk_status
---------------------------------------------------------------------------------
inst1p    primary    XXX.XX.XX.156  normal       abnormal(wal_sender)  normal
inst1s    unknown    XXX.XX.XX.161  normal       unknown               unknown
```

Note: When the secondary database instance (standby) setup is complete, the status of `inst1p` changes from `abnormal` to `normal`.

22.Go to the arbitration server and check the MC status (Example 5-30).

Example 5-30 Checking the Mirroring Controller status on the arbitration server

```
$ mc_arb status -M /arbiter/inst1
server_id  host          status
---------------------------------------
inst1p    XXX.XX.XX.176  online
inst1s    XXX.XX.XX.177  offline
```

Note: The status mode displays the connection status of the MC arbitration process with the MC processes of the primary server and standby server. To learn more about the **`mc_arb`** and **`mc_ctl`** commands, see Chapter 4, “MC Commands”, in *FUJITSU Enterprise Postgres Reference Manual*.

Up to this point, we have installed and configured the primary database server and the arbitration server. To complete the HA configuration, we will next install and configure the secondary database server (also called the standby server).

Installing and configuring the secondary database server on an IBM LinuxONE guest

The initial steps for installing and configuring FUJITSU Enterprise Postgres as a secondary database server are same as the steps of installing and configuring the primary database server:

1. To install and configure FUJITSU Enterprise Postgres as a secondary database instance, perform steps 1 on page 184 - 8 on page 185 in “Installing and configuring a primary database instance on an IBM LinuxONE guest” on page 184. After you complete these initial steps, run the following commands to prepare the directory for the MC on the secondary database instance, and then continue with the following steps:

   ```
   # mkdir -p /mcdir/inst1
   # chown -R fsepuser:fsepuser /mcdir
   # chmod 700 /mcdir/inst1
   ```

2. Because the network configuration files for the secondary database server are the same as those ones for the primary database server, copy the file `network.conf` from the

primary database server by running the following command. Replace XXX.XX.XX.156 with the IP address of your primary database server.

```
# scp fsepuser@XXX.XX.XX.156:/mcdir/inst1/network.conf /mcdir/inst1/
```

3. The identifier file on the secondary database (standby) server is the same as the one on the primary database server. Copy the server identifier file `inst1p.conf` from the primary database server by running the following command:

   ```
   #scp fsepuser@XXX.XX.XX.156:/mcdir/inst1/inst1p.conf
   ```

   The file name changed to `inst1s.conf`.

4. Change the file permissions by running the following commands:

   ```
   # chown -R fsepuser:fsepuser /mcdir
   # chmod -R 700 /mcdir/inst1
   ```

5. Configure the environment variables on the secondary database (standby) server, as shown in Example 5-31.

Example 5-31 Configuring the environment variables on the standby database server

```
# su - fsepuser
$ PATH=/opt/fsepv12server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

6. Use the **pg_basebackup** utility on the secondary database (standby) server to create the database instance from the primary database instance, which serves as the master copy. Replace XXX.XX.XX.156 with your primary database server IP address, as shown in Example 5-32.

Example 5-32 Creating the database instance on the standby server

```
$ pg_basebackup -D /database/inst1 -X fetch --waldir=/transaction/inst1 --progress
--verbose -R --dbname='application_name=inst1s' -h XXX.XX.XX.156 -p 27500
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/2000028 on timeline 1
41390/41390 kB (100%), 1/1 tablespace
(15555)
pg_basebackup: write-ahead log end point: 0/2000100
pg_basebackup: syncing data to disk ...
pg_basebackup: base backup completed
```

7. Configure the `postgresql.conf` file on the secondary database (standby) instance by updating the **synchronous_standby_names** parameter to point to the primary database instance (also called the master database instance) by running the following commands:

   ```
   $ vim /database/inst1/postgresql.conf
   synchronous_standby_names='inst1p'
   ```

8. Start the MC process on the secondary database (standby) instance, as shown in Example 5-33 on page 195.

Example 5-33 Starting the Mirroring Controller process on the standby database server

```
$ mc_ctl start -M /mcdir/inst1
starting Mirroring Controller (MCA00038)
requesting arbitration server "arbiter" to connect (MCA00149)
succeeded in connection with arbitration server "arbiter" (MCA00151)
enabled failover target server:"inst1s" (MCA00046)
Mirroring Controller started (MCA00039)
```

9. After starting the MC on the secondary database (standby) instance, check the status, as shown in Example 5-34.

Example 5-34 Checking the Mirroring Controller status

```
$ mc_ctl status --arbiter -M /mcdir/inst1
arbiter_id  host           status
---------------------------------------
arbiter     XXX.XX.XX.179  online
$ mc_ctl status -M /mcdir/inst1
mirroring status
-----------------
switchable
server_id  host_role  host           host_status  db_proc_status  disk_status
-----------------------------------------------------------------------------
inst1p    primary    XXX.XX.XX.156  normal       normal          normal
inst1s    standby    XXX.XX.XX.161  normal       normal          normal
```

10. Check the status of the arbitration process on the arbitration server, as shown in Example 5-35.

Example 5-35 Checking the arbitration process on the arbitration server

```
$ mc_arb status -M /arbiter/inst1
server_id  host           status
--------------------------------------
inst1p    XXX.XX.XX.176  online
inst1s    XXX.XX.XX.177  online
```

In the preceding steps, we configured the primary database instance, an arbitration server with FUJITSU Server Assistant and MC, and a secondary server. We also initiated synchronous streaming replication between the primary database instance and the secondary database (standby) instance. In the following steps, we check the status of the replication:

1. On the secondary database (standby) instance, run the **psql** utility and check that you can connect to the secondary database (standby) instance. Replace XXX.XX.XX.161 with the IP address of the secondary database (standby) instance in the command that is shown in Example 5-36.

Example 5-36 Testing the connectivity to the standby database server

```
$ psql -h XXX.XX.XX.161 -p 27500 -d postgres -U fsepuser
psql (12.1)
Type "help" for help.
postgres=#
```

2. On the primary database instance, confirm that the primary database instance and the secondary database (standby) instance are synchronized with each other by creating the `pgbench` database and the underlying tables, as shown in Example 5-37.

Example 5-37 Configuring the pgbench database on the primary database instance

```
$ pgbench -i postgres
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...
100000 of 100000 tuples (100%) done (elapsed 0.07 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done.
```

Note: The command that is shown in Example 5-37 on page 196 creates the `pgbench` tables on the primary database instance and loads the data. Because the HA is configured and assuming synchronous streaming replication is active, the tables that are created on the primary database instance should now be available for read-only queries on the secondary database (standby) instance. To verify this situation, proceed to the next step.

3. On the secondary database (standby) instance, connect to **`psql`** and list the database objects to confirm that the replication is active. Replace XXX.XX.XX.161 with the IP address of your secondary database (standby) instance in the **`psql`** command that is shown in Example 5-38.

Example 5-38 Verifying that the pgbench tables that were replicated from the primary to the standby database instance

```
$ psql -h XXX.XX.XX.161 -p 27500 -d postgres -U fsepuser
Password for user fsepuser:
psql (12.1)
Type "help" for help.
postgres=# \d
              List of relations
 Schema |       Name       | Type  |  Owner
--------+------------------+-------+----------
 public | pgbench_accounts | table | fsepuser
 public | pgbench_branches | table | fsepuser
 public | pgbench_history  | table | fsepuser
 public | pgbench_tellers  | table | fsepuser
(Four row)
```

The `pgbench` tables are now available on the secondary database (standby) instance.

Installing and configuring a read-replica database instance and backup database instance with a cascading secondary database (standby) instance

The steps to create and add standby database instances are like the steps that we followed in “Installing and configuring the secondary database server on an IBM LinuxONE guest” on page 193 to create the secondary database (standby) instance, but there are a few differences. The most important difference is that the replication between the secondary database (standby) instance and the cascaded database instance is always asynchronous streaming replication.

In this example, we start with creating the read-replica database instance on the IBM LinuxONE LPAR LPAR1 on a Linux guest followed by a backup database instance on the IBM LinuxONE LPAR LPAR3, also on a Linux guest:

1. To create a read-replica database instance setup on LPAR1, run the commands that are shown in Example 5-39 to install the international components for Unicode libraries through `libicu` on VM RDBKPGR2 [IP XXX.XX.XX.157]. We also install the library `libnsl`, which provides a transport layer interface to the networking services.

Example 5-39 Installing the libraries libnsl and libicu

```
# yum install libnsl
# yum install libicu
```

2. Set up `JAVA_HOME`. For this example, we used OpenJDK V1.8.0 and ran the following command:

   ```
   #export
   JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre"
   ```

3. Mount the FUJITSU Enterprise Postgres 12 Advanced Edition ISO and start the installation, as shown in Example 5-40.

Example 5-40 Installing FUJITSU Enterprise Postgres

```
#mount -t iso9660 -r -o loop /fep12iso/fsep12_ae_linux64.iso /mnt2
#cd /mnt2/SERVER/Linux/packages/r80s390x
# rpm -ivh FJSVfsep-SV-12-1200-0.el8.s390x.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:FJSVfsep-SV-12-1200-0.el8        ################################# [100%]
```

4. Create a database administrator user ID `fsepuser` and set the password by using the following commands:

   ```
   #useradd -m fsepuser
   #passwd fsepuser
   ```

5. After the user ID is created, prepare the directories to store the FUJITSU Enterprise Postgres binary files, transaction logs, and backups, as shown in Example 5-41. For more information about the role of each directory that is created in this step, see “Preparing Directories for Resource Deployment” in the *Installation and Setup Guide for Server*.

Example 5-41 Preparing the directories

```
# mkdir -p /database/inst1
# chown -R fsepuser:fsepuser /database
# chmod 700 /database/inst1
# mkdir -p /transaction/inst1
```

```
# chown -R fsepuser:fsepuser /transaction
# chmod 700 /transaction/inst1
# mkdir -p /backup/inst1
# chown -R fsepuser:fsepuser /backup
# chmod 700 /backup/inst1
```

6. To enable communications, grant access to port 27500 by running the commands that are shown in Example 5-42.

Example 5-42 Granting access to a port

```
# vim /etc/firewalld/services/fsep.xml
# cat /etc/firewalld/services/fsep.xml
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>FSEP12</short>
  <description>FUJITSU Enterprise Postgres Database Server</description>
  <port protocol="tcp" port="27500"/>
</service>
# firewall-cmd --reload
# firewall-cmd --get-services | grep fsep
# firewall-cmd --permanent --zone=public --add-service=fsep
# firewall-cmd --reload
# firewall-cmd --list-all | grep fsep
```

7. Configure the environment variables, as shown in Example 5-43.

Example 5-43 Configuring environment variables

```
# su – fsepuser
$ PATH=/opt/fsepv12server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

8. Use the **pg_basebackup** utility on the read-replica database server to create the database instance from the secondary database (standby) instance serving as the master copy. Replace XXX.XX.XX.161 with your secondary database (standby) server IP address (Example 5-44).

Example 5-44 Creating a read-replica by using a backup of a standby database instance

```
$ pg_basebackup -D /database/inst1 -X fetch --waldir=/transaction/inst1 --progress
--verbose -R --dbname='application_name=inst1cs' -h XXX.XX.XX.161 -p 27500
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/5000060 on timeline 1
56872/56872 kB (100%), 1/1 tablespace
(15555)
pg_basebackup: write-ahead log end point: 0/5C933A0
pg_basebackup: syncing data to disk ...
pg_basebackup: base backup completed
```

9. Configure the `postgresql.conf` file on the read-replica database instance by updating **synchronous_standby_names** with blanks, as shown in Example 5-45 on page 199.

Example 5-45 Configuring the synchronous_standby_names parameter in the postgresql.conf file

```
$ vim /database/inst1/postgresql.conf
synchronous_standby_names=''
```

> **Note:** Because the standby servers are connected to other standby servers in cascaded mode, they can use only asynchronous streaming replication. So, the parameter **`synchronous_standby_names`** is set without a name, which differs from the secondary database (standby) instance configuration.

10. On the read-replica database server, start FUJITSU Enterprise Postgres (Example 5-46).

Example 5-46 Starting the read-replica database server

```
$ pg_ctl start -D /database/inst1
waiting for server to start....2020-12-25 01:31:25.327 EST [340540] LOG:  starting
PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (GCC) 8.3.1 20190507 (Red Hat
8.3.1-4), 64-bit
2020-12-25 01:31:25.329 EST [340540] LOG:  listening on IPv4 address "0.0.0.0", port 27500
2020-12-25 01:31:25.329 EST [340540] LOG:  listening on IPv6 address "::", port 27500
2020-12-25 01:31:25.332 EST [340540] LOG:  listening on Unix socket "/tmp/.s.PGSQL.27500"
2020-12-25 01:31:25.339 EST [340540] LOG:  redirecting log output to logging collector
process
2020-12-25 01:31:25.339 EST [340540] HINT:  Future log output will appear in directory
"log".
 done
server started
```

11. Confirm that the replication is functional by initiating a **`psql`** CLI and listing the `pgbench` tables. Replace XXX.XX.XX.157 with your read-replica database server IP address in the command that is shown in Example 5-47.

Example 5-47 Confirming replication by checking the table definitions in the read-replica

```
$ psql -h XXX.XX.XX.157 -p 27500 -d postgres -U fsepuser
psql (12.1)
Type "help" for help.
postgres=# \d
              List of relations
 Schema |       Name       | Type  |  Owner
--------+------------------+-------+----------
 public | pgbench_accounts | table | fsepuser
 public | pgbench_branches | table | fsepuser
 public | pgbench_history  | table | fsepuser
 public | pgbench_tellers  | table | fsepuser
(Four row)
postgres=#
```

The setup is complete. Now, we add a backup database instance on LPAR3 in a Linux guest by completing the following steps:

1. To create a backup database instance on LPAR3, run the following commands to install the international components for Unicode libraries through `libicu`. Also, install the library `libnsl`, which provides a transport layer interface to the networking services.

   ```
   # yum install libnsl
   # yum install libicu
   ```

2. Set up `JAVA_HOME` by running the following command. For this example, we used OpenJDK V1.8.0.

   ```
   #export
   JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre"
   ```

3. Mount the FUJITSU Enterprise Postgres 12 Advanced Edition ISO and start the installation, as shown in Example 5-48.

Example 5-48 Installing FUJITSU Enterprise Postgres on the backup database server

```
#mount -t iso9660 -r -o loop /fep12iso/fsep12_ae_linux64.iso /mnt2
#cd /mnt2/SERVER/Linux/packages/r80s390x
# rpm -ivh FJSVfsep-SV-12-1200-0.el8.s390x.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:FJSVfsep-SV-12-1200-0.el8        ################################# [100%]
```

4. Create a database administrator user ID `fsepuser` and set the password by running the following commands:

   ```
   #useradd -m fsepuser
   #passwd fsepuser
   ```

5. After the user ID is created, prepare the directories to store the FUJITSU Enterprise Postgres binary files, transaction logs, and backups, as shown in Example 5-49.

Example 5-49 Creating directories to store binary files, transaction logs, and backups

```
# mkdir -p /database/inst1
# chown -R fsepuser:fsepuser /database
# chmod 700 /database/inst1
# mkdir -p /transaction/inst1
# chown -R fsepuser:fsepuser /transaction
# chmod 700 /transaction/inst1
# mkdir -p /backup/inst1
# chown -R fsepuser:fsepuser /backup
# chmod 700 /backup/inst1
```

For more information about the role of each directory that is created in this step, see "Preparing Directories for Resource Deployment" in the *FUJITSU Enterprise Postgres 12 Installation and Setup Guide*.

To enable communications, grant access to the port 27500 by running the commands that are shown in Example 5-50 to allow access by opening the port.

Example 5-50 Granting access to a port

```
# vim /etc/firewalld/services/fsep.xml
# cat /etc/firewalld/services/fsep.xml
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>FSEP12</short>
  <description>FUJITSU Enterprise Postgres Database Server</description>
  <port protocol="tcp" port="27500"/>
</service>
# firewall-cmd --reload
# firewall-cmd --get-services | grep fsep
# firewall-cmd --permanent --zone=public --add-service=fsep
```

```
# firewall-cmd --reload
# firewall-cmd --list-all | grep fsep
```

6. Configure the environment variables, as shown in Example 5-51.

Example 5-51 Configuring environment variables

```
# su - fsepuser
$ PATH=/opt/fsepv12server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

7. Use the **pg_basebackup** utility on the backup database server to create the database instance from the secondary database (standby) instance that serves as the master copy. Replace XXX.XX.XX.161 with your secondary database (standby) server IP address, as shown in Example 5-52.

Example 5-52 Creating a backup database instance by using a backup of the standby database

```
$ pg_basebackup -D /database/inst1 -X fetch --waldir=/transaction/inst1 --progress
--verbose -R --dbname='application_name=inst1bk' -h XXX.XX.XX.161 -p 27500
pg_basebackup: initiating base backup, waiting for checkpoint to complete
pg_basebackup: checkpoint completed
pg_basebackup: write-ahead log start point: 0/5000060 on timeline 1
56872/56872 kB (100%), 1/1 tablespace
(15555)
pg_basebackup: write-ahead log end point: 0/5C933A0
pg_basebackup: syncing data to disk ...
pg_basebackup: base backup completed
```

8. Configure the `postgresql.conf` file on the backup database instance by updating **synchronous_standby_names** with blanks, as shown in Example 5-53.

Example 5-53 Configuring the synchronous_standby_names parameter in postgresql.conf

```
$ vim /database/inst1/postgresql.conf
synchronous_standby_names=''
```

Note: Because the standby servers are connected to other standby servers in cascaded mode, they can use only asynchronous streaming replication. So, the parameter **synchronous_standby_names** is set without any name, which differs from the secondary database (standby) instance configuration.

9. On the backup database server, start FUJITSU Enterprise Postgres, as shown in Example 5-54.

Example 5-54 Starting the backup database instance

```
$ pg_ctl start -D /database/inst1
waiting for server to start....2020-12-25 01:36:37.837 EST [808206] LOG:  starting
PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (GCC) 8.3.1 20190507 (Red
Hat 8.3.1-4), 64-bit
2020-12-25 01:36:37.837 EST [808206] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2020-12-25 01:36:37.837 EST [808206] LOG:  listening on IPv6 address "::", port
27500
```

```
2020-12-25 01:36:37.838 EST [808206] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2020-12-25 01:36:37.845 EST [808206] LOG:  redirecting log output to logging
collector process
2020-12-25 01:36:37.845 EST [808206] HINT:  Future log output will appear in
directory "log".
 done
server started
```

10.Confirm that replication is functional by initiating a **psql** CLI and listing the `pgbench` tables. Replace XXX.XX.XX.157 with your read-replica database server IP address in the command that is shown in Example 5-55.

Example 5-55 Confirming replication by checking table definitions in the read-replica server

```
$ psql -h XXX.XX.XX.157 -p 27500 -d postgres -U fsepuser
psql (12.1)
Type "help" for help.
postgres=# \d
              List of relations
 Schema |       Name        | Type  |  Owner
--------+------------------+-------+----------
 public | pgbench_accounts | table | fsepuser
 public | pgbench_branches | table | fsepuser
 public | pgbench_history  | table | fsepuser
 public | pgbench_tellers  | table | fsepuser
(Four row)
postgres=#
```

You have now set up a HA FUJITSU Enterprise Postgres cluster that has the following components:

- FUJITSU Enterprise Postgres primary database instance on LPAR1
- FUJITSU Enterprise Postgres secondary database (standby) instance on LPAR2
- FUJITSU Enterprise Postgres arbitration server on LPAR3 by using FUJITSU Enterprise Postgres Server Assistant software
- FUJITSU Enterprise Postgres read-replica database instance on LPAR1 with database scale-out through cascading from the secondary database (standby) instance
- FUJITSU Enterprise Postgres Backup database instance on LPAR3 with database scale-out through cascading from the secondary database (standby) instance

5.4.5 Implementing an active-passive high availability architecture by using the WebAdmin GUI

In this section, we provide the steps to configure an active-passive high availability architecture by using the FUJITSU Enterprise Postgres WebAdmin GUI. A few of the steps to prepare the primary database server, secondary database (standby) server, and the arbitration server are the same as described in 5.4.4, “Implementing an active/passive HA architecture when using a command-line interface” on page 182.

This section assumes that FUJITSU Enterprise Postgres is installed and configured on the primary and secondary database (standby) servers and the arbitration server.

1. Configure the FUJITSU Enterprise Postgres primary and secondary database (standby) instances by completing steps 1 on page 184 - 8 on page 185 in “Installing and configuring a primary database instance on an IBM LinuxONE guest” on page 184 for the primary database, and completing steps 1 on page 193 - 10 on page 195 in “Installing and configuring the secondary database server on an IBM LinuxONE guest” on page 193 for the secondary database (standby) server.
2. On the primary database server, create the MC directory and change the ownership to the `fsepuser` user ID, as shown in Example 5-56.

Example 5-56 Creating a Mirroring Controller directory on the primary database server

```
# mkdir -p /mcdir/inst1
# chown -R fsepuser:fsepuser /mcdir
# chmod 700 /mcdir/inst1
```

3. On the arbitration server, run the following commands to install the international components for the Unicode libraries through `libicu`. Also, install the library `libnsl`, which provides a transport layer interface to the networking services.

```
# yum install libnsl
# yum install libicu
```

4. Also on the arbitration server, set up `JAVA_HOME` by using the following command. In our example, we used OpenJDK V1.8.0.

```
#export
JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre"
```

5. Again on the arbitration server, install the WebAdmin component of FUJITSU Enterprise Postgres, as shown in Example 5-57. WebAdmin provides an easy way to create and manage FUJITSU Enterprise Postgres clusters by using a GUI.

Example 5-57 Installing WebAdmin on the arbitration server

```
#mount -t iso9660 -r -o loop /fep12iso/fsep12_ae_linux64.iso /mnt2
#cd /mnt2/WEBADMIN/Linux/packages/r80s390x
# rpm -ivh FJSVfsep-WAD-12-1200-0.el8.s390x.rpm
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:FJSVfsep-WAD-12-1200-0.el8       ################################# [100%]
```

6. Continuing on the arbitration server, install the FUJITSU Enterprise Postgres Server Assistant software, as shown in Example 5-58.

Example 5-58 Installing Server Assistant on the arbitration server

```
# mount -t iso9660 -r -o loop /fep12iso/fsep12_assist.iso /mnt2
# cd /mnt2/ARBITER/Linux/packages/r80s390x
# rpm -ivh FJSVfsep-ARB-12-1200-0.el8.s390x.rpm
```

7. Configure the MC on the arbitration server (Example 5-59).

Example 5-59 Configuring the Mirroring Controller

```
# /opt/fsepv12assistant/bin/mc_update_jre_env
INFO:
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.s390x/jre/lib/s390x/server/l
ibjvm.so is in use.
```

8. Create the administrator user ID `fsepuser` and set the password by using the following commands:

   ```
   #useradd -m fsepuser
   #passwd fsepuser
   ```

9. Prepare the directory `arbiter/inst1` and change the ownership to `fsepuser`, as shown in Example 5-60.

Example 5-60 Preparing the directory

```
# mkdir /arbiter/inst1
# chown -R fsepuser:fsepuser /arbiter
# chmod 700 /arbiter/inst1
```

10. Grant access to ports 27515, 27516, and 27541, as shown in Example 5-61.

Example 5-61 Granting access to ports

```
# vim /etc/firewalld/services/fsep.xml
# cat /etc/firewalld/services/fsep.xml
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>FSEP12</short>
  <description>FUJITSU Enterprise Postgres Assistant Server</description>
  <port protocol="tcp" port="27515"/>
  <port protocol="tcp" port="27516"/>
  <port protocol="tcp" port="27541"/>
</service>
# firewall-cmd --reload
# firewall-cmd --get-services | grep fsep
# firewall-cmd --permanent --zone=public --add-service=fsep
# firewall-cmd --reload
# firewall-cmd --list-all | grep fsep
```

11. Again on the arbitration server, set up the hostnames (Example 5-62):

Example 5-62 Setting up hostnames

```
# vim /etc/hosts
XXX.XX.XX.156 inst1Primary
XXX.XX.XX.161 inst1Standby
```

12. Set up WebAdmin by running the following command on the primary, secondary (standby), and the arbitration servers:

   ```
   # cd /opt/fsepv12webadmin/sbin/
   # ./WebAdminSetup
   ```

13. As shown in Figure 5-12 on page 205, we access WebAdmin from the arbitration server [IP XXX.XX.XX.179].

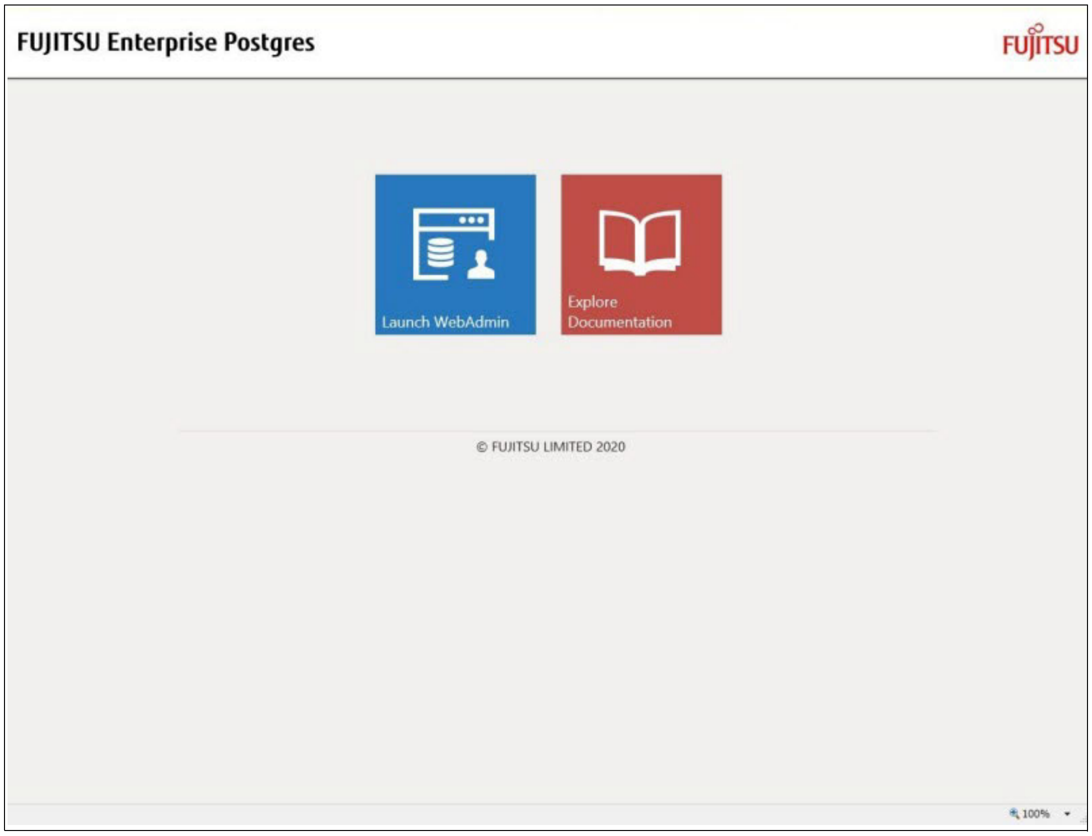


Figure 5-12 WebAdmin welcome page

14. Enter the user ID `fsepuser` and password, as shown in Figure 5-13.

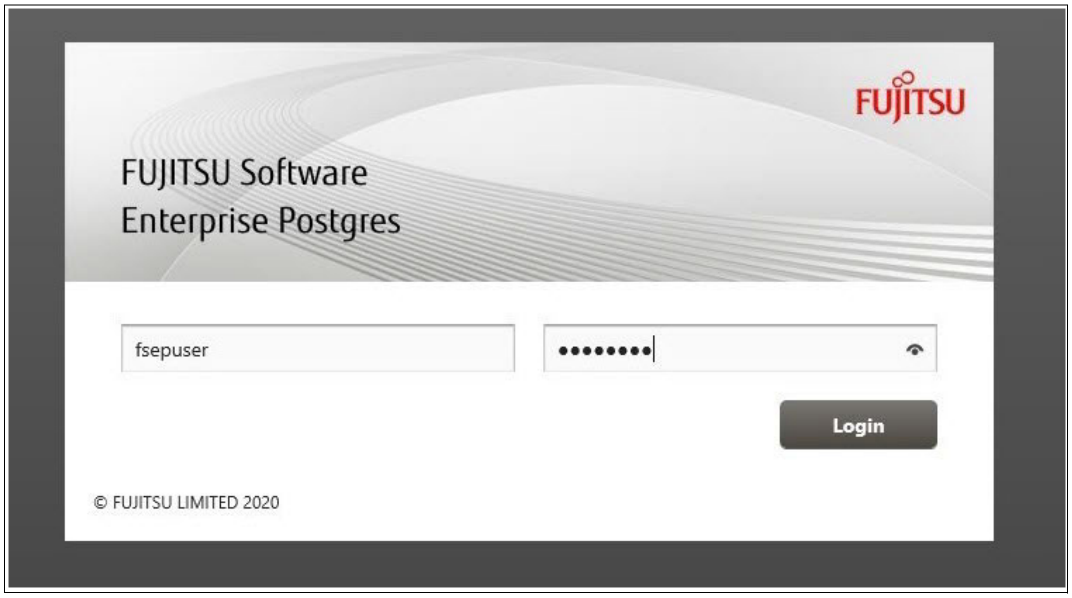


Figure 5-13 WebAdmin login page

15. Create the database instances by following the instructions in the web page that is shown in Figure 5-14.

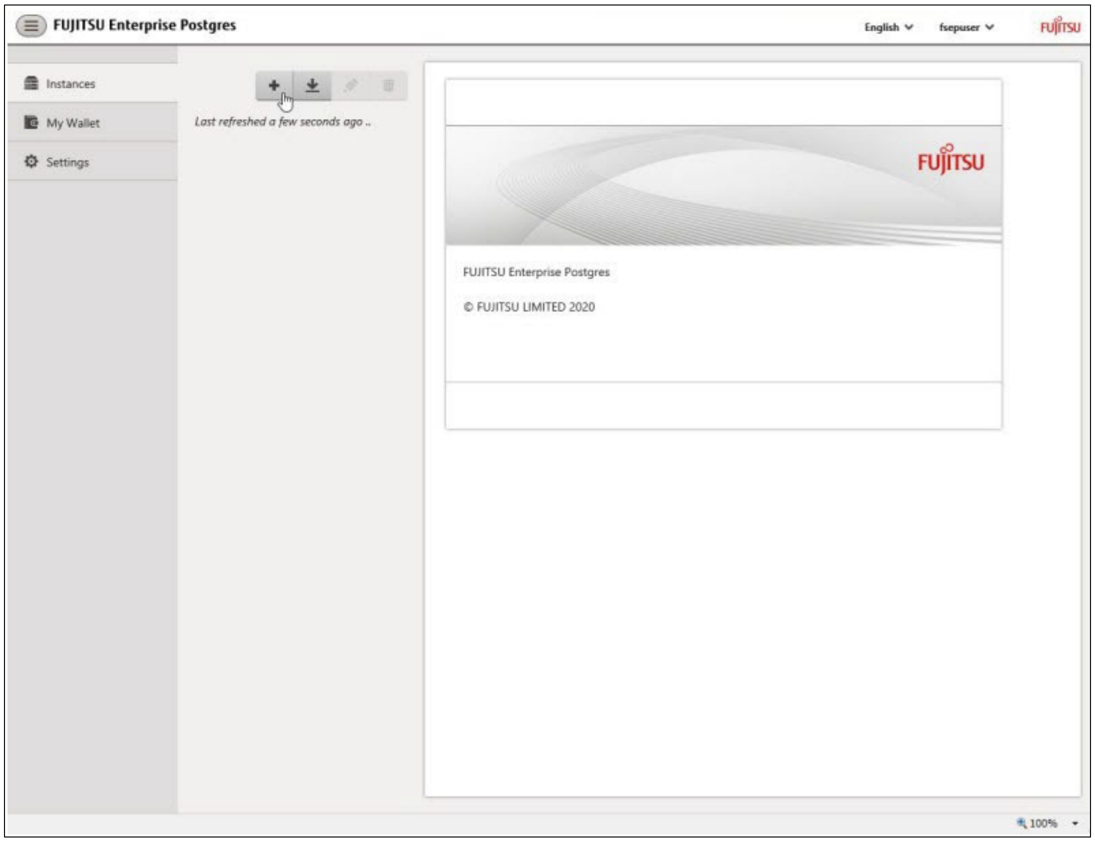
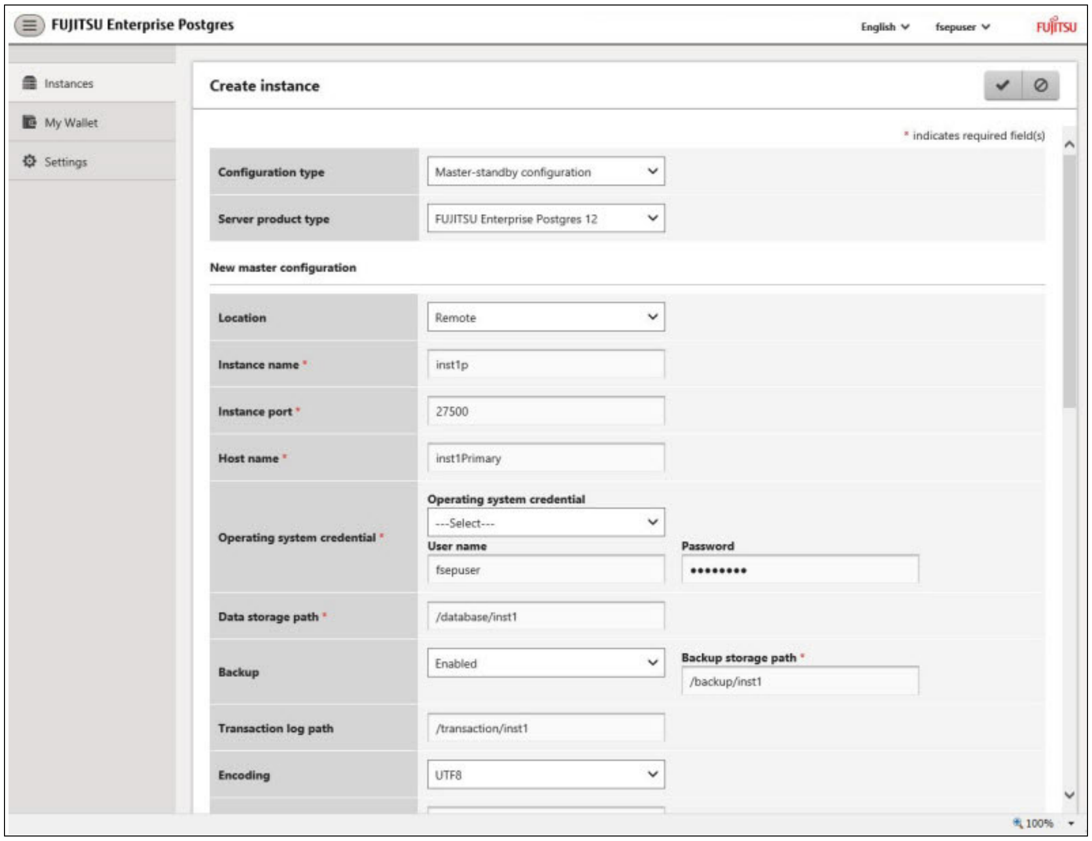

FUJITSU Enterprise Postgres
English
fsepuser
Instances
My Wallet
Settings
Last refreshed a few seconds ago ..
FUJITSU Enterprise Postgres
© FUJITSU LIMITED 2020
100%

Figure 5-14 WebAdmin: Create instance option

16. Select the configuration type as **Master-standby configuration** and name the instance `inst1p` (see Figure 5-15).



Create instance

* indicates required field(s)

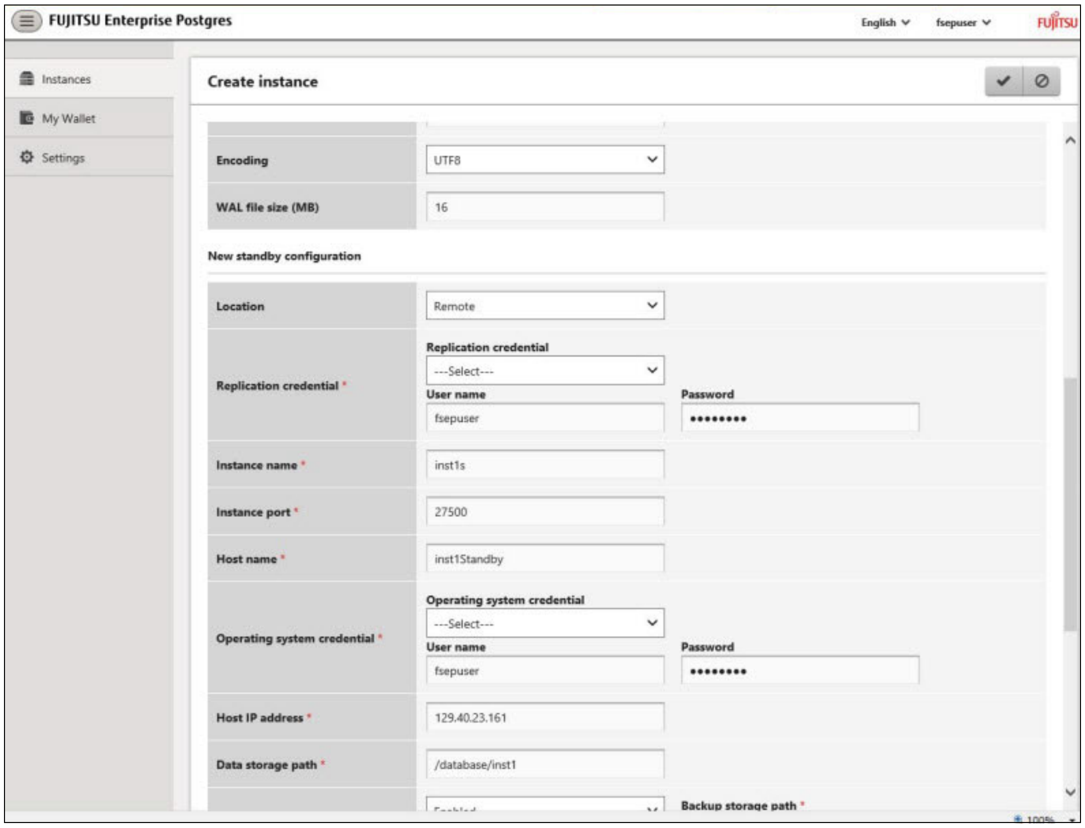
Configuration type	Master-standby configuration
Server product type	FUJITSU Enterprise Postgres 12

New master configuration

Location	Remote
Instance name *	inst1p
Instance port *	27500
Host name *	inst1Primary
Operating system credential *	Operating system credential: ---Select---; User name: fsepuser; Password: ••••••••
Data storage path *	/database/inst1
Backup	Enabled; Backup storage path *: /backup/inst1
Transaction log path	/transaction/inst1
Encoding	UTF8

Figure 5-15   WebAdmin: Create instance page - 1 of 3

17. Create the secondary instance with the instance name `inst1s` and hostname `inst1Standby` (see Figure 5-16).



Create instance

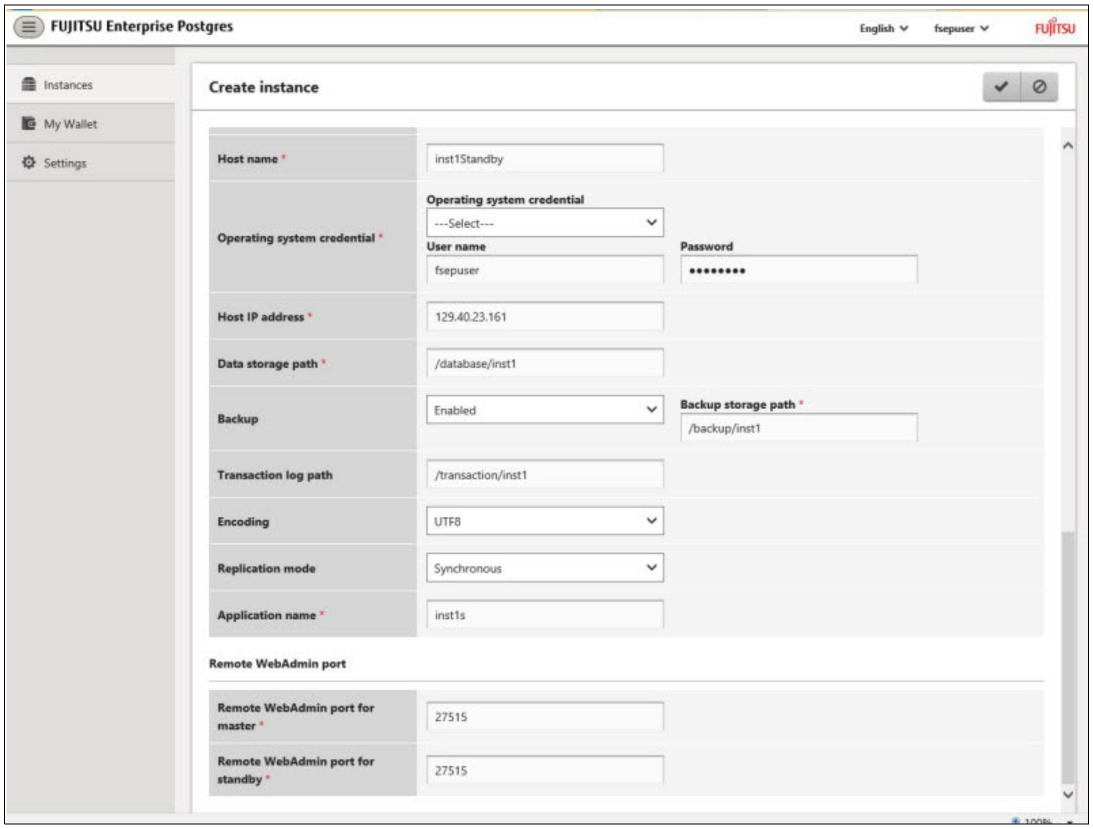
Encoding	UTF8
WAL file size (MB)	16

New standby configuration

Location	Remote	
Replication credential *	Replication credential: ---Select--- ; User name: fsepuser	Password: ••••••••
Instance name *	inst1s	
Instance port *	27500	
Host name *	inst1Standby	
Operating system credential *	Operating system credential: ---Select--- ; User name: fsepuser	Password: ••••••••
Host IP address *	129.40.23.161	
Data storage path *	/database/inst1	
	Enabled	Backup storage path *

Figure 5-16 WebAdmin: Create instance page - 2 of 3

18. Configure the secondary database (standby) instance and click the checkmark (see Figure 5-17).



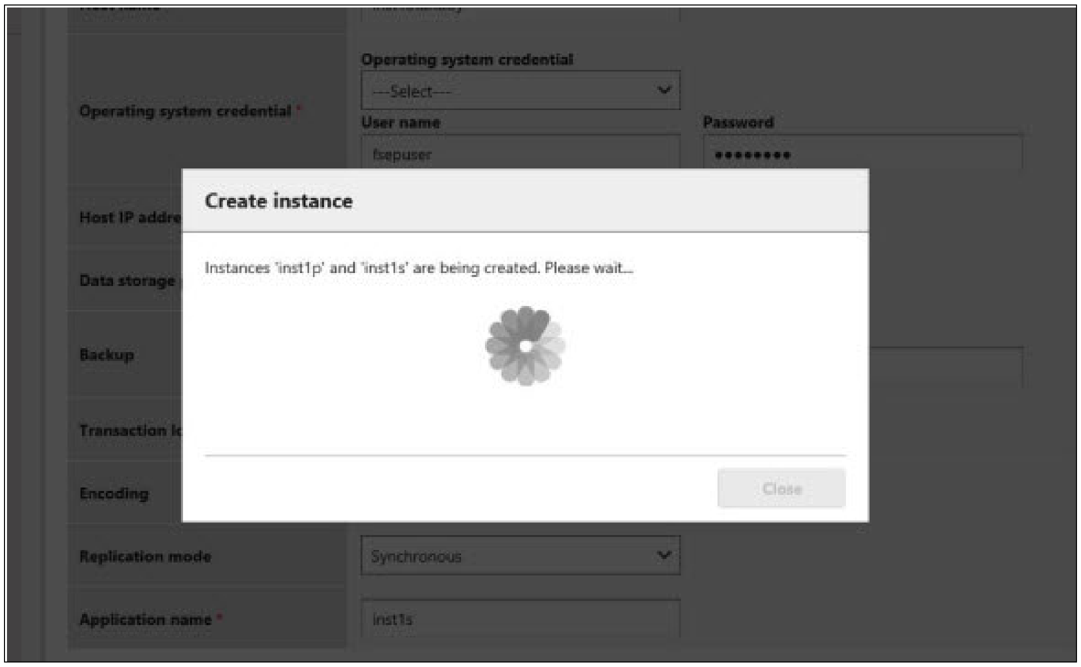
Host name *	inst1Standby
Operating system credential *	Operating system credential: ---Select--- ; User name: fsepuser ; Password: ••••••••
Host IP address *	129.40.23.161
Data storage path *	/database/inst1
Backup	Enabled ; Backup storage path *: /backup/inst1
Transaction log path	/transaction/inst1
Encoding	UTF8
Replication mode	Synchronous
Application name *	inst1s

Remote WebAdmin port

Remote WebAdmin port for master *	27515
Remote WebAdmin port for standby *	27515

Figure 5-17 WebAdmin: Create instance page - 3 of 3

19. You should receive the message that the instances `inst1p` (primary database instance) and `inst1s` (secondary) database are being created, as shown in Figure 5-18.

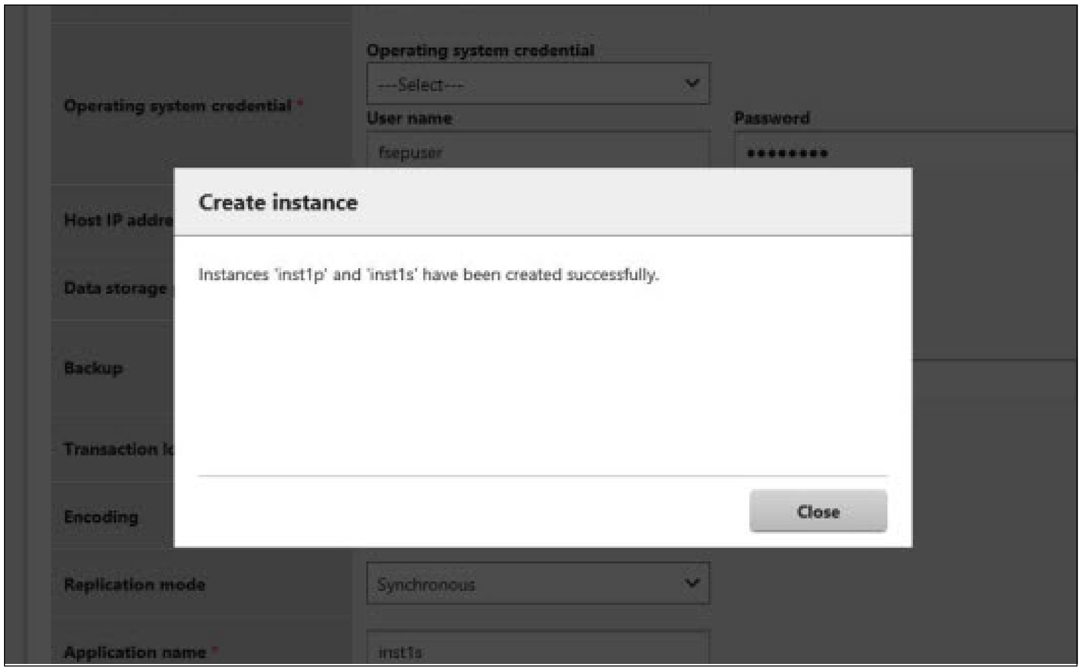


Create instance

Instances 'inst1p' and 'inst1s' are being created. Please wait...

Figure 5-18   WebAdmin: Create instance progress page

20. You receive the message that the instances `inst1p` (primary database instance) and `inst1s` (secondary) database instances were created successfully (see Figure 5-19).

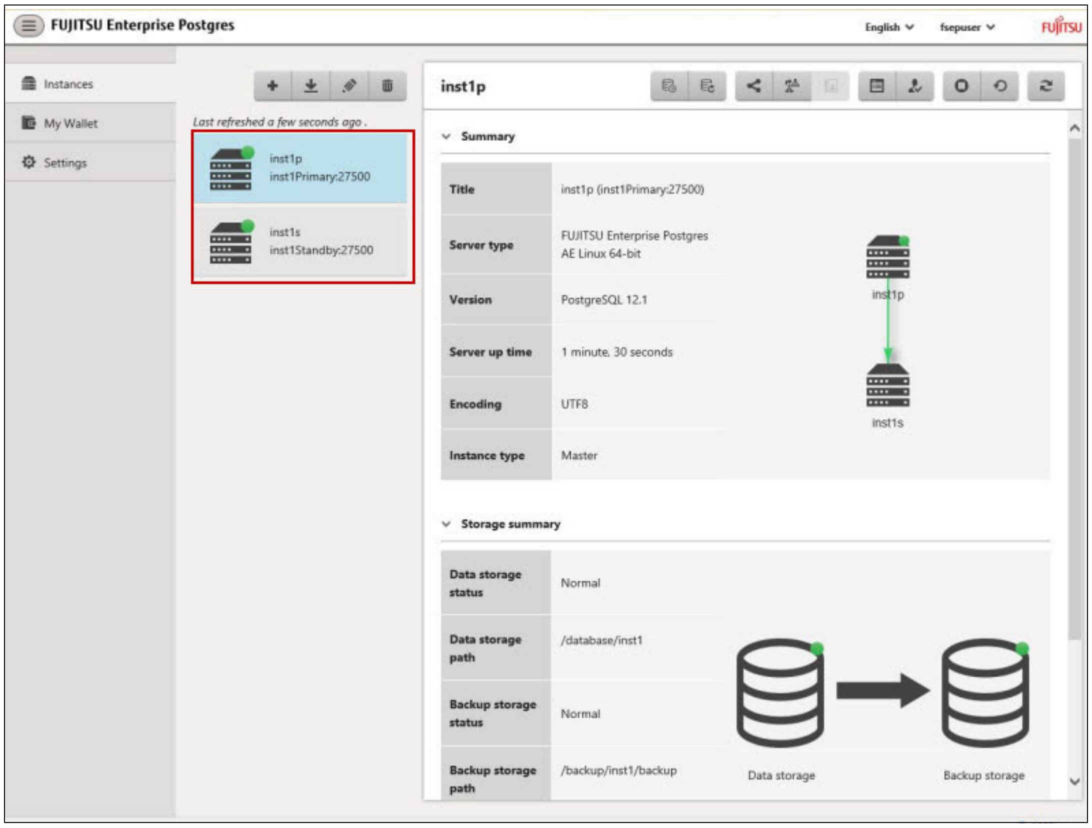


Create instance

Instances 'inst1p' and 'inst1s' have been created successfully.

Figure 5-19   WebAdmin: Create instance success page

21.Under the **Instance** tab, both `inst1p` and `inst1s` are visible as active (see Figure 5-20).


FUJITSU Enterprise Postgres

English fsepuser FUJITSU

Instances
My Wallet
Settings

Last refreshed a few seconds ago .

inst1p
inst1Primary:27500

inst1s
inst1Standby:27500

inst1p

Summary

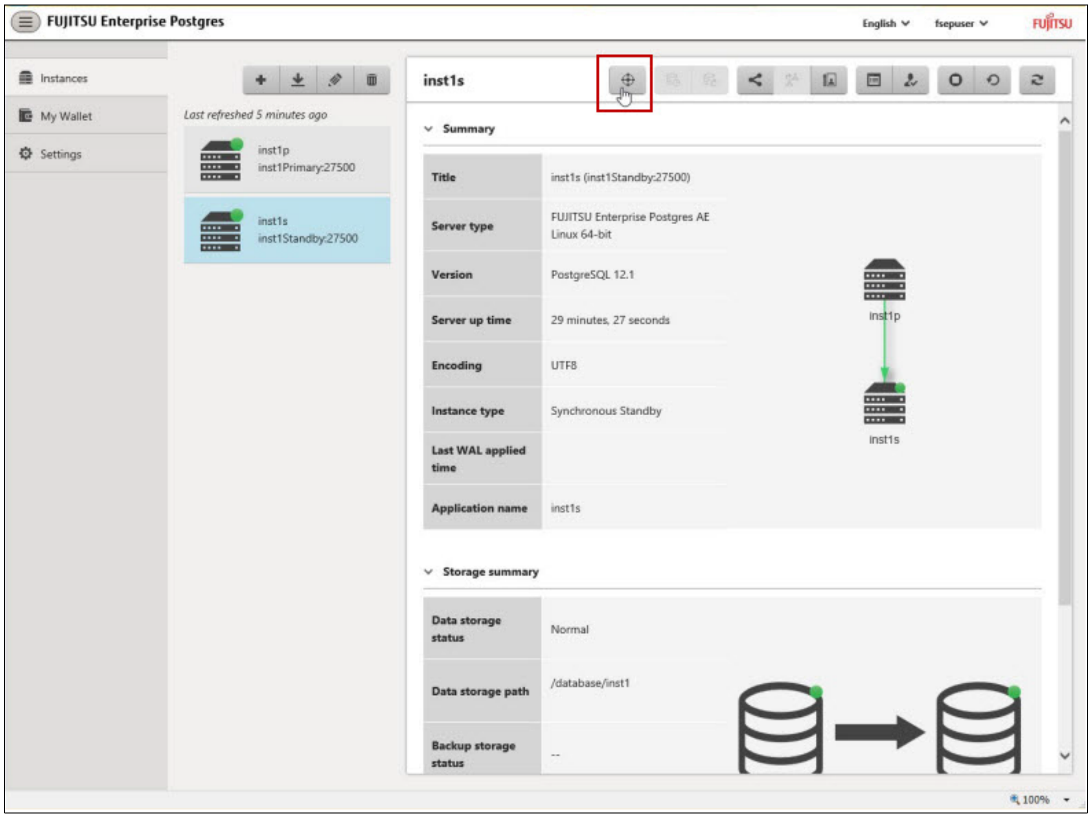
Title	inst1p (inst1Primary:27500)
Server type	FUJITSU Enterprise Postgres AE Linux 64-bit
Version	PostgreSQL 12.1
Server up time	1 minute, 30 seconds
Encoding	UTF8
Instance type	Master

Storage summary

Data storage status	Normal
Data storage path	/database/inst1
Backup storage status	Normal
Backup storage path	/backup/inst1/backup

Figure 5-20 WebAdmin: Instance inst1p primary database instance

22.Configure the MC by clicking the radar symbol (see Figure 5-21).

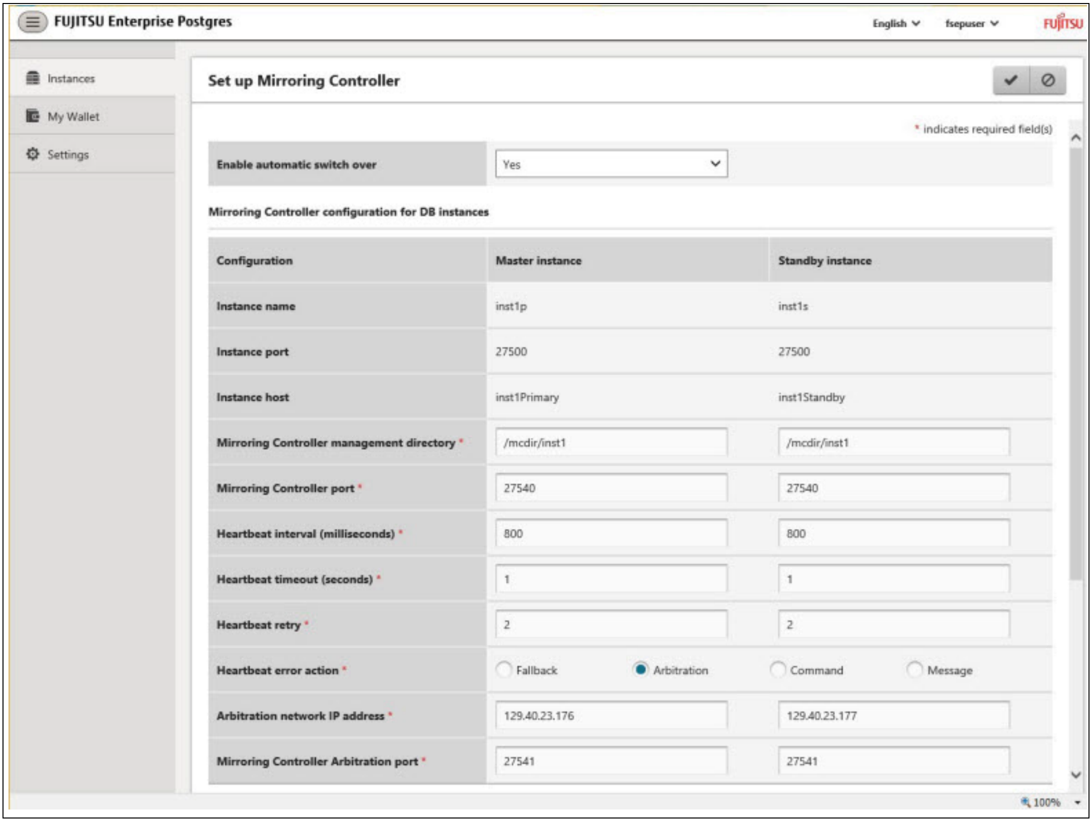


Title	inst1s (inst1Standby:27500)
Server type	FUJITSU Enterprise Postgres AE Linux 64-bit
Version	PostgreSQL 12.1
Server up time	29 minutes, 27 seconds
Encoding	UTF8
Instance type	Synchronous Standby
Last WAL applied time	
Application name	inst1s

Data storage status	Normal
Data storage path	/database/inst1
Backup storage status	--

Figure 5-21   WebAdmin: Instance inst1s standby database instance

23. Figure 5-22 shows the configuration parameters that must be set up for the MC agent.



Set up Mirroring Controller

* indicates required field(s)

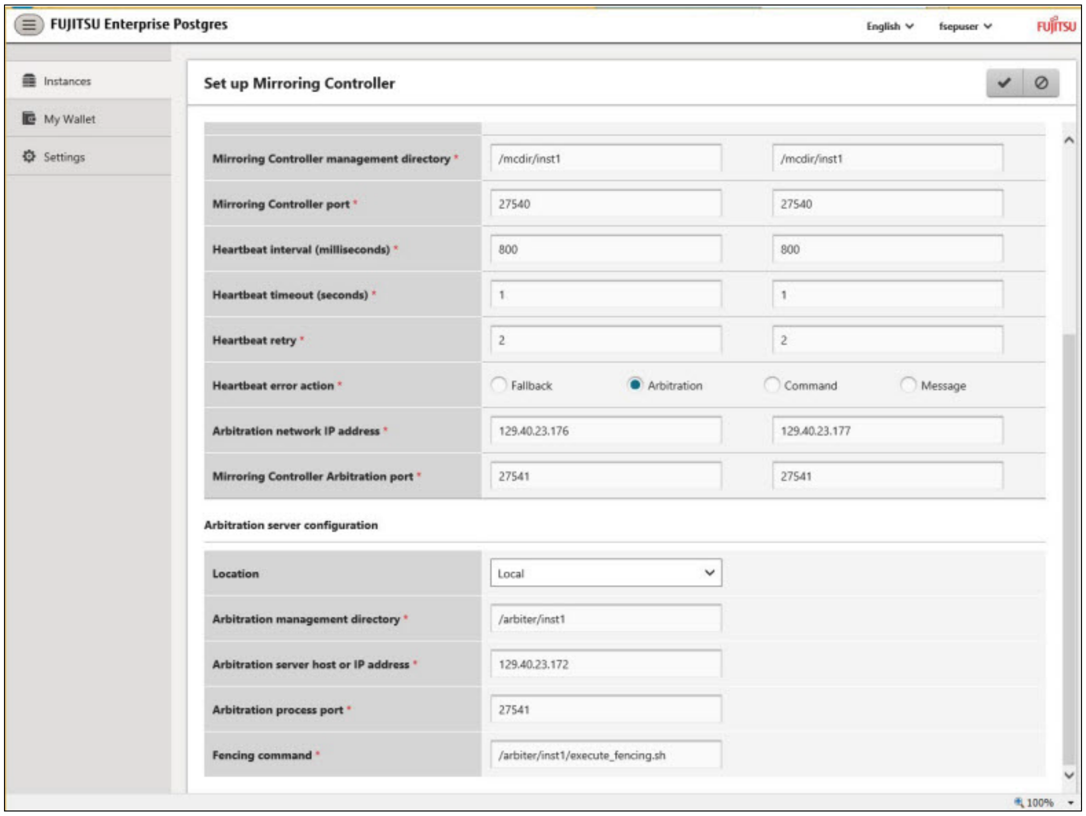
Enable automatic switch over	Yes

Mirroring Controller configuration for DB instances

Configuration	Master instance	Standby instance
Instance name	inst1p	inst1s
Instance port	27500	27500
Instance host	inst1Primary	inst1Standby
Mirroring Controller management directory *	/mcdir/inst1	/mcdir/inst1
Mirroring Controller port *	27540	27540
Heartbeat interval (milliseconds) *	800	800
Heartbeat timeout (seconds) *	1	1
Heartbeat retry *	2	2
Heartbeat error action *	Fallback [✓ Arbitration]	Command Message
Arbitration network IP address *	129.40.23.176	129.40.23.177
Mirroring Controller Arbitration port *	27541	27541

Figure 5-22 WebAdmin: Mirroring Controller setup

24. Figure 5-23 shows more configuration parameters after scrolling down the page for setting up the MC. For more information about the configuration parameters for the Server Assistant, see *FUJITSU Enterprise Postgres Cluster Operations Guide*.



FUJITSU Enterprise Postgres

Set up Mirroring Controller

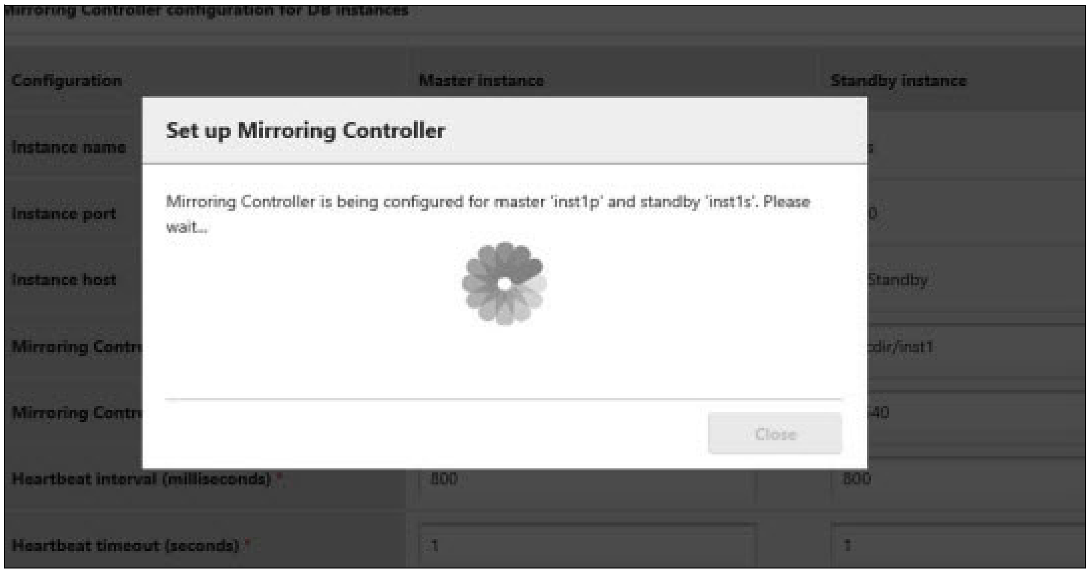
Mirroring Controller management directory *	/mcdir/inst1	/mcdir/inst1
Mirroring Controller port *	27540	27540
Heartbeat interval (milliseconds) *	800	800
Heartbeat timeout (seconds) *	1	1
Heartbeat retry *	2	2
Heartbeat error action *	Fallback [✓ Arbitration] Command Message	
Arbitration network IP address *	129.40.23.176	129.40.23.177
Mirroring Controller Arbitration port *	27541	27541

Arbitration server configuration

Location	Local
Arbitration management directory *	/arbiter/inst1
Arbitration server host or IP address *	129.40.23.172
Arbitration process port *	27541
Fencing command *	/arbiter/inst1/execute_fencing.sh

Figure 5-23 WebAdmin: Mirroring Controller setup

25. Click the checkmark to initiate the configuration of the MC for the primary (or master) database instance `inst1p` and the secondary (or standby) database instance `inst1s`. Figure 5-24 shows the progress.

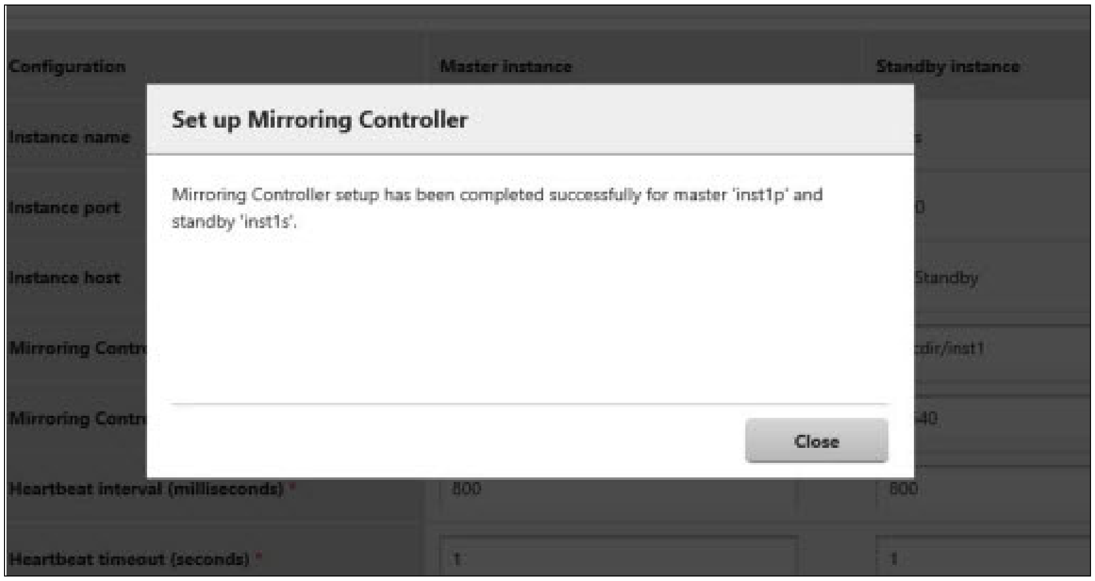


Set up Mirroring Controller

Mirroring Controller is being configured for master 'inst1p' and standby 'inst1s'. Please wait...

Figure 5-24 WebAdmin: Mirroring Controller setup progress

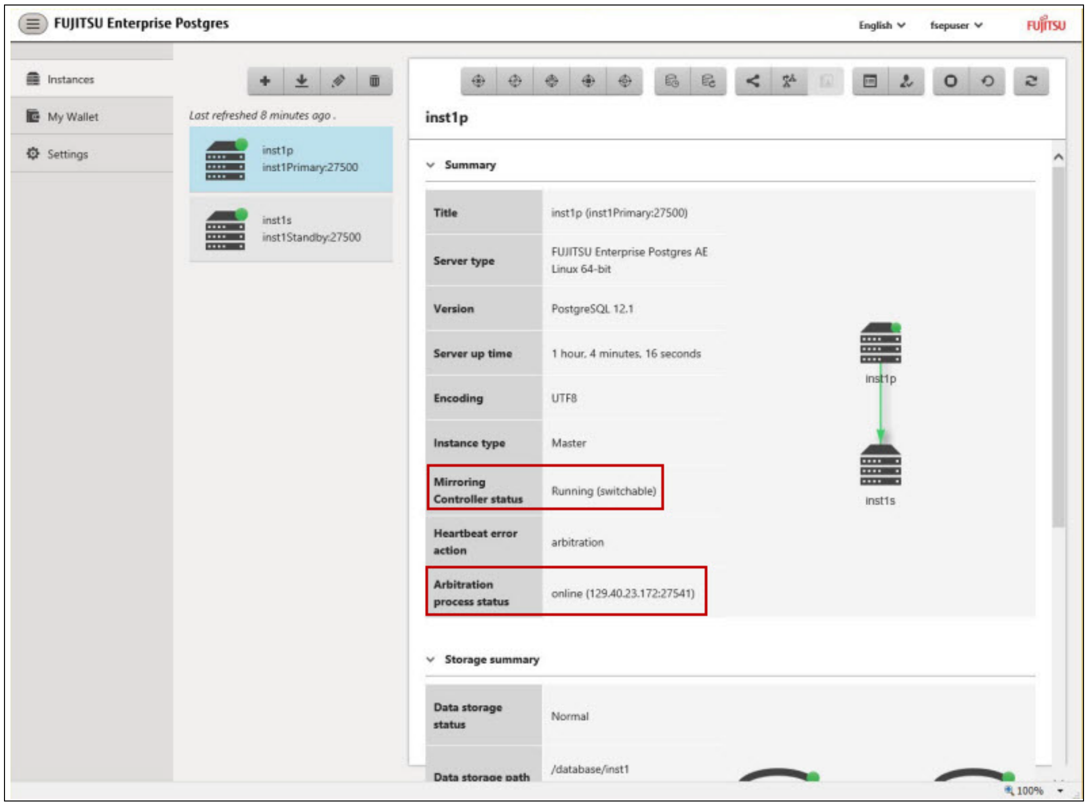
26. When WebAdmin finishes the configuration of the MC on the primary and secondary database (standby) instances, you see the message that is shown in Figure 5-25.


Set up Mirroring Controller

Mirroring Controller setup has been completed successfully for master 'inst1p' and standby 'inst1s'.

Figure 5-25 WebAdmin: Mirroring Controller setup success

27. Click the **Instances** tab and note the status of the FUJITSU Enterprise Postgres HA cluster (Figure 5-26).



Title	inst1p (inst1Primary:27500)
Server type	FUJITSU Enterprise Postgres AE Linux 64-bit
Version	PostgreSQL 12.1
Server up time	1 hour, 4 minutes, 16 seconds
Encoding	UTF8
Instance type	Master
Mirroring Controller status	Running (switchable)
Heartbeat error action	arbitration
Arbitration process status	online (129.40.23.172:27541)

Data storage status	Normal
Data storage path	/database/inst1

Figure 5-26 WebAdmin: Mirroring Controller setup success

28. Check the MC status on the primary and secondary database (standby) instances by running the command that is shown in Example 5-63.

Example 5-63 Checking the Mirroring Controller status

```
$ mc_ctl status --arbiter -M /mcdir/inst1
arbiter_id    host          status
-----------------------------------------
arbiter27541  XXX.XX.XX.179  online
$ mc_ctl status -M /mcdir/inst1
mirroring status
-----------------
switchable
server_id    host_role  host          host_status  db_proc_status  disk_status
-----------------------------------------------------------------------------
inst1s27500  standby    inst1Standby  normal       normal          normal
inst1p27500  primary    inst1Primary  normal       normal          normal
```

29. On the primary database instance, create database objects as shown in Example 5-64 by running the **pgbench** command.

Example 5-64 Configuring pgbench on the primary database instance

```
$ pgbench -i postgres
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...
100000 of 100000 tuples (100%) done (elapsed 0.07 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done.
```

30. Confirm that replication is functional by running the command that is shown in Example 5-65 on the secondary database (standby) instance.

Example 5-65 Confirming that the pgbench tables are replicated to the standby database instance

```
$ psql -h XXX.XX.XX.161 -p 27500 -d postgres -U fsepuser
Password for user fsepuser:
psql (12.1)
Type "help" for help.
postgres=# \d
              List of relations
 Schema |       Name        | Type  |  Owner
--------+-------------------+-------+----------
 public | pgbench_accounts  | table | fsepuser
 public | pgbench_branches  | table | fsepuser
 public | pgbench_history   | table | fsepuser
 public | pgbench_tellers   | table | fsepuser
(Four rows)
```

5.4.6 Simulating automatic failover

In this section, we simulate an automatic failover from the primary database instance to the standby database instance and restore HA by recovering the primary database instance from the secondary database (standby) instance.

We show a use case scenario in which the primary database instance is impacted, which triggers an automatic switchover from the primary database instance to the secondary database (standby) instance. Additionally, we show the steps to recover the primary database instance and reinstate the HA scenario.

For the purposes of this book, we simulated unavailability of the primary database instance by shutting down the Postgres process on the primary database server. Figure 5-27 shows the high-level five-step process through which the standby database instance is upgraded to primary database instance status and starts accepting read/write transactions from the application. After upgrading the standby server to the primary server, we show the steps to bring back the primary database instance and restore HA.

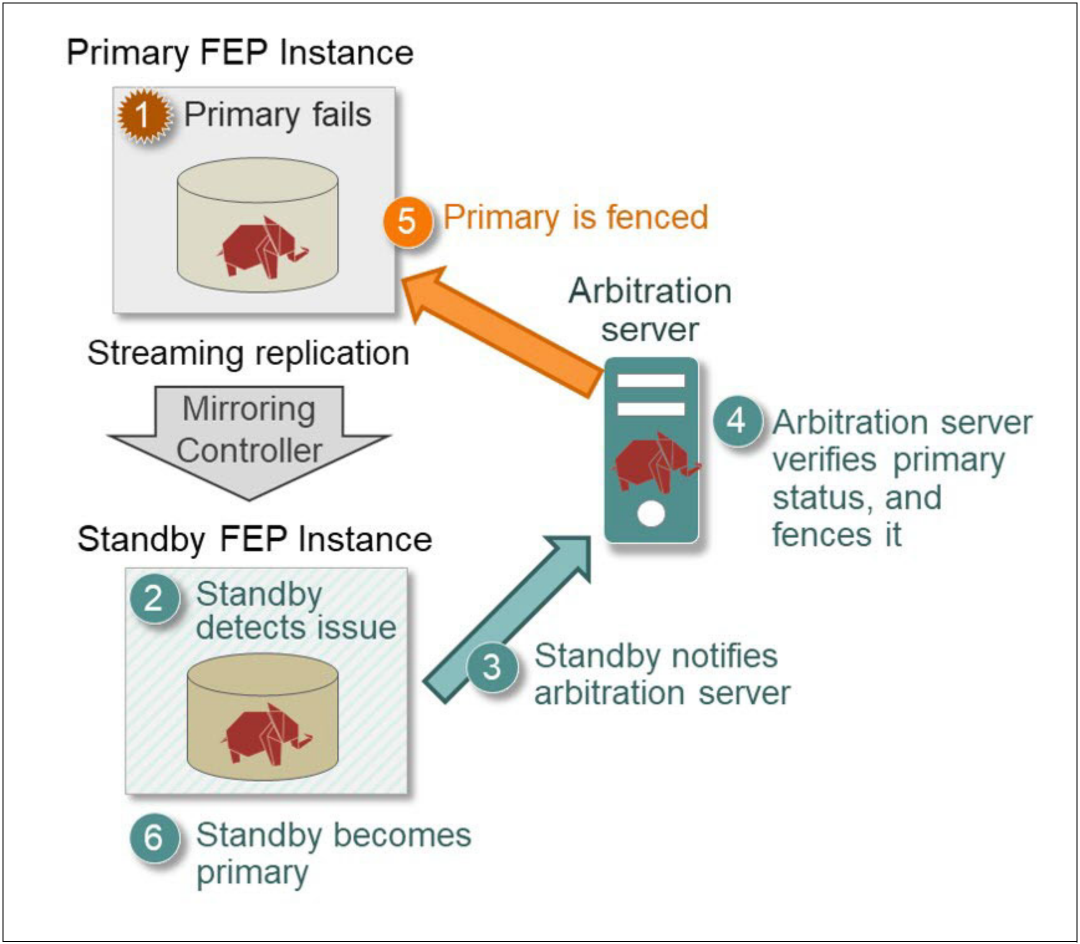


Figure 5-27 Automatic switchover from the primary to standby database instance and upgrading the standby to the primary database instance

For the following steps, we used the HA architecture that was implemented in 5.4.4, "Implementing an active/passive HA architecture when using a command-line interface" on page 182:

1. Shut down the primary database instance by running the command that is shown in Example 5-66.

Example 5-66 Simulating the impact on the primary database

```
$ pg_ctl stop -D /database/inst1 -mi
waiting for server to shut down.... done
server stopped
```

2. After bringing down the primary database instance, the standby database instance MC detects the abnormality and the arbitration server triggers the process to upgrade the secondary (or standby) database instance to an active database instance. The secondary database (standby) instance becomes an active database instance and is now ready to run both read and write SQL queries. To check the status of the secondary database (standby) instance, run the command that is shown in Example 5-67 on the secondary database (standby) instance.

Example 5-67 Automatic failover status check

```
$ mc_ctl status -M /mcdir/inst1
mirroring status
-----------------
switched
server_id  host_role                 host           host_status  db_proc_status        disk_status
----------------------------------------------------------------------------------------------------
inst1p    none(inactivated primary)  XXX.XX.XX.156  normal       abnormal(postmaster)  normal
inst1s    primary                    XXX.XX.XX.161  normal       abnormal(wal_sender)  normal
```

Note: When the primary database instance fails and the fallback to the standby database instance completes, the application that is connected to the primary database instance can connect to the secondary database (standby) instance (the new primary) by retrying the connection. For more information, see Chapter 7, "Application development, compatibility features, and migration" on page 251.

In a real-life scenario, you troubleshoot the root cause of the primary server's unavailability. Until this issue is resolved, the database cluster runs in the stand-alone mode of operation. So, after the issue on the primary database instance is resolved, the next task is to recover the old primary database instance and reinstate it as the new secondary database instance. After the recovery of the primary database instance, the original primary and the secondary database instance are swapped.

In the following steps, we recover the old primary database instance and reinstate it as the new secondary database (standby) instance.

There are two methods to recover the primary database instance:

– If the requirement is to rebuild all the data, then use **`pg_basebackup`**.

– If the requirement is to recover only differential data that is lost during the troubleshooting of the primary database instance, then use **`pg_rewind`**.

In this scenario, we use **`pg_rewind`** to recover the primary database instance to recover the differential data.

After the applications that are connected to the primary are disconnected, complete the following steps.

3. On the primary database instance, stop the MC, as shown in Example 5-68.

Example 5-68 Stopping the Mirroring Controller

```
$ mc_ctl stop -M /mcdir/inst1
stopping Mirroring Controller (MCA00041)
Mirroring Controller stopped  target server:"inst1p" (MCA00042)
```

4. Stop the primary database instance normally by running the command that is shown in Example 5-69.

Example 5-69 Stopping the primary database instance normally

```
$ pg_ctl start -D /database/inst1
waiting for server to start....2020-12-25 01:47:36.283 EST [2161757] LOG:
starting PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (GCC) 8.3.1
20190507 (Red Hat 8.3.1-4), 64-bit
2020-12-25 01:47:36.284 EST [2161757] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2020-12-25 01:47:36.284 EST [2161757] LOG:  listening on IPv6 address "::", port
27500
2020-12-25 01:47:36.285 EST [2161757] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2020-12-25 01:47:36.291 EST [2161757] LOG:  redirecting log output to logging
collector process
2020-12-25 01:47:36.291 EST [2161757] HINT:  Future log output will appear in
directory "log".
 done
server started
$ pg_ctl stop -D /database/inst1
waiting for server to shut down.... done
server stopped
```

5. On the secondary database instance (which is now the new primary database instance), check the status of the completion of recovery by querying the **`pg_is_in_recovery()`** function, as shown in Example 5-70. The output result is Boolean. A `true` value means that the recovery is still in progress, and a `false` value means otherwise. Replace XXX.XX.XX.161 with the IP address of your secondary database instance (which is now the new primary database instance).

Example 5-70 Recovery status check on the new standby database instance

```
$ psql -h XXX.XX.XX.161 -p 27500 -d postgres -U fsepuser -c "select
pg_is_in_recovery()"
 pg_is_in_recovery
-------------------
 f
(One row)
```

6. Run a checkpoint on the secondary database instance (which is now the new primary database instance). Replace XXX.XX.XX.161 with the IP address of your secondary database instance (which is now the new primary database instance) (Example 5-71).

Example 5-71 Checkpoint

```
$ psql -h XXX.XX.XX.161 -p 27500 -d postgres -U fsepuser -c "checkpoint"
CHECKPOINT
```

7. On the old primary database instance, run **pg_rewind** to recover the lost differential data by fetching the data from the secondary database instance (which is the new primary database instance), as shown in Example 5-72.

Example 5-72 Running pg_rewind to recover data from the old primary database instance

```
$ pg_rewind -D /database/inst1 --source-server='user=fsepuser host=XXX.XX.XX.161
port=27500 dbname=postgres'
pg_rewind: servers diverged at WAL location 0/5C93488 on timeline 1
pg_rewind: rewinding from last common checkpoint at 0/5C933D8 on timeline 1
pg_rewind: Done!
```

8. Continuing with the old primary database instance, reconfigure `postgresql.conf` with the new value for the parameter **synchronous_standby_names**, as shown in Example 5-73.

Example 5-73 Reconfiguring synchronous_standby_names on the old primary database instance

```
$ vim /database/inst1/postgresql.conf
synchronous_standby_names='inst1s'
```

9. On the old primary database instance, create `standby.signal` for replication, as shown in Example 5-74.

Example 5-74 Setting up standby.signal

```
$ touch /database/inst1/standby.signal
```

10. On the primary database instance, set up `postgresql.auto.conf`, as shown in Example 5-75.

Example 5-75 Configuring postgresql.auto.conf on the primary database instance

```
$ vim /database/inst1/postgresql.auto.conf
primary_conninfo = 'user=fsepuser passfile=''/home/fsepuser/.pgpass''
host=XXX.XX.XX.161 port=27500 application_name=inst1p sslmode=prefer
sslcompression=0 gssencmode=prefer krbsrvname=postgres target_session_attrs=any'
```

11. To make the old primary database instance the new standby server, run the command that is shown in Example 5-76 to start the MC.

Example 5-76 Starting the Mirroring Controller

```
$ mc_ctl start -M /mcdir/inst1
starting Mirroring Controller (MCA00038)
requesting arbitration server "arbiter" to connect (MCA00149)
succeeded in connection with arbitration server "arbiter" (MCA00151)
enabled failover  target server:"inst1p" (MCA00046)
Mirroring Controller started (MCA00039)
```

12. The old primary database instance is now the new standby server. On the new standby server, check the status and role of the two database instances, as shown in Example 5-77.

Example 5-77 Check the Mirroring Controller status on the new standby database server

```
$ mc_ctl status --arbiter -M /mcdir/inst1
arbiter_id  host           status
---------------------------------------
arbiter     XXX.XX.XX.179  online
```

```
$ mc_ctl status -M /mcdir/inst1
mirroring status
-----------------
switchable
server_id  host_role  host          host_status  db_proc_status  disk_status
-----------------------------------------------------------------------------
inst1p    standby    XXX.XX.XX.156  normal       normal          normal
inst1s    primary    XXX.XX.XX.161  normal       normal          normal
```

Note: Pay attention to the **host_role** *values* for `inst1p` (old primary database instance) and `inst1s` (new primary database instance).

In the architecture that we implemented in 5.4.4, "Implementing an active/passive HA architecture when using a command-line interface" on page 182, there are read-replica and the backup database instances that are connected by using asynchronous streaming replication. Originally, the two cascaded database instances (read-replica and the backup database instances) were connected to the old secondaryf database (standby) instances, which are now the new primary database instance. So, to revert to the original architecture of connecting cascaded database instances to the secondary database (standby) instance, we must change the replication source on the cascaded database instances pointing to the new secondary database (standby) instance.

To do this task, run the following steps first on the read-replica database instance and then on the backup database instance.

13.On the cascaded read-replica database, update the host IP address to redirect to the new secondary database (standby) instance, as shown in Example 5-78.

Example 5-78 Updating the IP address on the read-replica server

```
$ vim /database/inst1/postgresql.auto.conf
primary_conninfo = 'user=fsepuser passfile=''/home/fsepuser/.pgpass''
host=XXX.XX.XX.156 port=27500 application_name=inst1cs sslmode=prefer
sslcompression=0 gssencmode=prefer krbsrvname=postgres target_session_attrs=any'
```

14.On the cascaded backup database instance, update the host IP address to redirect to the new secondary database (standby) instance, as shown in Example 5-79.

Example 5-79 Updating the IP address on the backup database instance

```
$ vim /database/inst1/postgresql.auto.conf
primary_conninfo = 'user=fsepuser passfile=''/home/fsepuser/.pgpass''
host=XXX.XX.XX.156 port=27500 application_name=inst1bk sslmode=prefer
sslcompression=0 gssencmode=prefer krbsrvname=postgres target_session_attrs=any'
```

15.Restart both the cascaded database instances, that is, the read-replica and the backup database instances, as shown in Example 5-80.

Note: Run the **pg_ctl restart -D /database/inst1** command on the read-replica database instance and backup database instance.

Example 5-80 Restarting the read-replica and backup database instances

```
$ pg_ctl restart -D /database/inst1
waiting for server to shut down.... done
server stopped
```

```
waiting for server to start....2020-12-25 02:22:31.311 EST [340730] LOG:  starting
PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (GCC) 8.3.1 20190507 (Red
Hat 8.3.1-4), 64-bit
2020-12-25 02:22:31.311 EST [340730] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2020-12-25 02:22:31.311 EST [340730] LOG:  listening on IPv6 address "::", port
27500
2020-12-25 02:22:31.312 EST [340730] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2020-12-25 02:22:31.318 EST [340730] LOG:  redirecting log output to logging
collector process
2020-12-25 02:22:31.318 EST [340730] HINT:  Future log output will appear in
directory "log".
 done
server started
```

We have completed re-creating the HA implementation of a FUJITSU Enterprise Postgres cluster after the event of a primary database instance becoming unavailable.

At this stage, the old primary database instance and old secondary (or standby) database instances are swapped with each other. Also, the cascaded database instances are now connected to the old primary database instance because it became the new secondary (or standby) database instance. On occasion, for various infrastructure and application management reasons, you might decide to reinstate the “before-primary-database-failure” state of the FUJITSU Enterprise Postgres High Availability cluster. To implement this state, swapping the new primary database instance to the old secondary database (standby) instance and swapping the new secondary database (standby) instance to the old primary database instance is required. Furthermore, the replication source for the cascaded database instances must be swapped to the old secondary (or standby) database instance.

To reinstate the original HA architecture, complete the following steps:

1. On the new primary database instance, run the switching MC command, as shown in Example 5-81.

Example 5-81 Switching the Mirroring Controller

```
$ mc_ctl switch -M /mcdir/inst1
starting to switch over (MCA00071)
switch over completed.switched from inst1s to inst1p (MCA00022)
```

Note: In the command that is shown in Example 5-81, the new secondary database instance is enabled to accept both read and write transactions, and the new primary database instance accepts only the read transactions.

2. On the new secondary database (standby) instance, stop the MC, as shown in Example 5-82.

Example 5-82 Stopping the Mirroring Controller on a new standby database instance

```
$ mc_ctl stop -M /mcdir/inst1
stopping Mirroring Controller (MCA00041)
Mirroring Controller stopped  target server:"inst1s" (MCA00042)
```

3. Repeat steps 4 on page 219 to 14 on page 221. While doing so, ensure that you are using the correct IP addresses and instance names.

Although we described only a one-use case scenario of database instance failure, there might be other scenarios where the HA database cluster might become unstable. The following list includes only some of the possible scenarios that you should consider when testing HA implementations:

- Changes in operation when the standby database server is stopped.
- Changes in operation when an error occurs in the admin network that is used for exclusive access to all the database instances and VMs.
- Changes in operation when there is planned upgrade activity on the database servers.
- Changes in operation when building the primary database instance and the secondary database (standby) instance on the same LPAR or VM.

For more information about the operations of these scenarios, see the FUJITSU Enterprise Postgres Advanced Edition documentation.

5.5 IBM LinuxONE with IBM z/VM live guest relocation for FUJITSU Enterprise Postgres Database

This section describes how IBM LinuxONE with IBM z/VM live guest relocation (LGR) can be used as an extra layer to provide HA for FUJITSU Enterprise Postgres Database. We also describe a scenario of relocating a FUJITSU Enterprise Postgres Database between two members of a z/VM single-system image (SSI) cluster.

A z/VM SSI cluster is a multi-system environment on which the z/VM systems can be managed as a single resource pool and guests can be moved from one system to another while they are running. Each SSI member is a z/VM LPAR connected through channel-to-channel (CTC) connections.

A z/VM SSI cluster consists of up to four z/VM systems in an Inter-System Facility for Communications (ISFC) collection. Each z/VM system is a member of the SSI cluster. The cluster is self-managed by the z/VM control program (CP) by using ISFC messages that flow across CTC devices between the members. All members can access shared direct-access storage device (DASD) volumes, the same Ethernet LAN segments, and the same storage area networks (SANs).

Figure 5-28 shows a four-member SSI cluster.

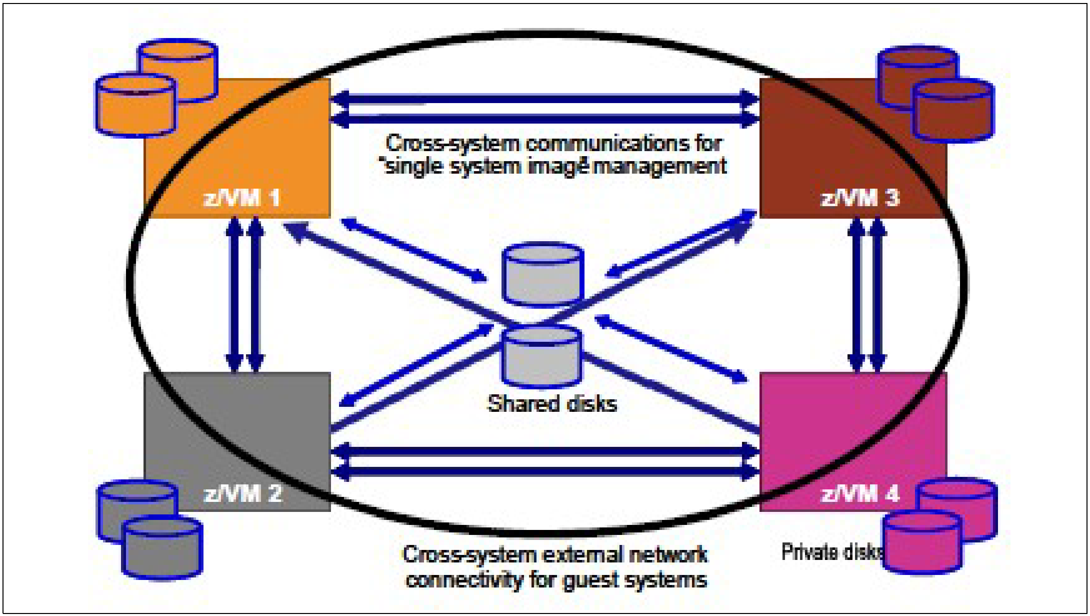

Cross-system communications for "single system image" management
z/VM 1
z/VM 3
Shared disks
z/VM 2
z/VM 4
Cross-system external network connectivity for guest systems
Private disks

Figure 5-28 Four-member SSI cluster

For more information about the z/VM SSI cluster and LGR, see z/VM Single System Image Overview.

5.5.1 Single-system image clustered hypervisor with live guest relocation

Some of the features of the z/VM SSI feature and LGR include:

- ▶ Included in base z/VM 7.1 (previously an optional priced feature).
- ▶ Connect up to four z/VM systems as members of an SSI cluster.
- ▶ Cluster members can be run on the same or different IBM Z servers.
- ▶ Simplifies management of a multi-z/VM environment:
  - – Single user directory.
  - – Cluster management from any member.
  - – Apply maintenance to all members in the cluster from one location.
  - – Issue commands from one member to operate on another.
- ▶ Built-in cross-member capabilities.
- ▶ Resource coordination and protection of network and disks.
- ▶ Allows LGR of running Linux guests.

5.5.2 LGR

With the IBM z/VM SSI, a running Linux on IBM System z® VM can be relocated from one member system to any other, which is a process that is known as LGR. LGR occurs without disruption to the business. It provides application continuity across planned z/VM and hardware outages and flexible workload balancing that allows work to be moved to available system resources.

You might need to relocate a running virtual server for the following reasons:

- Maintaining hardware or software
- Fixing performance problems
- Rebalancing workload

Relocating virtual servers can be useful for load balancing and for moving workload off a physical server or member system that requires maintenance. After maintenance is applied to a member, guests can be relocated back to that member, which allows you to maintain z/VM and keep your Linux on IBM Z virtual servers available.

At the time of writing, the certification of FUJITSU Enterprise Postgres under LGR is in progress. Before implementing this solution in a production environment, contact IBM for the latest guidelines.

5.5.3 Lab environment

Our lab environment has a z/VM SSI cluster with four members. We use two members to demonstrate the process of relocating FUJITSU Enterprise Postgres Database between these members by using the LGR function.

5.5.4 Overview of the z/VM SSI cluster

The SSI cluster is named RDBKSSI1. It is composed of four members: RDBKZVM1, RDBKZVM2, RDBKZVM3, and RDBKZVM4, as shown in Figure 5-29.

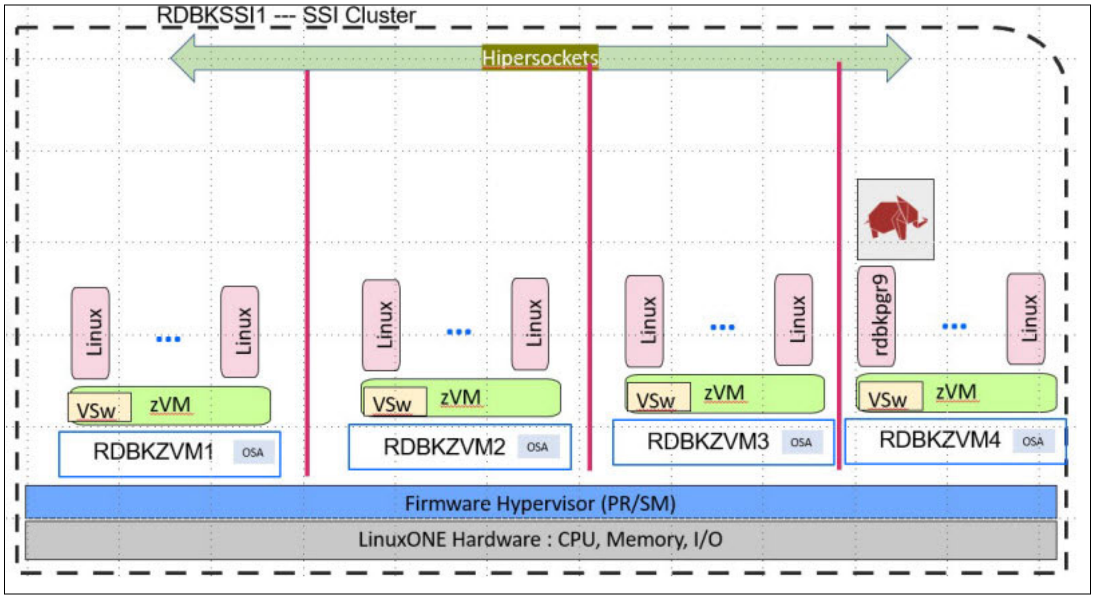


Figure 5-29 SSI cluster of our lab environment

We can use the CP command **query ssi** to display the information about the SSI cluster, as shown in Example 5-83.

Example 5-83 Displaying information about the SSI cluster

```
==> query ssi
SSI Name: RDBKSSI1
SSI Mode: Stable
```

```
Cross-System Timeouts: Enabled
SSI Persistent Data Record (PDR) device: RDCOM1 on 9830
SLOT SYSTEMID STATE     PDR HEARTBEAT       RECEIVED HEARTBEAT
   1 RDBKZVM1 Joined    01/12/21   14:19:28 01/12/21   14:19:28
   2 RDBKZVM2 Joined    01/12/21   14:19:23 01/12/21   14:19:23
   3 RDBKZVM3 Joined    01/12/21   14:19:11 01/12/21   14:19:11
   4 RDBKZVM4 Joined    01/12/21   14:19:11 01/12/21   14:19:11
Ready; T=0.01/0.01 14:19:32
```

5.5.5 Simulating a database workload

During our scenario, we simulated the client workload on the FUJITSU Enterprise Postgres Database by using a TCP-C benchmark. We used the HammerDB benchmark tool to simulate the application connection. For more information, see HammerDB.

5.5.6 Relocating an active FUJITSU Enterprise Postgres Database by using LGR

In this scenario, we describe the steps that are used to relocate an active FUJITSU Enterprise Postgres Database that is running in a Linux on IBM Z guest between members of a z/VM SSI cluster.

To build this scenario, we installed and set up a stand-alone FUJITSU Enterprise Postgres Database.

5.5.7 Setup information

We installed FUJITSU Enterprise Postgres Database on a Red Hat Enterprise Linux 8.0 guest, rdbkpgr9, running on one of the SSI cluster members, RDBKZVM4.

We can use the CP command **`query rdbkpgr9 at all`** to display information about which LPAR the Linux guest is running on, as shown in Example 5-84.

Example 5-84   Displaying information about the LPAR

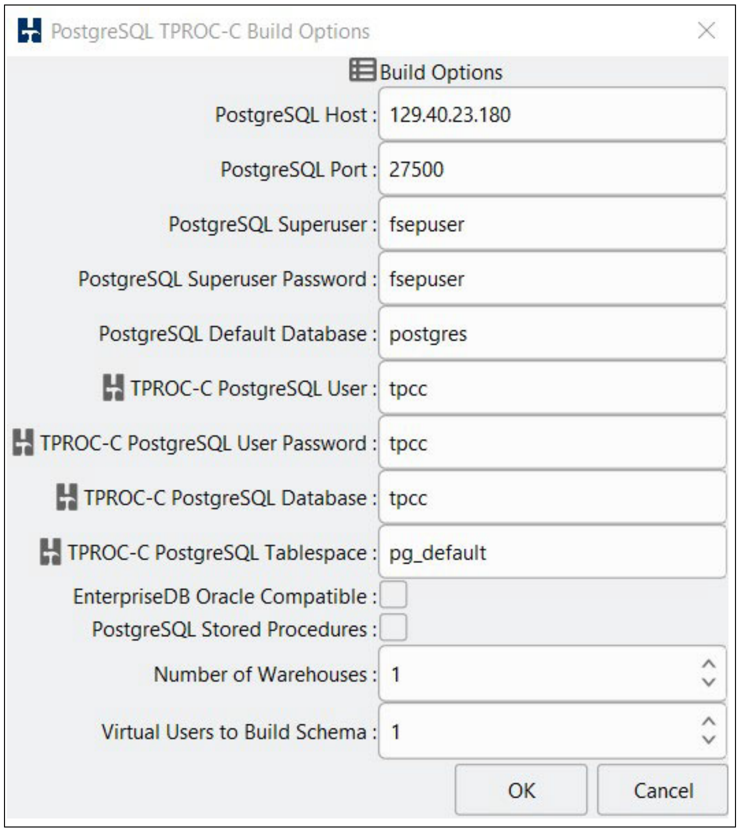
```
Ready; T=0.01/0.01 14:19:32
==> q rdbkpgr9 at all
RDBKZVM4 : RDBKPGR9 - DSC
Ready; T=0.01/0.01 14:33:30
```

We will relocate the rdbkpgr9 Linux guest from RDBKZVM4 to another LPAR, RDBKZVM1.

5.5.8 Simulating the client workload

The client workload is simulated by using the HammerDB benchmark tool from a Windows client that is on the same network.

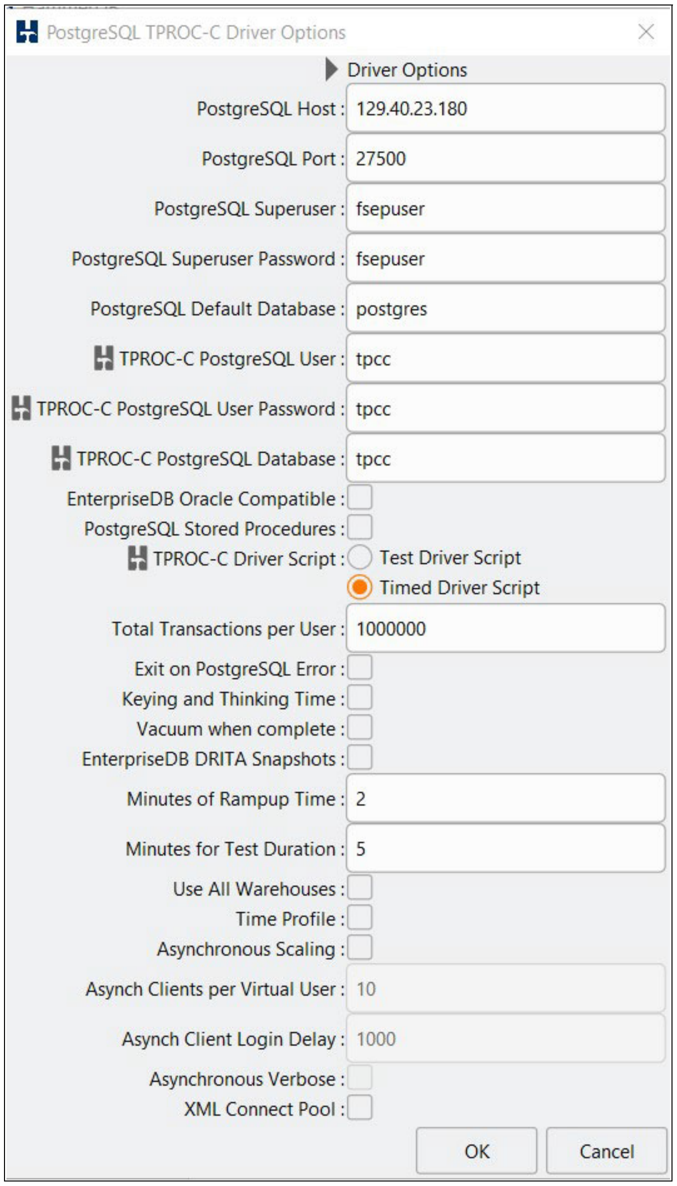
In our example, Figure 5-30 shows the options that we used for schema building for the benchmark.



PostgreSQL Host :	129.40.23.180
PostgreSQL Port :	27500
PostgreSQL Superuser :	fsepuser
PostgreSQL Superuser Password :	fsepuser
PostgreSQL Default Database :	postgres
TPROC-C PostgreSQL User :	tpcc
TPROC-C PostgreSQL User Password :	tpcc
TPROC-C PostgreSQL Database :	tpcc
TPROC-C PostgreSQL Tablespace :	pg_default
EnterpriseDB Oracle Compatible :	☐
PostgreSQL Stored Procedures :	☐
Number of Warehouses :	1
Virtual Users to Build Schema :	1

Figure 5-30 Build options

Figure 5-31 show the options that we used to create 10 virtual users.



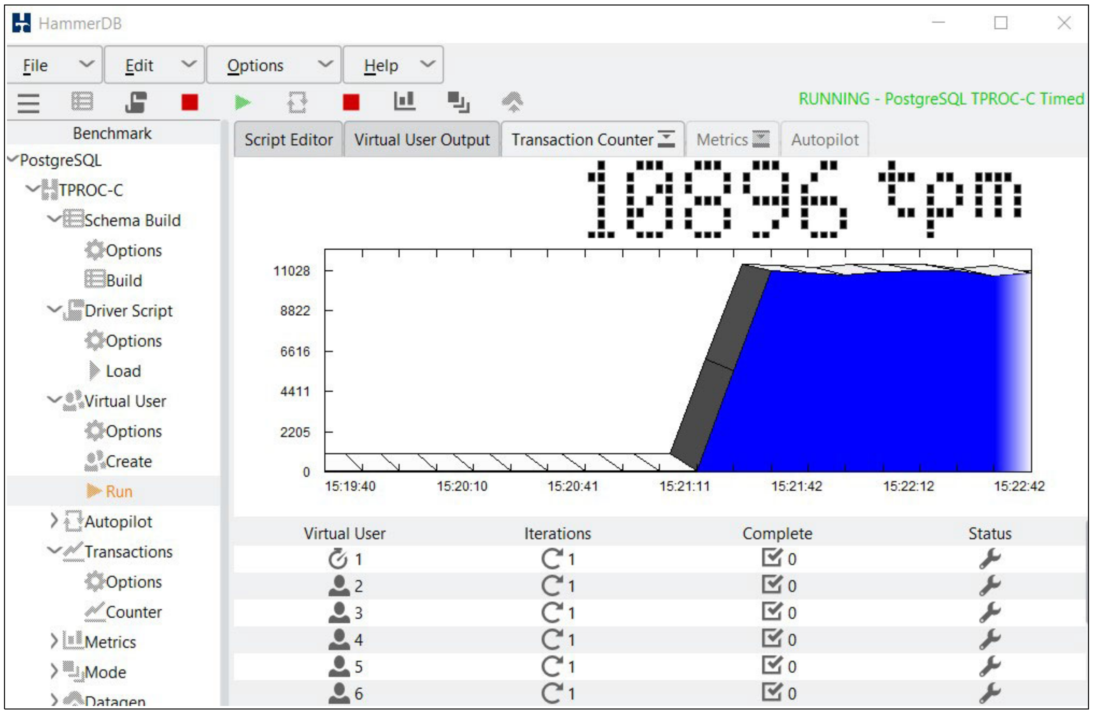
PostgreSQL TPROC-C Driver Options

Driver Options

PostgreSQL Host :	129.40.23.180
PostgreSQL Port :	27500
PostgreSQL Superuser :	fsepuser
PostgreSQL Superuser Password :	fsepuser
PostgreSQL Default Database :	postgres
TPROC-C PostgreSQL User :	tpcc
TPROC-C PostgreSQL User Password :	tpcc
TPROC-C PostgreSQL Database :	tpcc
EnterpriseDB Oracle Compatible :	☐
PostgreSQL Stored Procedures :	☐
TPROC-C Driver Script :	○ Test Driver Script / ● Timed Driver Script
Total Transactions per User :	1000000
Exit on PostgreSQL Error :	☐
Keying and Thinking Time :	☐
Vacuum when complete :	☐
EnterpriseDB DRITA Snapshots :	☐
Minutes of Rampup Time :	2
Minutes for Test Duration :	5
Use All Warehouses :	☐
Time Profile :	☐
Asynchronous Scaling :	☐
Asynch Clients per Virtual User :	10
Asynch Client Login Delay :	1000
Asynchronous Verbose :	☐
XML Connect Pool :	☐

Figure 5-31 Creating 10 virtual users

When we started the TPC-C benchmark, we saw the transactions in the output that is shown in Figure 5-32.



Virtual User	Iterations	Complete	Status
1	1	0	
2	1	0	
3	1	0	
4	1	0	
5	1	0	
6	1	0	

Figure 5-32 Starting the benchmark

We also verified the connectivity between the HammerDB tool and the FUJITSU Enterprise Postgres Database by checking that the TCP sockets were open between HammerDB and the database guest `rdbkpgr9`, as shown in Example 5-85.

Example 5-85 Verifying the connectivity

```
[fsepuser@rdbkpgr9 man]$ netstat -a | grep 9.160
tcp        0      0 rdbkpgr9.pbm.ihos:27500 9.160.13.158:57525      ESTABLISHED
tcp        0     74 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64897      ESTABLISHED
tcp        0      0 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64146      ESTABLISHED
tcp        0      0 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64892      ESTABLISHED
tcp        0     74 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64899      ESTABLISHED
tcp        0     72 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64909      ESTABLISHED
tcp        0     73 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64902      ESTABLISHED
tcp        0     72 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64894      ESTABLISHED
tcp        0     74 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64911      ESTABLISHED
tcp        0     73 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64904      ESTABLISHED
tcp        0     73 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64916      ESTABLISHED
tcp        0      0 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64908      TIME_WAIT
tcp        0     64 rdbkpgr9.pbm.ihost.:ssh 9.160.13.158:50459      ESTABLISHED
tcp        0     74 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64913      ESTABLISHED
tcp        0     74 rdbkpgr9.pbm.ihos:27500 9.160.13.158:64907      ESTABLISHED
[fsepuser@rdbkpgr9 man]$
```

5.5.9 Relocating the FUJITSU Enterprise Postgres Database Guest

To relocate the FUJITSU Enterprise Postgres Database Guest in our example, we ran the z/VM commands from a class A user ID (for example, `MAINT`) of RDBKZVM4.

To perform LGR, complete the following steps:

1. Ensure that RDBKPGR9 is running in RDBKZVM4, as shown in Example 5-86.

Example 5-86 Checking where the guest is running

```
==> QUERY RDBKPGR9 AT ALL
RDBKZVM4 : RDBKPGR9 - DSC
```

2. Test whether the guest is eligible for relocation to RDBKZVM1, as shown in Example 5-87.

Example 5-87 Testing the eligibility

```
==> VMRELOCATE TEST RDBKPGR9 TO RDBKZVM1
User RDBKPGR9 is eligible for relocation to RDBKZVM1
Ready; T=0.01/0.01 15:33:55
```

3. Perform the effective relocation to RDBKZVM1, as shown in Example 5-88.

Example 5-88 Performing relocation

```
==> VMRELOCATE MOVE RDBKPGR9 TO RDBKZVM1
Relocation of RDBKPGR9 from RDBKZVM4 to RDBKZVM1 started
User RDBKPGR9 has been relocated from RDBKZVM4 to RDBKZVM1
Ready; T=0.01/0.01 16:24:10
```

4. Verify that the relocation was successful, as shown in Example 5-89.

Example 5-89 Verifying the relocation

```
==> QUERY RDBKPGR9 AT ALL
RDBKZVM1 : RDBKPGR9 - DSC
Ready; T=0.01/0.01 16:24:57
```

5.5.10 Validating the relocation

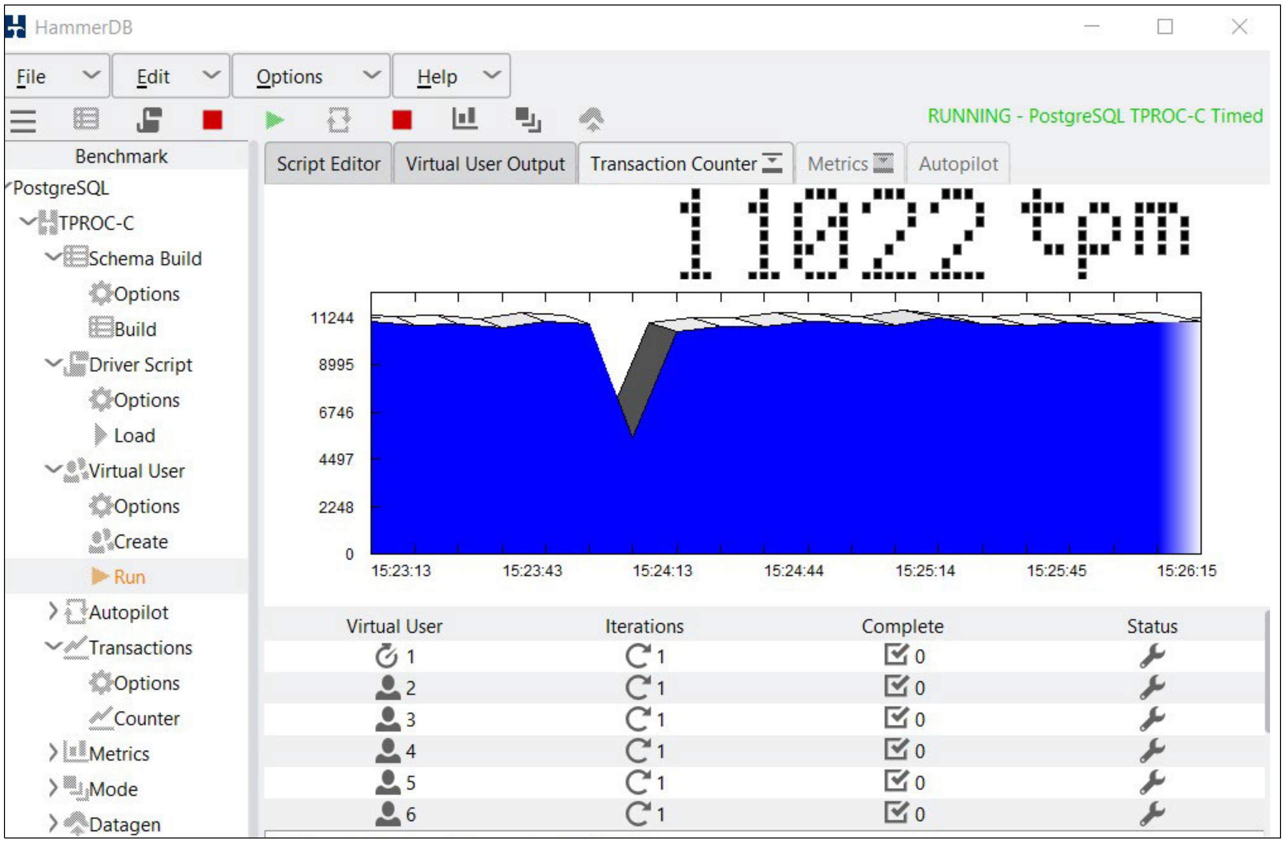
You can see the quiesce time of the guest by running a **ping** command to the gateway each second during the relocation process. In our test, RDBKPGR9 is freezing for about 2 seconds (16:25:25 - 16:25:27), as shown in Example 5-90.

Example 5-90 Ping test

```
# ping XXX.XX.XX.180 | awk '/64/ {"date" | getline date ; print $0, "\t\t" date ;
close("date")}'
64 bytes from xxx.xx.xx.180: icmp_seq=101 ttl=64 time=0.261 msTues Jan 12 16:25:19 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=102 ttl=64 time=0.276 msTues Jan 12 16:25:20 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=103 ttl=64 time=0.257 msTues Jan 12 16:25:21 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=104 ttl=64 time=0.290 msTues Jan 12 16:25:22 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=105 ttl=64 time=0.282 msTues Jan 12 16:25:23 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=106 ttl=64 time=0.287 msTues Jan 12 16:25:24 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=107 ttl=64 time=0.288 msTues Jan 12 16:25:25 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=108 ttl=64 time=1.54  msTues Jan 12 16:25:26 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=109 ttl=64 time=0.304 msTues Jan 12 16:25:27 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=110 ttl=64 time=0.231 msTues Jan 12 16:25:28 EST 2021
```

```
64 bytes from xxx.xx.xx.180: icmp_seq=111 ttl=64 time=1.44  msTues Jan 12 16:25:29 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=112 ttl=64 time=0.241 msTues Jan 12 16:25:30 EST 2021
64 bytes from xxx.xx.xx.180: icmp_seq=113 ttl=64 time=0.263 msTues Jan 12 16:25:31 EST 2021
```

In the HammerDB GUI, we observe that during the quiesce time, the transactions froze (Figure 5-33) in the Transaction Counter graph. However, all the users remained connected to the database.



Virtual User	Iterations	Complete	Status
1	1	0	
2	1	0	
3	1	0	
4	1	0	
5	1	0	
6	1	0	

Figure 5-33   Transactions that are frozen but all users remain connected

6

Connection pooling and load balancing with Pgpool-II

In this chapter, we continue to build the FUJITSU Enterprise Postgres high availability (HA) architecture by adding connection pooling and the load-balancing software Pgpool-II, which comes with FUJITSU Enterprise Postgres.

Pgpool-II is a feature-rich software that provides load distribution, load balancing, and a connection pooling facility. Pgpool-II enables load distribution by sending the WRITE queries to the primary database instance while load balancing the READ-ONLY queries across the active-hot-standby database instances.

Pgpool-II is open source software that is packaged with FUJITSU Enterprise Postgres. It is supported by bug fixes from FUJITSU. The tool has strong community development support with rich documentation.

For more information about the latest developments in Pgpool-II, see What is Pgpool-II?

This chapter includes the following topics:

- Connection pooling
- Pgpool-II
- Installing and configuring Pgpool-II for FUJITSU Enterprise Postgres

6.1 Connection pooling

PostgreSQL is a process-based database software. The postgres (postmaster deprecated) process of PostgreSQL initiates new processes for every connection to the PostgreSQL database. Every time a connection to the database is initiated, this process consumes 2 - 3 MB of system memory for every new connection that is created. Furthermore, for each connection, the operating system (OS) must allocate processor time and schedule it for execution. For applications, which might have thousands of connections, the database server might be overwhelmed because every connection takes up memory and processor cycles for SQL processing, file handling, network traffic management, and process management. The situation is even worse with idle connections. This situation eventually exhausts the computing resources and adversely impacts the database performance because there will be increased context switching, which causes timeouts.

Note: For more information about how connections are established in PostgreSQL, see How Connections Are Established.

In modern applications, developers do not want to hold a database connection for long when other operations are in progress. In fact, the goal of the developers is to keep the transaction duration as short as possible by using techniques that allows them to delay opening and closing the database connection while possible. Although this idea is a programming best practice, it often works against the PostgreSQL architecture because forking server processes and back-end processes that frequently open and close short-duration transactions impact the database. As the number of connections increase, resource contention also increases, which leads to latency in transactions and application unresponsiveness in a worst-case scenario.

This problem is resolved by introducing connection pooling software between the application and the database server whose main role is to make sure that the connections are not rejected by the database or cause excessive overhead on the database for forking numerous server and back-end processes.

Figure 6-1 shows a high-level setup where the number of connections is maintained by the connection pooling software while keeping only a few connections active between the database server and the connection pool. Although the connection pooling software can be installed on the database server, the best practice is to host it on an independent server or virtual machine (VM).

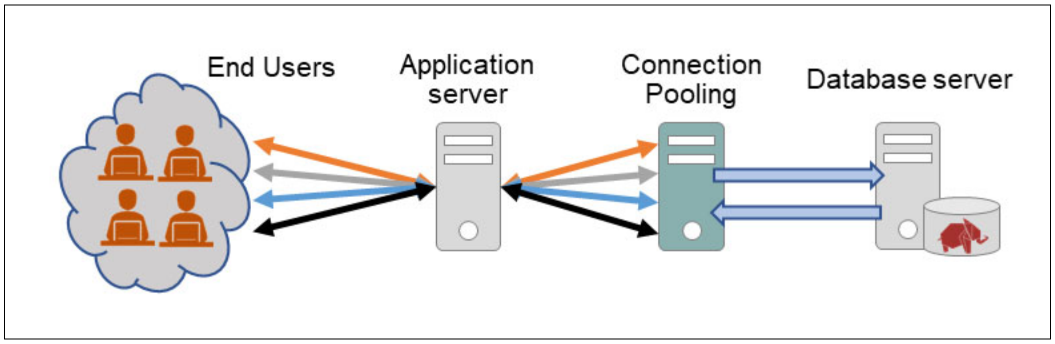


Figure 6-1 Connection pooling between an application server and a database server

6.2 Pgpool-II

There are two open source community developed connection pooling software products that are widely used for connection pooling for PostgreSQL: Pgpool-II and Pgbouncer. Both provide connection pooling as a basic minimum. While Pgbouncer is lightweight, it lacks HA features. Pgpool-II, on the other-hand, is considered a heavyweight and provides the enterprise-grade HA features. HA and load balancing are not supporting by Pgbouncer, whereas Pgpool-II supports HA through its Watchdog process and load balancing with intelligent routing of read/write queries to the primary database instance and read-only queries to standby database instances. Furthermore, Pgpool-II provides both command-line interface (CLI) and GUI administration interfaces.

6.2.1 Pgpool-II features

Pgpool-II is available for Linux and Solaris architectures, but not for Microsoft Windows. It is available for PostgreSQL V7.4 and later.

Table 6-1 explains the features of Pgpool-II that are used with FUJITSU Enterprise Postgres.

Table 6-1 Pgpool-II features

Feature category	Feature	Feature description
Performance Improvement	Connection pooling	Pgpool-II provides connection pooling between the clients and the database server by using the PostgreSQL back-end and front-end protocols and transferring the messages between the back end and the front end. Pgpool-II remains transparent between the application/clients (front end) and the FUJITSU Enterprise Postgres Database instances (back end). So, Pgpool-II may be used with the existing applications with no change to the source code.
	Load balancing	Pgpool-II provides load balancing by intelligently distributing the read-only queries across the available database instances by taking advantage of the replication that exists between the primary and the secondary database instances. This feature improves application throughput.
Pgpool-II HA	Watchdog	To avoid Pgpool-II being a single point of failure, Pgpool-II supports a HA cluster of three Pgpool-II features that are coordinated by the Watchdog subprocess of Pgpool-II. Watchdog exchanges information between multiple Pgpool-II nodes and ensures that the client connections are restricted to other Pgpool-II nodes when recovery is triggered.

Note: Pgpool-II also has features for conducting automatic failover from the primary database instance to the secondary standby database instance. It can also conduct online recovery by restoring or adding database servers without affecting the availability of the connected application. In FUJITSU Enterprise Postgres, automatic switchover is managed by the Mirroring Controller (MC) and Server Assistant. For more information about Pgpool-II features, see the Pgpool wiki.

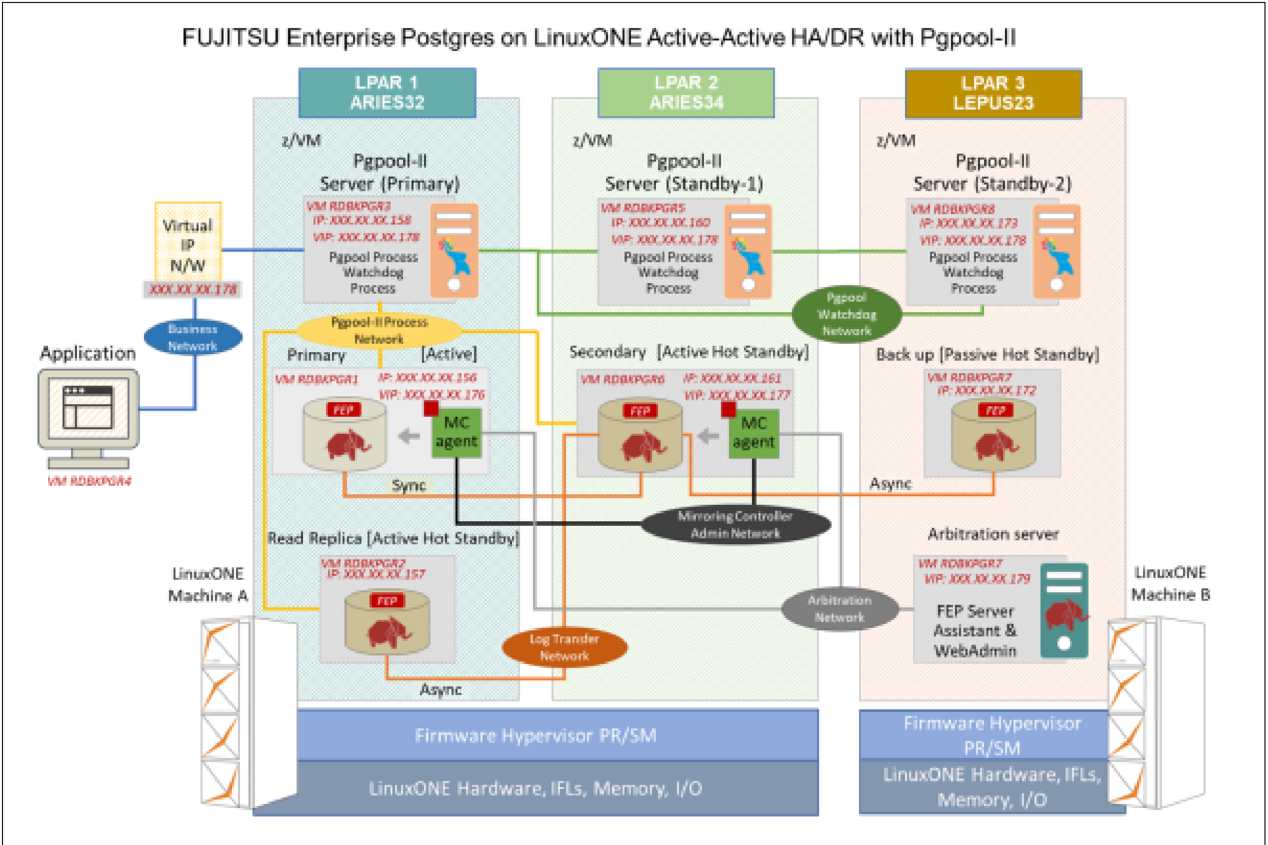
6.3 Installing and configuring Pgpool-II for FUJITSU Enterprise Postgres

In this section, we add Pgpool-II to the FUJITSU Enterprise Postgres HA architecture that we built in Chapter 5, “High availability and high reliability architectures” on page 165. Figure 6-2 on page 237 shows the lab environment that we used for showing a simple configuration of FUJITSU Enterprise Postgres Advanced Edition on IBM LinuxONE with Pgpool-II. Both FUJITSU Enterprise Postgres Advanced Edition and Pgpool-II are implemented with HA.

Figure 6-2 on page 237 shows the following configuration:

- Two IBM LinuxONE machines: Machine A and Machine B.
- Three logical partitions (LPARs): LPAR1, LPAR2, and LPAR3, each running z/VM:

  LPAR1 has three VMs, LPAR 2 has two VMs, and LPAR 3 has two VMs, each running Red Hat Enterprise Linux.
- FUJITSU Enterprise Postgres Advanced Edition HA configuration:
  - A primary database server with an MC (labeled as MC Agent) on VM RDBKPGR1 on LPAR1 ARIES32 on IBM LinuxONE Machine A.
  - A secondary database server with an MC (labeled as MC Agent) on VM RDBKPGR6 on LPAR2 ARIES34 on IBM LinuxONE Machine A in hot-standby mode with synchronous streaming replication with the primary database instance.
  - An arbitration server running FUJITSU Enterprise Postgres Server Assistant software on VM RDBKPGR7 on LPAR3 LEPUS23 on IBM LinuxONE Machine B for providing a quorum service to the primary and secondary database instances.
  - A backup database instance on VM RDBKPGR7 on LPAR3 LEPUS23 on IBM LinuxONE Machine B in passive hot-standby mode. Because the application runs in passive mode, it does not have connectivity to the backup database instance. It is also in hot standby because it is connected to a secondary database instance through asynchronous streaming replication. The arbitration server is also installed on VM RDBKPGR7.
- A read-replica database instance on VM RDBKPGR2 on LPAR1 ARIES32 on IBM LinuxONE Machine A. The read-replica instance is load balanced with the secondary database instance for read-only queries. Load balancing is provided by Pgpool-II.
- Pgpool-II HA configuration: Pgpool-II is configured in a three-unit HA architecture:
  - The Pgpool-II primary is installed and configured on VM RDBKPGR3 on LPAR1 ARIES32 on IBM LinuxONE Machine A.
  - Pgpool-II standby-1 is installed and configured on VM RDBKPGR5 on LPAR2 ARIES34 on IBM LinuxONE Machine A.
  - Pgpool-II standby-2 is installed and configured on VM RDBKPGR8 on LPAR3 LEPUS23 on IBM LinuxONE Machine B.



FUJITSU Enterprise Postgres on LinuxONE Active-Active HA/DR with Pgpool-II
LPAR 1 ARIES32
LPAR 2 ARIES34
LPAR 3 LEPUS23
z/VM
Pgpool-II Server (Primary)
Pgpool-II Server (Standby-1)
Pgpool-II Server (Standby-2)
VM RDBKPGR3 IP: XXX.XX.XX.158 VIP: XXX.XX.XX.178
VM RDBKPGR5 IP: XXX.XX.XX.160 VIP: XXX.XX.XX.178
VM RDBKPGR8 IP: XXX.XX.XX.173 VIP: XXX.XX.XX.178
Pgpool Process Watchdog Process
Virtual IP N/W
XXX.XX.XX.178
Business Network
Application
VM RDBKPGR4
Pgpool-II Process Network
Pgpool Watchdog Network
Primary [Active]
Secondary [Active Hot Standby]
Back up [Passive Hot Standby]
VM RDBKPGR1 IP: XXX.XX.XX.156 VIP: XXX.XX.XX.176
VM RDBKPGR6 IP: XXX.XX.XX.161 VIP: XXX.XX.XX.177
VM RDBKPGR7 IP: XXX.XX.XX.172
FEP
MC agent
Sync
Async
Mirroring Controller Admin Network
Read Replica [Active Hot Standby]
VM RDBKPGR2 IP: XXX.XX.XX.157
Arbitration server
VM RDBKPGR7 VIP: XXX.XX.XX.179
FEP Server Assistant & WebAdmin
Arbitration Network
Log Transfer Network
LinuxONE Machine A
LinuxONE Machine B
Firmware Hypervisor PR/SM
LinuxONE Hardware, IFLs, Memory, I/O

Business N/W	🟦	Uses Virtual IP: XXX.XX.XX.178	Application connects to Pgpool-II Virtual IP
Pgpool-II Watchdog N/W	🟩	Uses IP: XXX.XX.XX.158/160/173	High Availability of Pgpool-II using Watchdog
Pgpool-II Process N/W	🟨	Use IP: XXX.XX.XX.156/161/157	Pgpool-II Process for Load Balancing, Load Distribution and Connection Pooling
Arbitration N/W	⬜	Uses VIP: XXX.XX.XX.176/177/179	Quorum Service by Server Assistant
Log Transfer using Replication N/W	🟧	Uses IP: XXX.XX.XX.156/161/157/172	Streaming Replication b/w database instances
Mirroring Controller Admin N/W	⬛	Uses IP: XXX.XX.XX.156/161	High Availability of FUJITSU Enterprise Postgres

Figure 6-2 Pgpool-II HA configuration with FUJITSU Enterprise Postgres Advanced Edition in HA/disaster recovery mode

Note: This chapter assumes that you implemented the FUJITSU Enterprise Postgres HA architecture that is described in Chapter 5, "High availability and high reliability architectures" on page 165.

6.3.1 Installing and configuring Pgpool-II in a stand-alone configuration

In this section, you install and configure a stand-alone Pgpool-II on VM RDBKPGR3 on LPAR 1 on IBM LinuxONE Machine A. Complete the following steps:

1. Configure the primary, secondary, read-replica, and the backup database instances for Pgpool-II. Pgpool-II comes packaged with FUJITSU Enterprise Postgres. In this step, copy the Pgpool-II files, as shown in Example 6-1.

Example 6-1 Copying the Pgpool-II files

```
# cp -r /opt/fsepv12server64/OSS/Pgpool-II/* /opt/fsepv12server64
```

2. Configure the environment variables, as shown in Example 6-2.

Example 6-2 Configuring the environment variables

```
# su - fsepuser
$ PATH=/opt/fsepv12server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

3. On the primary database instance (xxx.xx.xx.156), create the `pgpool_recovery` extension, as shown in Example 6-3

Example 6-3 Creating the pgpool_recovery extension

```
$ psql -h xxx.xx.xx..156 -p 27500 -d postgres -U fsepuser
Password for user fsepuser:
psql (12.1)
Type "help" for help.

postgres=# CREATE EXTENSION pgpool_recovery;
CREATE EXTENSION
```

4. Next, modify the `postgresql.conf` file by adding the path for the parameter **`pgpool.pg_ctl`**. The command that is shown in Example 6-4 should be run for the primary (xxx.xx.xx.156), secondary (xxx.xx.xx.161), read-replica (xxx.xx.xx.157), and the backup (xxx.xx.xx.157) database instances.

Example 6-4 Adding the parameter path to the configuration file

```
$ vim /database/inst1/postgresql.conf
pgpool.pg_ctl= '/opt/fsepv12server64/bin/pg_ctl'
```

5. Stop the MC on the primary (xxx.xx.xx.156) and secondary (xxx.xx.xx.161) database servers by using the following command on each one:

   ```
   $ mc_ctl stop -M /mcdir/inst1
   ```

6. Start the MC on primary (xxx.xx.xx.156) and secondary (xxx.xx.xx.161) database servers by using the following command:

   ```
   $ mc_ctl start -M /mcdir/inst1
   ```

7. After recycling the MC on the primary and secondary database instances, restart the primary (xxx.xx.xx.156) and secondary (xxx.xx.xx.161) database instances by using the following command:

   ```
   $ pg_ctl restart -D /database/inst1
   ```

   Note: Run the command in step 7 on both the primary and secondary database instances.

   After the database instances are prepared and configured for Pgpool-II, continue with the installation and configuration of Pgpool-II on the VM RDBKPGR3 on LPAR 1. This VM acts as the active Pgpool-II server, as shown in Figure 6-2 on page 237.

8. On VM RDBKPGR3 (xxx.xx.xx.158) on LPAR1, install the libraries, as shown in Example 6-5 on page 239.

Example 6-5 Installing all the necessary libraries

```
# yum install libnsl
# yum install libicu
# yum install libmemcached
```

9. On VM RDBKPGR3 (xxx.xx.xx.158) on LPAR1, install Pgpool-II, as shown in Example 6-6.

Example 6-6 Installing Pgpool-II

```
# mount -t iso9660 -r -o loop /fep12iso/fsep12_client.iso /mnt2
# cd /mnt2/PGPOOL2/Linux/packages/r80s390x/
# rpm -ivh FJSVfsep-POOL2-12-1200-0.el8.s390x.rpm
```

10. On VM RDBKPGR3 (xxx.xx.xx.158) on LPAR1, install the FUJITSU Enterprise Postgres client software, as shown in Example 6-7.

Example 6-7 Installing the client software

```
# cd /mnt2/CLIENT64/Linux/packages/r80s390x/
# rpm -ivh FJSVfsep-CL-12-1200-0.el8.s390x.rpm
```

11. On VM RDBKPGR3 (xxx.xx.xx.158) on LPAR1, create directories for Pgpool-II, as shown in Example 6-8.

Example 6-8 Creating directories

```
# mkdir -p /pgpool/inst1
# chown -R pgpool:pgpool /pgpool
# chmod 700 /pgpool/inst1
```

12. On VM RDBKPGR3 (xxx.xx.xx.158) on LPAR1, grant access for Pgpool-II ports 9000, 9898, and 9999, as shown n Example 6-9.

Example 6-9 Granting access to ports

```
# vim /etc/firewalld/services/pgpool.xml
# cat /etc/firewalld/services/pgpool.xml
<?xml version="1.0" encoding="utf-8"?>
<service>
  <short>Pgpool</short>
  <description>Pgpool</description>
  <port protocol="tcp" port="9000"/>
  <port protocol="tcp" port="9898"/>
  <port protocol="tcp" port="9999"/>
</service>
# firewall-cmd --reload
# firewall-cmd --get-services | grep pgpool
# firewall-cmd --permanent --zone=public --add-service=pgpool
# firewall-cmd --reload
# firewall-cmd --list-all | grep pgpool
```

Note: Pgpool-II uses four ports for different operations. On port 9999, Pgpool-II accepts connections. On port 9898, PCP process accepts connections. Port 9000 is used by Watchdog to accept connections. Port 9694 is a UDP port for receiving Watchdog's heartbeat signal.

13. On VM RDBKPGR3 (xxx.xx.xx.158) on LPAR1, create user `pgpool` and assign a password, as shown in Example 6-10.

Example 6-10 Creating a user ID and password

```
# useradd -m pgpool
# passwd pgpool
```

14. Continuing to work on VM RDBKPGR3 (xxx.xx.xx.158) on LPAR1, create a file that is named `pgpool.conf` from the sample `pgpool` configuration file, as shown in Example 6-11.

Example 6-11 Creating the pgpool.conf file

```
$ cp -p /opt/fsepv12pgpool-II/etc/pgpool.conf.sample-stream
/pgpool/inst1/pgpool.conf
```

15. On VM RDBKPGR3 (xxx.xx.xx.158) on LPAR1, add parameters to the pgpool.conf file as shown in Example 6-12 for the primary (xxx.xx.xx.156), secondary (xxx.xx.xx.161), and read-replica (xxx.xx.xx.157) FUJITSU Enterprise Postgres Advanced Edition database instances.

Example 6-12 Adding configuration parameters to pgpool.conf

```
$ vim /pgpool/inst1/pgpool.conf
listen_addresses = '*'
backend_hostname0 = 'xxx.xx.xx.156'
backend_port0 = 27500
backend_data_directory0 = '/database/inst1'
backend_hostname1 = 'xxx.xx.xx.161'
backend_port1 = 27500
backend_data_directory1 = '/database/inst1'
backend_hostname2 = 'xxx.xx.xx.157'
backend_port2 = 27500
backend_data_directory2 = '/database/inst1'
pid_file_name = '/pgpool/inst1/pgpool.pid'
sr_check_user = 'fsepuser'
health_check_period=5
health_check_user='fsepuser'
health_check_database='postgres'
failover_on_backend_error = off
```

Note: The parameter **`health_check_period = 5`** sets the health check interval to 5 seconds. Also, we have set off the **`failover_on_backend_error`** parameter so that Pgpool-II reports only the error.

16. Create an md5 password for user ID `pgpool` on VM RDBKPGR3 (xxx.xx.xx.158), as shown in Example 6-13.

Example 6-13 Creating an md5 password

```
$pg_md5 pgpool
md5_pass
```

17. Configure the encrypted md5 password from step 16 in the `pcp.conf` configuration file on VM RDBKPGR3 (xxx.xx.xx.158), as shown in Example 6-14 on page 241.

Example 6-14 Configuring the md5 password

```
$ vim /pgpool/inst1/pcp.conf
pgpool:md5_pass
```

18. Start Pgpool on VM RDBKPGR3 (xxx.xx.xx.158), as shown in Example 6-15.

Example 6-15 Starting Pgpool

```
$ pgpool -f /pgpool/inst1/pgpool.conf -F /pgpool/inst1/pcp.conf -n >> pgpool.log 
2>&1 &
[1] 70944
```

19. Pgpool-II is now in a stand-alone configuration and running. Test the connectivity to the database instance by using a command from the application server VM RDBKPGR4, as shown in Example 6-16.

Example 6-16 Testing the connectivity

```
$ psql -d postgres -U fsepuser -h xxx.xx.xx.158 -p 9999 -c 'select 
inet_server_addr()'
 inet_server_addr
------------------
 xxx.xx.xx.156
(One row)
```

Note: IP address xxx.xx.xx.158 is the Pgpool-II stand-alone server. PostgreSQL function inet_server_addr() returns the IP address of the server on which the connection was accepted. In the results shown in Example 6-16, the connection was accepted on the primary database instance with the IP address xxx.xx.xx.156

In the previous steps, we installed and configured Pgpool-II and established a connection between the application VM RDBKPGR4 and the database instance through Pgpool-II. In the following steps, we configure load distribution and load balancing across the three available instances, which are primary, secondary, and read-replica FUJITSU Enterprise Postgres Advanced Edition instances. Complete the following steps:

1. Configure load balancing by setting the parameter **`load_balance_mode = on`** in the `pgpool.conf` configuration file on the Pgpool-II server (IP xxx.xx.xx.158). Set the load balancing weights equal to 1 for all three FUJITSU Enterprise Postgres Database nodes, which are the primary, secondary, and read-replica database instances. The configuration is shown in Example 6-17.

Example 6-17 Configuration parameters for load balancing

```
load_balance_mode = on
backend_weight0 = 1
backend_weight1 = 1
backend_weight2 = 1
```

2. Stop Pgpool-II on the VM RDBKPGR3. An example of this command and its results are shown in Example 6-18.

Example 6-18 Stopping Pgpool-II

```
$ pgpool -f /pgpool/inst1/pgpool.conf -F /pgpool/inst1/pcp.conf stop
2020-12-25 03:59:07: pid 814312: LOG:  stop request sent to pgpool. waiting for 
termination...
```

```
.done.
[1]+  Done                    pgpool -f /pgpool/inst1/pgpool.conf -F
/pgpool/inst1/pcp.conf -n >> pgpool.log 2>&1
```

3. Start the Pgpool-II on the VM RDBKPGR3, as shown in Example 6-19.

Example 6-19 Starting PGpool-II

```
$ pgpool -f /pgpool/inst1/pgpool.conf -F /pgpool/inst1/pcp.conf -n >> pgpool.log
2>&1 &
[1] 814322
```

4. Test the load balancing of Pgpool-II as shown in Example 6-20 from the application server VM RDBKPGR4. IP xxx.xx.xx.158 is the Pgpool-II server.

Example 6-20 Testing load balancing

```
$ psql -d postgres -U fsepuser -h xxx.xx.xx.158 -p 9999 -c 'select
inet_server_addr()'
 inet_server_addr
------------------
 xxx.xx.xx.161(1 row)
```

As the results show, the connection was accepted by the secondary database instance xxx.xx.xx.161.

In the previous steps above, we configured load balancing across primary, secondary, and read-replica instances with an equal weight given to all three database instances. In the following steps, we configure connection pooling:

1. In the pgpool.conf configuration file on the Pgpool-II server (IP xxx.xx.xx.158) on VM RDBKPGR3, set the parameter **`connection_cache=off`**. The **`connection_cache`** parameter, by default, is on. This parameter provides the connection pooling by caching the connections to FUJITSU Enterprise Postgres. We must first turn off the connection pooling to run a test.
2. Stop Pgpool-II on the VM RDBKPGR3 (Example 6-21).

Example 6-21 Stopping Pgpool-II

```
$ pgpool -f /pgpool/inst1/pgpool.conf -F /pgpool/inst1/pcp.conf stop
2020-12-25 03:59:07: pid 814312: LOG:  stop request sent to pgpool. waiting for
termination...
.done.
[1]+  Done                    pgpool -f /pgpool/inst1/pgpool.conf -F
/pgpool/inst1/pcp.conf -n >> pgpool.log 2>&1
```

3. Start the Pgpool-II on the VM RDBKPGR3 to refresh the file `pgpool.conf` (Example 6-22.).

Example 6-22 Starting Pgpool-II

```
$ pgpool -f /pgpool/inst1/pgpool.conf -F /pgpool/inst1/pcp.conf -n >> pgpool.log
2>&1 &
[1] 814322
```

4. On the application server VM RDBKPGR4, run the command that is shown in Example 6-23 on page 243 to create a database with the name `pgpooldb`.

Example 6-23 Creating a database

```
$ psql -d postgres -U fsepuser -h xxx.xx.xx.158 -p 9999 -c 'create database
pgpooldb'
CREATE DATABASE
```

5. From the application server VM RDBKPGR4, run the command that is shown in Example 6-24 to initialize **pgbench**, which is a simple program for running benchmark tests on PostgreSQL.

Example 6-24 Initializing pgbench

```
$ pgbench -h xxx.xx.xxxxx.xx.xx.158 -p 9999 -U fsepuser -i pgpooldb
dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping (11929)
NOTICE:  table "pgbench_branches" does not exist, skipping (11929)
NOTICE:  table "pgbench_history" does not exist, skipping (11929)
NOTICE:  table "pgbench_tellers" does not exist, skipping (11929)
creating tables...
generating data...
100000 of 100000 tuples (100%) done (elapsed 0.02 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done.
```

6. From the application server VM RDBKPGR4, run the command that is shown in Example 6-25 to run an application test, which runs transactions.

Example 6-25 Running an application test

```
$ pgbench -h xxx.xx.xx.158 -p 9999 -U fsepuser -C -T 30 pgpooldb
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 30 s
number of transactions actually processed: 3491
latency average = 8.594 ms
tps = 116.361924 (including connections establishing)
tps = 373.129425 (excluding connections establishing)
```

Note: Before running the **pgbench** test, turn off connection pooling by setting the parameter **connection_cache=off** in `pgpool.conf`. You see that the latency average is about 8.594 milliseconds, and the transactions per second is 116.36. In the next step, we switch connection pooling back on and rerun the pgbench test to see the difference between with connection pooling and without connection pooling.

7. To turn on connection pooling, change **connection_cache** to on in the Pgpool-II server VM RDBKPGR3 by editing the `pgpool.conf` configuration file on the Pgpool-II server (IP xxx.xx.xx.158) VM RDBKPGR3 and set the parameter **connection_cache** to `on`:

```
connection_cache = on
```

8. Stop Pgpool-II on the VM RDBKPGR3 server, as shown in Example 6-26.

Example 6-26 Stopping Pgpool-II

```
$ pgpool -f /pgpool/inst1/pgpool.conf -F /pgpool/inst1/pcp.conf stop
2020-12-25 04:00:41: pid 814366: LOG:  stop request sent to pgpool. waiting for
termination...
.done.
[1]+  Done                    pgpool -f /pgpool/inst1/pgpool.conf -F
/pgpool/inst1/pcp.conf -n >> pgpool.log 2>&1
```

9. Start Pgpool-II on the VM RDBKPGR3 to refresh `pgpool.conf` (Example 6-27).

Example 6-27 Starting Pgpool-II

```
$ pgpool -f /pgpool/inst1/pgpool.conf -F /pgpool/inst1/pcp.conf -n >> pgpool.log
2>&1 &
[1] 814369
```

10. From the application server VM RDBKPGR4, we run the **pgbench** command that is shown in Example 6-28 to run an application test that runs transactions.

Example 6-28 Running pgbench

```
$ pgbench -h xxx.xx.xx.158 -p 9999 -U fsepuser -C -T 30 pgpooldb
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
duration: 30 s
number of transactions actually processed: 4318
latency average = 6.948 ms
tps = 143.924903 (including connections establishing)
tps = 444.078874 (excluding connections establishing)
```

Note: Before running the **pgbench** test, we turned off connection pooling by setting the parameter **connection_cache=off** in `pgpool.conf`. We note the latency average = 8.594 milliseconds and the transactions per second is 116.36. In steps 7 on page 243 - 10, we turned on the connection pooling and reran the **pgbench** test again to see the difference between with connection pooling and without connection pooling. After connection pooling is turned on, we see that the latency average = 6.948 milliseconds and the transactions per second is 143.92.

6.3.2 Installing and configuring Pgpool-II in a highly available architecture

In this section, we install and configure two standby Pgpool-II servers on VM RDBKPGR5 on LPAR1 (ARIES32, IBM LinuxONE machine A) and RDBKPGR8 on LPAR 3 **(**LEPUS23, IBM LinuxONE Machine B). To provide a single IP address interface to the application server (VM RDBKPGR4), we configure a Virtual IP Address (VIPA) for the three Pgpool-II VMs: RDBKPGR3, RDBKPGR5, and RDBKPGR8 (see Figure 6-2 on page 237) by completing the following steps:

1. Add and configure two standby Pgpool-II servers on VM RDBKPGR5 and VM RDPKGR8 by completing steps 8 on page 238 - 18 on page 241 on VM RDBKPGR5 and then on VM RDBKPGR8.

Note: We refer to the Pgpool-II installation on VM RDBKPGR5 as Standby-1 and Pgpool-II installation on VM RDBKPGR8 as Standby-2 Pgpool-II.

2. Stop the Pgpool-II on the three Pgpool-II VMs (VMs RDBKPGR3, RDBKPGR5, and RDBKPGR8) one by one on all three Pgpool-II VMs by running the command that is shown in Example 6-29 on each of them.

Example 6-29 Stopping Pgpool-II

```
$ pgpool -f /pgpool/inst1/pgpool.conf -F /pgpool/inst1/pcp.conf stop
2020-12-25 03:53:50: pid 424747: LOG:  stop request sent to pgpool. waiting for
termination...
.done.
[1]+  Done                  pgpool -f /pgpool/inst1/pgpool.conf -F
/pgpool/inst1/pcp.conf -n >> pgpool.log 2>&1
```

Note: Shut down Pgpool-II on all three Pgpool-II VMs.

3. We identify the NIC device names so that we can configure VIPAs for the HA Pgpool-II implementation by running the **`ifconfig`** command on the Active, Standby-1, and Standby-2 Pgpool-II VMs, as shown in Example 6-30.

Example 6-30 Running the ifconfig command on each Pgpool-II virtual machine

```
$ ifconfig
enc640: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet xxx.xx.xx.158  netmask 255.255.255.0  broadcast xxx.xx.xx.255
        inet6 fe80::1:2ff:fe00:37  prefixlen 64  scopeid 0x20<link>
        ether 02:01:02:00:00:37  txqueuelen 1000  (Ethernet)
        RX packets 26765037  bytes 4490032089 (4.1 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6982353  bytes 6875883233 (6.4 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Note: Run the **`ifconfig`** command on the Active, Standby-1, and Standby-2 Pgpool-II VMs.

4. On the active Pgpool-II server VM RDBKPGR3, we configure the `pgpool.conf` parameters to enable Watchdog, provide a VIPA, and add the details of the Standby-1 and Standby-2 Pgpool-II servers, as shown in Example 6-31.

Example 6-31 Configuring parameters on the active Pgpool-II server

```
use_watchdog = on
wd_hostname = 'xxx.xx.xx.158'
wd_priority = 3
delegate_IP = 'xxx.xx.xx.178'
if_up_cmd = '/usr/bin/sudo /usr/sbin/ip addr add $_IP_$/24 dev enc640 label
enc640:0'
if_down_cmd = '/usr/bin/sudo /usr/sbin/ip addr del $_IP_$/24 dev enc640'
enable_consensus_with_half_votes = off
heartbeat_destination0 = ' xxx.xx.xx.160'
heartbeat_destination1 = ' xxx.xx.xx.173'
other_pgpool_hostname0 = ' xxx.xx.xx.160'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
other_pgpool_hostname1 = ' xxx.xx.xx.173'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
```

Note: The **`wd_priority`** parameter is set in the order of priority of master promotion. The parameter **`enable_consensus_with_half_votes`** defaults to `off`, in which case the virtual IP is not started without half + one pgpool. Up to Pgpool-II V4.0, the parameter defaults to `on`.

5. On the Standby-1 Pgpool-II server VM RDBKPGR5, we configure the `pgpool.conf` parameters to enable Watchdog, provide a VIPA, and add the details of the Active and Standby-2 Pgpool-II servers, as shown in Example 6-32.

Example 6-32 Configuring the parameters on Standby-1

```
use_watchdog = on
wd_hostname = 'xxx.xx.xx.160'
wd_priority = 2
delegate_IP = 'xxx.xx.xx.178'
if_up_cmd = '/usr/bin/sudo /usr/sbin/ip addr add $_IP_$/24 dev enc640 label
enc640:0'
if_down_cmd = '/usr/bin/sudo /usr/sbin/ip addr del $_IP_$/24 dev enc640'
enable_consensus_with_half_votes = off
heartbeat_destination0 = 'xxx.xx.xx.158'
heartbeat_destination1 = 'xxx.xx.xx.173'
other_pgpool_hostname0 = 'xxx.xx.xx.158'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
other_pgpool_hostname1 = 'xxx.xx.xx.173'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
```

6. On the Standby-2 Pgpool-II server VM RDBKPGR8, we configure the `pgpool.conf` parameters to enable Watchdog, provide a VIPA, and add the details of the Active and Standby-1 Pgpool-II servers, as shown in Example 6-33 on page 247.

Example 6-33 Configuring parameters on the Standby-2 Pgpool-II server

```
use_watchdog = on
wd_hostname = 'xxx.xx.xx.173'
wd_priority = 1
delegate_IP = 'xxx.xx.xx.178'
if_up_cmd = '/usr/bin/sudo /usr/sbin/ip addr add $_IP_$/24 dev enc640 label
enc640:0'
if_down_cmd = '/usr/bin/sudo /usr/sbin/ip addr del $_IP_$/24 dev enc640'
enable_consensus_with_half_votes = off
heartbeat_destination0 = 'xxx.xx.xx.158'
heartbeat_destination1 = 'xxx.xx.xx.160'
other_pgpool_hostname0 = 'xxx.xx.xx.158'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
other_pgpool_hostname1 = 'xxx.xx.xx.160'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
```

On the active Pgpool-II server VM RDBKPGR3, we start the Pgpool-II by using the following command:

```
$ pgpool -f /pgpool/inst1/pgpool.conf -n >> pgpool.log 2>&1 &
```

7. On the active Pgpool-II server VM RDBKPGR3, we check whether the VIPA xxx.xx.xx.178 is active by running the following command:

```
$ ifconfig | grep xxx.xx.xx.178
```

Note: VIPA xxx.xx.xx.178 is not yet up or active.

8. On the Standby-1 VM RDBKPGR5 and Standby-2 Pgpool-II server VM RDBKPGR8, we start the Pgpool-II by using the following command:

```
$ pgpool -f /pgpool/inst1/pgpool.conf -n >> pgpool.log 2>&1 &
```

Note: Run this command to start Pgpool-II on both Standby-1 and Standby-2 Pgpool-II servers.

9. On the active Pgpool-II server VM RDBKPGR3, we again check whether the VIPA xxx.xx.xx.178 is active by running the following command:

```
$ ifconfig | grep xxx.xx.xx.178
inet xxx.xx.xx.178  netmask 255.255.255.0  broadcast 0.0.0.0
```

Note: The VIPA xxx.xx.xx.178 is now active.

10. On the active Pgpool-II server VM RDBKPGR3, we now check the Pgpool-II Watchdog status by using the VIPA xxx.xx.xx.178, as shown in Example 6-34.

Example 6-34 Checking the Watchdog status on the active server

```
$ pcp_watchdog_info -h xxx.xx.xx.178 -p 9898 -U pgpool
Password:
3 YES xxx.xx.xx.158:9999 Linux rdbkpgr3.pbm.ihost.com xxx.xx.xx.158

xxx.xx.xx.158:9999 Linux rdbkpgr3.pbm.ihost.com xxx.xx.xx.158 9999 9000 4 MASTER
xxx.xx.xx.160:9999 Linux rdbkpgr5.pbm.ihost.com xxx.xx.xx.160 9999 9000 7 STANDBY
xxx.xx.xx.173:9999 Linux rdbkpgr8.pbm.ihost.com xxx.xx.xx.173 9999 9000 7 STANDBY
```

Note: VIPA xxx.xx.xx.178 is now up and the Pgpool-II Watchdog shows the three Unit Pgpool-II HA.

11. On the active Pgpool-II server VM RDBKPGR3, we now stop the Pgpool-II server, as shown in Example 6-35.

Example 6-35 Stopping the Pgpool-II server

```
$ pgpool -f /pgpool/inst1/pgpool.conf -F /pgpool/inst1/pcp.conf stop
2020-12-25 04:08:54: pid 814616: LOG:  stop request sent to pgpool. waiting for
termination...
.done.
[1]+  Done                    pgpool -f /pgpool/inst1/pgpool.conf -F
/pgpool/inst1/pcp.conf -n >> pgpool.log 2>&1
```

Note: Run the command that is shown in Example 6-35 on page 248 to stop Pgpool-II on the active Pgpool-II servers to verify that the VIPA is still available and see the status of the Pgpool-II Watchdog.

12. On the Standby-1 Pgpool-II server VM RDBKPGR5, we now check the status of the VIPA xxx.xx.xx.178, as shown in the following command:

```
$ ifconfig | grep xxx.xx.xx.178
inet xxx.xx.xx.178  netmask 255.255.255.0  broadcast 0.0.0.0
```

Note: VIPA xxx.xx.xx.178 is still available.

13. On the active Pgpool-II server VM RDBKPGR3, we now again check the Pgpool-II Watchdog status by using the VIPA xxx.xx.xx.178, as shown in Example 6-36.

Example 6-36 Checking the PgPool-II Watchdog status

```
$ pcp_watchdog_info -h xxx.xx.xx.178 -p 9898 -U pgpool
Password:
3 YES xxx.xx.xx.160:9999 Linux rdbkpgr5.pbm.ihost.com xxx.xx.xx.160

xxx.xx.xx.160:9999 Linux rdbkpgr5.pbm.ihost.com xxx.xx.xx.160 9999 9000 4 MASTER
xxx.xx.xx.158:9999 Linux rdbkpgr3.pbm.ihost.com xxx.xx.xx.158 9999 9000 10
SHUTDOWN
xxx.xx.xx.173:9999 Linux rdbkpgr8.pbm.ihost.com xxx.xx.xx.173 9999 9000 7 STANDBY
```

Note: The VIPA xxx.xx.xx.178 is now available, and the Pgpool-II Watchdog shows the two-unit Pgpool-II HA instead of three-unit one. The previous master with IP address xxx.xx.xx.158 is in the SHUTDOWN state, and the previous Standby with IP address xxx.xx.xx.160 is promoted to the new Master in the now two unit Pgpool-II HA.

14. On the active Pgpool-II server VM RDBKPGR3, we again start Pgpool-II, as shown in the following command, and bring Pgpool-II back to three units with the HA configuration:

```
$ pgpool -f /pgpool/inst1/pgpool.conf -n >> pgpool.log 2>&1 &
```

In the following steps, we simulate a primary database instance failure and check the operation and status of the HA Pgpool-II and FUJITSU Enterprise Postgres implementation:

1. On the primary FUJITSU Enterprise Postgres Database instance VM RDBKPGR1, check the status of the MC, as shown in Example 6-37.

Example 6-37 Checking the status of the Mirroring Controller

```
$ mc_ctl status --arbiter -M /mcdir/inst1
arbiter_id    host          status
---------------------------------------
arbiter27541  xxx.xx.xx.179  online
$ mc_ctl status -M /mcdir/inst1
mirroring status
-----------------
switchable

server_id  host_role  host           host_status  db_proc_status  disk_status
-----------------------------------------------------------------------------
inst1p     primary    xxx.xx.xx.156  normal       normal          normal
inst1s     standby    xxx.xx.xx.161  normal       normal          normal
```

Note: The servers `inst1p` and `inst1s` are working with a status of `normal`.

2. On the active Pgpool-II server VM RDBKPGR3, we check the Pgpool-II Watchdog status by using the VIPA xxx.xx.xx.178, as shown in Example 6-38.

Example 6-38 Checking the status of Watchdog

```
$ pcp_watchdog_info -h xxx.xx.xx.178 -p 9898 -U pgpool
Password:
3 YES xxx.xx.xx.158:9999 Linux rdbkpgr3.pbm.ihost.com xxx.xx.xx.158

xxx.xx.xx.158:9999 Linux rdbkpgr3.pbm.ihost.com xxx.xx.xx.158 9999 9000 4 MASTER
xxx.xx.xx.160:9999 Linux rdbkpgr5.pbm.ihost.com xxx.xx.xx.160 9999 9000 7 STANDBY
xxx.xx.xx.173:9999 Linux rdbkpgr8.pbm.ihost.com xxx.xx.xx.173 9999 9000 7 STANDBY
```

Note: VIPA xxx.xx.xx.178 is now available, and the Pgpool-II Watchdog shows a three-unit Pgpool-II with the HA configuration.

3. From the application server VM RDBKPGR4, we create a temporary table on FUJITSU Enterprise Postgres and check the database instance that runs the create table statement, as shown in Example 6-39.

Example 6-39 Creating a temporary table

```
$ psql -d pgpooldb -U fsepuser -h xxx.xx.xx.178 -p 9999 -c 'create temporary table
pgpool_t1(c1 int);select inet_server_addr();'
 inet_server_addr
------------------
 xxx.xx.xx.156
(One row)
```

Note: The results show that the **`UPDATE`** statement runs on the primary database instance with the IP xxx.xx.xx.156 because the **`CREATE`** SQL statements are categorized as read/write statements, which run on the primary database instance.

4. To simulate automatic failover and promotion of the secondary active hot-standby database instance to the primary, we shut down the `postgres` process on the primary database server VM RDBKPGR1, as shown in Example 6-40.

Example 6-40 Shutting down the postgres process

```
$ pg_ctl stop -D /database/inst1 -mi
waiting for server to shut down.... done
server stopped
```

5. On the primary FUJITSU Enterprise Postgres Database instance VM RDBKPGR1, check the status of the MC, as shown in Example 6-41.

Example 6-41 Checking the status of the Mirroring Controller

```
'$ mc_ctl status --arbiter -M /mcdir/inst1
arbiter_id    host          status
-----------------------------------------
arbiter27541  xxx.xx.xx.179  online
$ mc_ctl status -M /mcdir/inst1
mirroring status
-----------------
switched

server_id  host_role               host          host_status  db_proc_status
disk_status
-----------------------------------------------------------------------------------
-----------------
inst1p     none(inactivated primary)  xxx.xx.xx.156  normal
abnormal(postmaster)  normal
inst1s     primary                    xxx.xx.xx.161  normal
abnormal(wal_sender)  normal
```

Note: The `inst1p` server is now an `inactivated` database instance, and `inst1s` is promoted automatically by the arbitration server to become the new primary.

6. From the application server VM RDBKPGR4, we again create a temporary table in the FUJITSU Enterprise Postgres Database and verify that the new primary database instance ('*inst1s'* on VM RDBKPGR6) runs the create table statement (Example 6-42).

Example 6-42 Creating and verifying that the create table statement ran

```
$ psql -d pgpooldb -U fsepuser -h xxx.xx.xx.178 -p 9999 -c 'create temporary table
pgpool_t1(c1 int);select inet_server_addr();'
 inet_server_addr
------------------
 xxx.xx.xx.161
(One row)
```

Note: The results show that the **`UPDATE`** statement runs on the new primary database instance with the IP xxx.xx.xx.161 because the **`CREATE`** statement that shows that the promotion of the secondary to the new primary was transparent to the application. The application continued using the Pgpool-II and FUJITSU Enterprise Postgres HAty implementation.

7

Application development, compatibility features, and migration

In this chapter, we describe topics that are related to application development, FUJITSU Enterprise Postgres compatibility features compared to other enterprises databases, and aspects of migration to FUJITSU Enterprise Postgres from other databases.

In this chapter, we use the following environment to describe the application development use cases:

- Red Hat Enterprise Linux 8 on IBM LinuxONE
- Two FUJITSU Enterprise Postgres instances with streaming replication (primary + standby)
- Application that uses a JDBC driver or a C library (libpq)
- FUJITSU Enterprise Postgres Server installation path `/opt/fsepv12server64`
- FUJITSU Enterprise Postgres Client installation path `/opt/fsepv12client64`

This chapter covers the following topics:

- Application development by using database drivers
- Connection Manager
- Oracle compatibility features
- Migrating to FUJITSU Enterprise Postgres

7.1 Application development by using database drivers

In this section, we describe application development by using supported drivers, the steps to configure and use database drivers, and the application connection switch feature on FUJITSU Enterprise Postgres 12 client with Red Hat Enterprise Linux 8 on IBM LinuxONE.

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

The application development interface that is provided by FUJITSU Enterprise Postgres is compatible with PostgreSQL.

7.1.1 JDBC driver

The FUJITSU Enterprise Postgres client installation provides JDBC driver files for Java SE Development Kit or JRE V6 onwards. Based on the version of Java SE Development Kit or JRE that is installed on the system, the **`CLASSPATH`** environment variable is configured with the correct JDBC driver file. In this section, we describe how to set up the driver file. Table 7-1 shows the compatible driver files for different versions of the Java SE Development Kit or JRE.

Table 7-1 Driver files for Java SE Development Kit or JRE versions

Java SE Development Kit or JRE version	JDBC driver file
Java SE Development Kit 6 or JRE 6	`postgresql-jdbc4.jar`
Java SE Development Kit 7 or JRE 7	`postgresql-jdbc41.jar`
Java SE Development Kit 8, JDK11, JRE 8, or JRE 11	`postgresql-jdbc42.jar`

Example 7-1 demonstrates the configuration and setup of the JDBC driver file for Java SE Development Kit 8. The command remains the same for all other versions; only the JDBC driver file name changes.

Example 7-1 Setting the environment variable for the JDBC driver

```
[fsepclient@rdbkpgr3 ~]$ java -version
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 8.0.6.15 - pxz6480sr6fp15-20200724_01(SR6
FP15))
IBM J9 VM (build 2.9, JRE 1.8.0 Linux s390x-64-Bit Compressed References
20200724_452227 (JIT enabled, AOT enabled)
OpenJ9   - 4ce4b9d
OMR      - 08b0594
IBM      - 70917a2)
JCL - 20200720_01 based on Oracle jdk8u261-b13
[fsepclient@rdbkpgr3 ~]$
CLASSPATH=/opt/fsepv12client64/jdbc/lib/postgresql-jdbc42.jar:$CLASSPATH;export
CLASSPATH
[fsepclient@rdbkpgr3 ~]$
```

Connecting to the database by using the JDBC driver

Here are the steps to connect to the databases by using the DriverManager class. The application code to use the DriverManager class is shown in Example 7-2.

1. Load the JDBC driver.
2. Define a connection string.

Example 7-2 Application code by using the DriverManager class

```
import java.sql.Connection;
//import DriverManager
import java.sql.DriverManager;
………

public class jdbc_test {
        public static void main (String[] args) {
………
                // load JDBC driver
                Class.forName("org.postgresql.Driver");

 // Connection String
               String url =
"jdbc:postgresql://xxx.xx.xx.158:27500/postgres?user=fsepuser&password=fsepuser&lo
ginTimeout=20&socketTimeout=20";

                // connection
                conn = DriverManager.getConnection(url);
………
}
```

To connect the databases by using the `PGConnectionPoolDataSource`, set the data source properties with connection parameters, as shown in Example 7-3.

Example 7-3 Application code by using the PGConnectionPoolDataSource class

```
import java.sql.Connection;
//import PGConnectionPoolDataSource
import org.postgresql.ds.PGConnectionPoolDataSource;
………
public class jdbc_test1{
        public static void main (String[] args) {
   //Declare Data Source and specify connection parameters
                PGConnectionPoolDataSource source = new
PGConnectionPoolDataSource();
                source.setServerName("xxx.xx.xx.158");
                source.setPortNumber(27500);
                source.setDatabaseName("postgres");
                source.setUser("fsepuser");
                source.setLoginTimeout(20);
                source.setSocketTimeout(20);

                // connection
                conn = source.getConnection();
………
```

To connect to the database by using the `PGXADataSource` class, set the data source properties with connection parameters, as shown in Example 7-4.

Example 7-4 Application code that uses the PGXADataSource class

```
import java.sql.Connection;
//import PGXADataSource;
import org.postgresql.xa.PGXADataSource;
.........
public class jdbc_test1{
        public static void main (String[] args) {
   // Declare Data Source and specify connection parameters
                PGXADataSource source = new PGXADataSource();
                source.setServerName("xxx.xx.xx.158");
                source.setPortNumber(27500);
                source.setDatabaseName("postgres");
                source.setUser("fsepuser");
                source.setLoginTimeout(20);
                source.setSocketTimeout(20);

                // connection
                conn = source.getConnection();
.........
```

7.1.2 ODBC driver

FUJITSU Enterprise Postgres supports ODBC V3.5. In this section, we describe how to set up the data drivers and register the ODBC data sources.

Registering the ODBC data drivers.

To register the ODBC data drivers, complete the following steps:

1. Install the ODBC driver manager.
2. Edit the `odbcinst.ini` file of the ODBC driver manager.
3. Register the ODBC driver by running the command that is shown in Example 7-5.

Example 7-5 Registering the ODBC driver

```
//install unixODBC
[root@rdbkpgr3 ~]# yum install unixODBC
.........
Installed:
  unixODBC-2.3.7-1.el8.s390x

//Verify the directory of installed driver
[root@rdbkpgr3 ~]# odbcinst -j
unixODBC 2.3.7
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /root/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
//Edit odbcinst.ini file
```

```
[root@rdbkpgr3 /]# vi /etc/odbcinst.ini
//Add the below details to odbcinst.ini file
[FUJITSU Enterprise Postgres12s390xansi]
Description = FUJITSU Enterprise Postgres 12 s390x Unicode driver
Driver64 = /opt/fsepvclient64/odbc/lib/psqlodbcw.so
FileUsage = 1
Threading = 2
```

Registering ODBC data sources

To register the ODBC data sources and configure the environment variables, complete the following steps:

1. Register the ODBC data sources.

   To register the ODBC data sources, edit the `odbc.ini` definition file, as shown in Example 7-6.

Example 7-6 Registering ODBC data sources

```
// Edit odbc.ini file
[root@rdbkpgr3 /]# vi /etc/odbc.ini

//Add the below details to odbc.ini file
[MyDataSource]
Description = FUJITSU Enterprise Postgres
Driver = FUJITSU Enterprise Postgres12s390xansi
Database = postgres
Servername = xxx.xx.xx.158
Username = fsepuser
Port = 27500
ReadOnly
```

   We edited the `odbc.ini` file that is available in the directory `unixODBCInstallDir`. These settings are shared by all users that log in to the system. If you want the settings to be saved for a single user, create a `odbc.ini` file in the `$HOME` directory of the specific user.

2. Configure the environment variables.

   To run applications, set the FUJITSU Enterprise Postgres client libraries `unixODBC` and `libtool` libraries path to the **`LD_LIBRARY_PATH`** environment variable, as shown in Example 7-7.

Example 7-7 Configuring LD_LIBRARY_PATH

```
// Update .bashrc file to include updated environment variables.
vi ~/.bashrc

// Add the following line to ~/.bashrc
export LD_LIBRARY_PATH=/opt/fsepv12client64/lib:$LD_LIBRARY_PATH

// Run the following command to check the database connection by using the ODBC
driver: isql -v <ODBC data source> <username> <password>
//The following is what we used in our lab environment:
isql -v MyDataSource fsepuser fsepuser
```

   We installed `unixODBC` and `libtool` to the default path, so there was no need to include a `unixODBC` and libtool libraries path. If you installed `unixODBC` and `libtool` to a different path, then set it in **`LD_LIBRARY_PATH`**.

7.1.3 C library (libpq)

The C application programming interface (API) to PostgreSQL, `libpq`, is a set of library functions that allow client programs to pass queries to the PostgreSQL back-end server and to receive the results of these queries.

To run `libpq`, configure the following environment variables:

1. To compile the code and link the libraries, set the FUJITSU Enterprise Postgres client libraries path to `LD_LIBRARY_PATH`, as shown in Example 7-8.
2. To run the application, set the FUJITSU Enterprise Postgres shared locale path to `PGLOCALEDIR`.

Example 7-8 Configuring LD_LIBRARY_PATH

```
// Update .bashrc file to include updated environment variables.
vi ~/.bashrc

// Add below line to ~/.bashrc
export LD_LIBRARY_PATH=/opt/fsepv12client64/lib:$LD_LIBRARY_PATH
export PGLOCALEDIR=/opt/fsepv12client64/share/locale:$PGLOCALEDIR
```

While compiling the application, include the following paths:

- FUJITSU Enterprise Postgres Include file path: `/opt/fsepv12client64/include`
- FUJITSU Enterprise Postgres Library Path: `/opt/fsepv12server64/lib`

The sample code that is shown in Example 7-9 is used to build the C application, and it is taken from Chapter 33, “libpq - C Library”, of PostgreSQL 12.7 Documentation.

Example 7-9 Compiling the sample code

```
// Sample Code files.
[fsepclient@rdbkpgr3 ~]$ ls *.c
testlibpq.c  testlibpq_insert.c  testlibpq_retry.c

// Compile the sample code.
[fsepclient@rdbkpgr3 ~]$ cc -c -I/opt/fsepv12client64/include testlibpq.c
[fsepclient@rdbkpgr3 ~]$ cc -o testprog testlibpq.o -L/opt/fsepv12client64/lib
-lpq
```

7.1.4 Embedded SQL in C

When using embedded SQL in C, the same environment settings that are used when using the C library (libpq) are required. You must set the path for the precompiler ecpg in the PATH environment variable, as shown in Example 7-10.

Example 7-10 Setting the PATH environment variable

```
// Update .bashrc file to include updated environment variables.
vi ~/.bashrc

// Add below line to ~/.bashrc
export PATH=/opt/fsepv12clinet64/bin:$PATH
```

For more information about how to connect by using the client driver, see the *FUJITSU Enterprise Postgres 12 Application Development Guide*.

7.1.5 Application connection switch feature

The application connection switch feature enables an automatic connection to the target database server when there are multiple servers with redundant configurations. To use this feature, we specify the details of primary server and secondary server in the application connection information.

Using the application connection switch feature

The application connection switch feature connection parameter includes details of available database instances and the selection sequence of servers to which an application connects.

This section explains the parameters that are specific to the application connection switch feature. Table 7-2 describes these parameters.

Table 7-2 Application connection switch parameters

Parameter	Description
`host1, host2`	Specify the IP address or hostname of the database instances.
`port1, port2`	Specify the port number of the database instances.
`targetServerType / target_session_attrs`	Specify the selection sequence of the server to which the application connects.
`sslservercertcn`	Specify the Secure Sockets Layer (SSL) server certificate Common Name (CN). This parameter is used to perform SSL authentication by creating the same server certificate for each server. If it is omitted, the value is null, and the server certificate CN is authenticated by using the hostname that is specified in the host.

For more information about the application connection switch feature parameters, see the *FUJITSU Enterprise Postgres 12 Application Development Guide.*

Table 7-3 shows the value of the application connection switch parameter **`targetServerType`** (for JDBC driver) and **`target_session_attrs`** for drivers other than JDBC.

Table 7-3 Driver values

Server selection order	JDBC drivers	Other drivers
Primary server	`master` or `primary`	`read-write`
Standby server	`subordinate secondary`	-
Priority to standby server	`preferSlave` or `preferSecondary`	`prefer-read`
Any (default)	`any`	`any`

Note: The words *master* and *slave* are being deprecated. The latest versions of the driver silently accept them, but primary and secondary are now accepted as well.

An example of the use of these values is shown in Example 7-11 and Example 7-12

Example 7-11 Connection parameters for JDBC driver with IPv4

```
// Connection String parameters IPv4 and targetServerType "primary".

String url =
"jdbc:postgresql://xxx.xx.xx.158:27500,xxx.xx.xx.158:27501/postgres?targetServerTy
pe=primary&user=fsepuser&loginTimeout=20&socketTimeout=20";
```

Notice in Example 7-12 when using IPv6 with the JDBC driver, the host is specified in square brackets [host]. For all other drivers, you specify the host as `host`.

Example 7-12 Connection parameters for JDBC driver with IPv6

```
// Connection String parameters IPv6 and targetServerType "preferSecondary".

String url =
"jdbc:postgresql://[::1]:27500,xxx.xx.xx.158:27501/postgres?targetServerType=prefe
rSecondary&user=fsepuser&loginTimeout=20&socketTimeout=20";
```

Example 7-13 shows the connection parameters that are used with an ODBC driver.

Example 7-13 Connection parameters for ODBC driver

```
// Connection parameters target_session_attrs "read-write".

Servername={xxx.xx.xx.158, xxx.xx.xx.158}
Port={27500,27501}
target_session_attrs=read-write
```

7.1.6 Connecting an application to a database by using the application connection switch feature

FUJITSU Enterprise Postgres allows you to connect an application to a database by using the application connection switch feature. In this section, we use the following environment to describe three scenarios that demonstrate how to accomplish this connection with the application connection switch feature:

- Two FUJITSU Enterprise Postgres instanceswith Streaming Replication (Primary + Standby)
- Application that uses the JDBC driver

Scenario 1: Both database instances are running

In this scenario, we perform the following steps to demonstrate how to connect an application to a database by using the application connection switch feature when both databases instances are running:

1. In Example 7-14 on page 259, we create a simple Java application, **`jdbc_test`**, that uses the JDBC driver and DriverManager class to connect to the database by using the application connection switch feature.

Example 7-14 DriverManager class that is used in application code

```
import java.sql.Connection;
//import DriverManager
import java.sql.DriverManager;
………

public class jdbc_test {
        public static void main (String[] args) {
………
                // load JDBC driver
                Class.forName("org.postgresql.Driver");

 // Connection String master xxx.xx.xx.158:27500, slave xxx.xx.xx.158:27501,
targetServerType=slave
               String url =
"jdbc:postgresql://xxx.xx.xx.158:27500,xxx.xx.xx.158:27501/postgres?targetServerTy
pe=slave&user=fsepuser&loginTimeout=20&socketTimeout=20";

                // connection
                conn = DriverManager.getConnection(url);
………
}
```

2. The targetServerType in Example 7-14 is defined as `slave`. Compile and run the Java code, as shown in Example 7-15.

Example 7-15 Compiling and running the application with targetServerType as slave

```
[fsepclient@rdbkpgr3 java_work]$ javac jdbc_test.java
[fsepclient@rdbkpgr3 java_work]$ java jdbc_test
1
2
3
200
exit !!
```

Because we are using **`targetServerType=slave`**, the request is routed to the standby server. In our logs, we can verify where the request is being routed by looking at the **`syslog_ident`** value, as shown in Example 7-16. In our case, it is `FEP2` that is the standby server.

Example 7-16 Verifying the request is routed to the standby server

```
[lnxadmin@rdbkpgr3 log]$ sudo tail -4 messages
Nov  8 19:08:16 rdbkpgr3 FEP2[77926]: [8-1] 2020-11-08 19:08:16.109 EST [77926]
LOG:  execute <unnamed>: show transaction_read_only (11091)
Nov  8 19:08:16 rdbkpgr3 FEP2[77926]: [9-1] 2020-11-08 19:08:16.109 EST [77926]
LOG:  execute <unnamed>: SET extra_float_digits = 3 (11091)
Nov  8 19:08:16 rdbkpgr3 FEP2[77926]: [10-1] 2020-11-08 19:08:16.109 EST [77926]
LOG:  execute <unnamed>: SET application_name = 'PostgreSQL JDBC Driver' (11091)
Nov  8 19:08:16 rdbkpgr3 FEP2[77926]: [11-1] 2020-11-08 19:08:16.119 EST [77926]
LOG:  execute <unnamed>: SELECT * FROM tbl (11091)
```

3. Change the Java code that is shown in Example 7-14 so that the application connection switch parameter, **`targetServerType`**, is now `master` or `primary`.

4. Compile and run the Java code, as shown in Example 7-17.

Example 7-17 Compiling and running the application with targetServerType=master

```
[fsepclient@rdbkpgr3 java_work]$ javac jdbc_test.java
[fsepclient@rdbkpgr3 java_work]$ java jdbc_test
1
2
3
200
200
exit !!
```

Because we are using **targetServerType=master**, the request is routed to the master server. In our logs, we can verify where the request is being routed by looking at the **syslog_ident** value, as shown in Example 7-18. In our case, it is `FEP1` that is now the standby server.

Example 7-18 Verifying that the request is routed to the standby server

```
Nov  9 23:33:29 rdbkpgr3 FEP1[80829]: [8-1] 2020-11-09 23:33:29.721 EST [80829]
LOG:  execute <unnamed>: show transaction_read_only (11091)
Nov  9 23:33:29 rdbkpgr3 FEP1[80829]: [9-1] 2020-11-09 23:33:29.725 EST [80829]
LOG:  execute <unnamed>: SET extra_float_digits = 3 (11091)
Nov  9 23:33:29 rdbkpgr3 FEP1[80829]: [10-1] 2020-11-09 23:33:29.725 EST [80829]
LOG:  execute <unnamed>: SET application_name = 'PostgreSQL JDBC Driver' (11091)
Nov  9 23:33:29 rdbkpgr3 FEP1[80829]: [11-1] 2020-11-09 23:33:29.735 EST [80829]
LOG:  execute <unnamed>: SELECT * FROM tbl (11091)
```

5. Insert data into a table with the connection parameter **targetServerType** as `slave`, as shown in Example 7-19.

Example 7-19 Inserting data in a table with targetServerType as slave

```
[fsepclient@rdbkpgr3 java_work]$ java jdbc_test
org.postgresql.util.PSQLException: ERROR: cannot execute INSERT in a read-only
transaction (11113)
        at
org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.ja
va:2497)
        at
org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:223
3)
        at
org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:310)
        at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:446)
        at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:370)
        at org.postgresql.jdbc.PgStatement.executeWithFlags(PgStatement.java:311)
        at org.postgresql.jdbc.PgStatement.executeCachedSql(PgStatement.java:297)
        at org.postgresql.jdbc.PgStatement.executeWithFlags(PgStatement.java:274)
        at org.postgresql.jdbc.PgStatement.executeUpdate(PgStatement.java:246)
        at jdbc_test.main(jdbc_test.java:30)
exit !!
```

The standby server is in read-only mode, so no write operation can be performed when using **targetServerType=slave**.

Scenario 2: Database primary instance is down

In this scenario, we perform the following steps to demonstrate how to connect an application to a database by using the application connection switch feature when the database primary *instance* is down and an application makes a connection request:

1. Stop the database primary instance, as shown in Example 7-20.

Example 7-20 Stopping the database primary instance

```
[fsepuser@rdbkpgr3 ~]$ pg_ctl stop -D /database/inst1
waiting for server to shut down.... done
server stopped
```

2. Run the application with **targetServerType** as `MASTER`, which allows only read-only transactions, as shown in Example 7-21.

Example 7-21 Reading data from a table with targetServerType as MASTER

```
[fsepclient@rdbkpgr3 java_work]$ java jdbc_test_read
org.postgresql.util.PSQLException: Could not find a server with specified
targetServerType: master
        at org.postgresql.Driver$ConnectThread.getResult(Driver.java:409)
        at org.postgresql.Driver.connect(Driver.java:267)
        at java.sql.DriverManager.getConnection(DriverManager.java:675)
        at java.sql.DriverManager.getConnection(DriverManager.java:281)
        at jdbc_test_read.main(jdbc_test_read.java:25)
exit !!
```

3. Run the application with **targetServerType** as `ANY` with read-only transactions, as shown in Example 7-22.

Example 7-22 Reading data from a table with targetServerType as ANY

```
[fsepclient@rdbkpgr3 java_work]$ java jdbc_test_read
1
2
3
200
200
exit !!
```

4. Run the application with **targetServerType** as `ANY` and having write transactions, as shown in Example 7-23.

Example 7-23 Inserting data into a table with the database primary instance down and targetServerType as ANY

```
[fsepclient@rdbkpgr3 java_work]$ java jdbc_test
org.postgresql.util.PSQLException: ERROR: cannot execute INSERT in a read-only
transaction (11113)
        at
org.postgresql.core.v3.QueryExecutorImpl.receiveErrorResponse(QueryExecutorImpl.ja
va:2497)
        at
org.postgresql.core.v3.QueryExecutorImpl.processResults(QueryExecutorImpl.java:223
3)
        at
org.postgresql.core.v3.QueryExecutorImpl.execute(QueryExecutorImpl.java:310)
```

```
        at org.postgresql.jdbc.PgStatement.executeInternal(PgStatement.java:446)
        at org.postgresql.jdbc.PgStatement.execute(PgStatement.java:370)
        at org.postgresql.jdbc.PgStatement.executeWithFlags(PgStatement.java:311)
        at org.postgresql.jdbc.PgStatement.executeCachedSql(PgStatement.java:297)
        at org.postgresql.jdbc.PgStatement.executeWithFlags(PgStatement.java:274)
        at org.postgresql.jdbc.PgStatement.executeUpdate(PgStatement.java:246)
        at jdbc_test.main(jdbc_test.java:30)
exit !!
```

Scenario 3: Database primary server is down

In this scenario, we demonstrate how to connect an application to a database by using the application connection switch feature when the database primary server is down and an application makes a connection request. The database instance comes back online while the application is retrying the connection.

Example 7-24 shows an application connecting with the **`targetServerType`** as `master` and displaying read transactions.

Example 7-24 Database primary instance comes back online

```
[fsepclient@rdbkpgr3 java_work]$ java jdbc_test_retry
sleep a few sec.
finally close
Connected!
Wait 10sec.
##########
1
2
3
200
200
200
finally close
exit !!
[fsepclient@rdbkpgr3 java_work]$
```

Scenario 4: Actions taken when a database primary server is down

In Example 7-26 on page 263, we demonstrate the actions that are taken when an application makes a connection request and the database primary server is down. The standby instance is promoted while the application retries the connection.

Example 7-25 Standby instance is promoted

```
[fsepclient@rdbkpgr3 java_work]$ java jdbc_test_retry
sleep a few sec.
finally close

.........
```

While the application is retrying the connection to the database, Example 7-26 shows the promotion of the standby instance from another console.

Example 7-26 Promoting the standby instance

```
[fsepuser@rdbkpgr3 ~]$ pg_ctl promote -D /database/inst2
waiting for server to promote.... done
server promoted
Connected!

………
sleep a few sec.
finally close
Connected!
Wait 10sec.
##########
1
2
3
200
200
200
200
finally close
exit !!
```

Note: The application connection switch feature is used for database multiplexing. It does not include automatic switching from primary to standby. For automatic switchover from primary to standby in a failover, use the Mirroring Controller (MC).

For more information about FUJITSU Enterprise Postgres MC, see *FUJITSU Enterprise Postgres 12 Cluster Operation Guide*.

7.2 Connection Manager

In this section, we describe the steps to configure and use Connection Manager on a FUJITSU Enterprise Postgres 12 server with Red Hat Enterprise Linux Server 8 on IBM LinuxONE.

FUJITSU Enterprise Postgres Connection Manager provides heartbeat monitoring and transparent connection support features. The Connection Manager monitors client/server and FUJITSU Enterprise Postgres instances running on the database server. If there is a physical server failure and the inter-server network link goes down, the Connection Manager notifies the client and the database instance. The Connection Manager transparent connection feature allows the application to connect to an instance from a set of instances that is configured for replication. The connection to an underlying database instance is transparent to the application.

FUJITSU Enterprise Postgres Connection Manager is available by default with FUJITSU Enterprise Postgres V12.

For more information about FUJITSU Enterprise Postgres Connection Manager features, see “Transparent application switching and using Connection Manager” on page 26.

Here are some benefits of using Connection Manager:

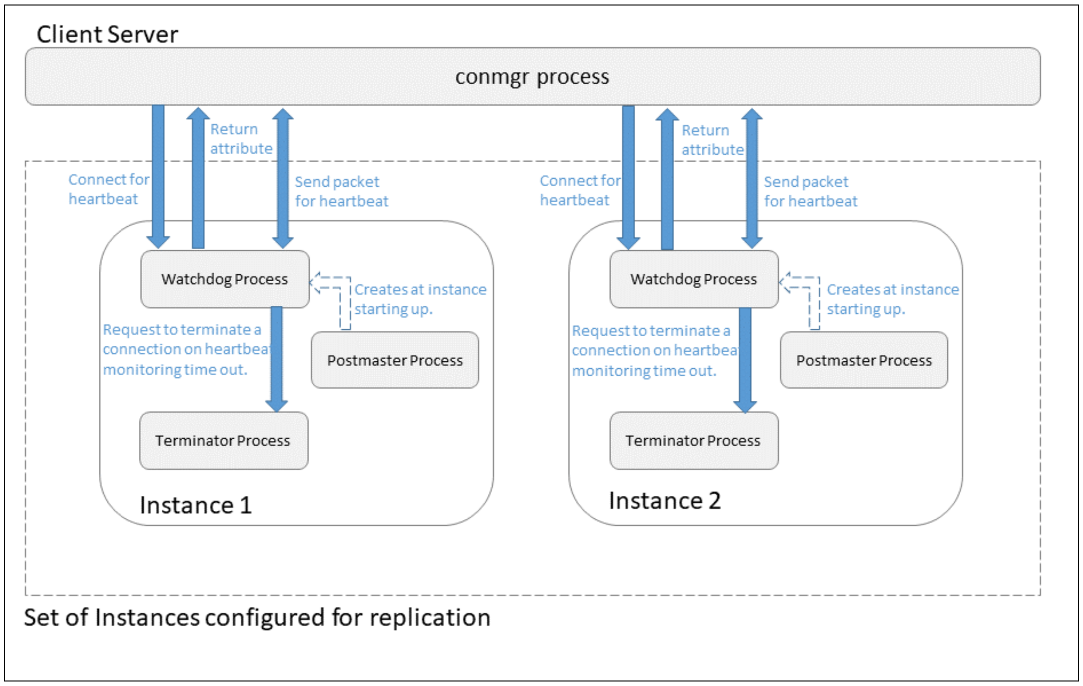
- Automatic reaping of connection happens when an abnormality is observed on the application server.
- The connection to the database instance is transparent to the application, and the application is notified at the time of the database instance failure.
- The application connection switches to the available database instance when database failover happens.
- After a connection is established, the application requests directly go to the database instance, so there is no performance degradation.

7.2.1 Configuring the Connection Manager

Connection Manager is configured on the client/server and the database server. On the client side, a monitoring process is started that is called the conmgr process to monitor the set of database instances.

On the server side, the PostgreSQL extension Watchdog is installed. Due to the Watchdog extension, Postmaster starts two processes as background workers at database instance startup.

One process that is called Watchdog is for heartbeat monitoring, and the second process that is called Terminator is to forcibly terminate client SQL connections when the Watchdog process detects a failure on heartbeat monitoring.



Client Server
conmgr process
Return attribute
Connect for heartbeat
Send packet for heartbeat
Watchdog Process
Creates at instance starting up.
Request to terminate a connection on heartbeat monitoring time out.
Postmaster Process
Terminator Process
Instance 1
Return attribute
Connect for heartbeat
Send packet for heartbeat
Watchdog Process
Creates at instance starting up.
Request to terminate a connection on heartbeat monitoring time out.
Postmaster Process
Terminator Process
Instance 2
Set of Instances configured for replication

Figure 7-1 Connection Manager communication flow diagram

To demonstrate the functions of Connection Manager, we use the following system setup:

- Two FUJITSU Enterprise Postgres instances with Streaming Replication (Primary + Standby).
- Application that uses the C Library (libpq).

Setting up Connection Manager on the database server

Complete the following steps to set up Connection Manager on the database server:

1. Connect to the primary database server.
2. Switch to the FUJITSU Enterprise Postgres user.
3. Configure the Connection Manager parameters in the `postgresql.conf` configuration file. Query the database to find the `postgresql.conf` configuration file, as shown in Example 7-27.

Example 7-27 Checking the path of the postgresql.conf configuration file

```
[root@rdbkpgr3 ~]# su - fsepuser
Last login: Tue Nov 10 01:40:07 EST 2020 on pts/4
[fsepuser@rdbkpgr3 ~]$ psql -d postgres -c 'SHOW config_file'
LOG:  statement: SHOW config_file (11077)
           config_file
--------------------------------
 /database/inst1/postgresql.conf
(One row)
```

4. Add the Connection Manager parameters, as shown in Example 7-28.

Example 7-28 Adding the port number and preinstalled libraries to the postgresql.conf file

```
shared_preload_libraries = 'watchdog'
watchdog.port = 27545
```

To check the list of Connection Manager configuration parameters, see *FUJITSU Enterprise Postgres 12 on IBM LinuxONE*.

5. Restart the database instance. The database instance must be restarted for the parameter change to take effect (Example 7-29).

Example 7-29 Restarting the database instance

```
[fsepuser@rdbkpgr3 ~]$ pg_ctl restart -D /database/inst1
waiting for server to shut down.... done
server stopped
waiting for server to start....2020-11-10 01:52:04.302 EST [82573] LOG:  starting
PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (GCC) 8.3.1 20190507 (Red
Hat 8.3.1-4), 64-bit
2020-11-10 01:52:04.302 EST [82573] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2020-11-10 01:52:04.302 EST [82573] LOG:  listening on IPv6 address "::", port
27500
2020-11-10 01:52:04.304 EST [82573] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2020-11-10 01:52:04.309 EST [82573] LOG:  redirecting log output to logging
collector process
2020-11-10 01:52:04.309 EST [82573] HINT:  Future log output will appear in
directory "pg_log".
 done
server started
```

6. Connect to the postgres instance by using **psql**, and install the Watchdog extension, as shown in Example 7-30.

Example 7-30 Installing the Watchdog extension

```
[fsepuser@rdbkpgr3 ~]$ psql -p 27500 -d postgres
psql (12.1)
Type "help" for help.

postgres=# CREATE EXTENSION watchdog;
CREATE EXTENSION
postgres=# \dx
                   List of installed extensions
   Name   | Version |   Schema   |          Description
----------+---------+------------+--------------------------------
 plpgsql  | 1.0     | pg_catalog | PL/pgSQL procedural language
 watchdog | 1.0     | public     | watchdog for Connection Manager
(Two rows)
```

7. Connect to the standby server, and repeat steps 2 on page 265 - 5 on page 265. There is no need to run step 6. As part of replication, the Watchdog extension is installed on the standby server (Example 7-31).

Example 7-31 Adding the Connection Manager configuration on the standby server

```
[fsepuser@rdbkpgr3 ~]$ psql -d postgres –p 27501 -c 'SHOW config_file'
LOG:  statement: SHOW config_file (11077)
           config_file
---------------------------------
 /database/inst1/postgresql.conf
(One row)

//Add to postgresql.conf configuration file
shared_preload_libraries = 'watchdog'
watchdog.port = 27547

//Restart the standby server.
[fsepuser@rdbkpgr3 ~]$ pg_ctl restart -D /database/inst2

//Check installed extensions on standby server.
[fsepuser@rdbkpgr3 ~]$ psql -p 27501 -d postgres
psql (12.1)
Type "help" for help.

postgres=# \dx
                   List of installed extensions
   Name   | Version |   Schema   |          Description
----------+---------+------------+--------------------------------
 plpgsql  | 1.0     | pg_catalog | PL/pgSQL procedural language
 watchdog | 1.0     | public     | watchdog for Connection Manager
(Two rows)
```

Setting up Connection Manager on a client/server

Complete the following steps to set up Connection Manager on a client/server:

1. Connect to the client/server.
2. Switch to the FUJITSU Enterprise Postgres user.
3. Create a dedicated directory with read, write, and execute permissions for the user who starts the Connection Manager process.
4. Change to the newly created directory and create a Connection Manager configuration file (Example 7-32).

Example 7-32 Connection Manager setting on the client/server

```
// Create a conmgr directory
[fsepclient@rdbkpgr3 ~]$ mkdir conmgr
[fsepclient@rdbkpgr3 ~]$ ls -l
total 0
drwxrwxr-x. 2 fsepclient fsepclient 6 Oct 23 09:14 conmgr

//Create a configuration file conmgr.conf (backend_host* can be
//backend_hostaddr*):
[fsepclient@rdbkpgr3 conmgr]$ vi conmgr.conf
port = 27546
backend_host0 = 'xxx.xx.xx.158'
backend_host1 = 'xxx.xx.xx.158'
backend_port0 = 27500
backend_port1 = 27501
watchdog_port0 = 27545
watchdog_port1 = 27547
```

   To monitor more than two instances, add instance details in the `conmgr.conf` configuration file.

   To check the list of Connection Manager configuration parameters, see *FUJITSU Enterprise Postgres 12 on IBM LinuxONE*.

5. Start the Connection Manager process by using the **`cm_ctl`** command, as shown in Example 7-33.

Example 7-33 Starting the Connection Manager process on the client

```
[fsepclient@rdbkpgr3 conmgr]$ cm_ctl start -D /home/fsepclient/conmgr
2020-10-23 09:22:12.742 LOG:  conmgr process is starting (25140)
2020-10-23 09:22:12.743 LOG:  listening on IPv4 port 27546 (25149)
2020-10-23 09:22:12.743 LOG:  listening on IPv6 port 27546 (25149)
2020-10-23 09:22:12.744 LOG:  heartbeat connection is established (25157):
host=xxx.xx.xx.158 port=27545 pid=3932
2020-10-23 09:22:12.744 LOG:  heartbeat connection is established (25157):
host=xxx.xx.xx.158 port=27547 pid=3943
2020-10-23 09:22:12.744 LOG:  attribute of the instance(host='xxx.xx.xx.158',
port=27545) is changed from 'unknown' to 'primary' (25203)
2020-10-23 09:22:12.744 LOG:  attribute of the instance(host='xxx.xx.xx.158',
port=27547) is changed from 'unknown' to 'standby' (25203)
cm_ctl: conmgr process is ready (25077)
cm_ctl: waiting conmgr process to connect to watchdog (25070)
cm_ctl: started conmgr process successfully (25078)
```

In this setup, while running the **cmd_ctl** command, we specify the conmgr directory path with the **-D** option. Alternatively, the directory path can be set as the environment variable **CMDATA**.

6. Connect to the database server and check the details of the Connection Manager process, which is connected to the Watchdog process, as shown in Example 7-34.

Example 7-34 Verifying the Connection Manager setting on the database server

```
[fsepuser@rdbkpgr3 ~]$ psql -p 27500 -d postgres -c "SELECT * FROM
pgx_stat_watchdog"
  conmgr_addr  | conmgr_port | heartbeat_interval | heartbeat_timeout
---------------+-------------+--------------------+-------------------
 xxx.xx.xx.158 |       45026 |                 10 |                20
(One row)

[fsepuser@rdbkpgr3 ~]$ psql -p 27501 -d postgres -c "SELECT * FROM
pgx_stat_watchdog"
  conmgr_addr  | conmgr_port | heartbeat_interval | heartbeat_timeout
---------------+-------------+--------------------+-------------------
 xxx.xx.xx.158 |       43446 |                 10 |                20
(One row)
```

7.2.2 Using Connection Manager from an application

An application connects to the Connection Manager process as it originally connects to the database instance. It is the client driver that automatically determines whether the connection is to an instance or a **conmgr** process.

In FUJITSU Enterprise Postgres V12, the available client drivers for Connection Manager are:

- libpq (C language library)
- ECPG (Embedded SQL in C)

An application connection to a database instance by using Connection Manager is a two-step process, as shown in Figure 7-2 on page 269.

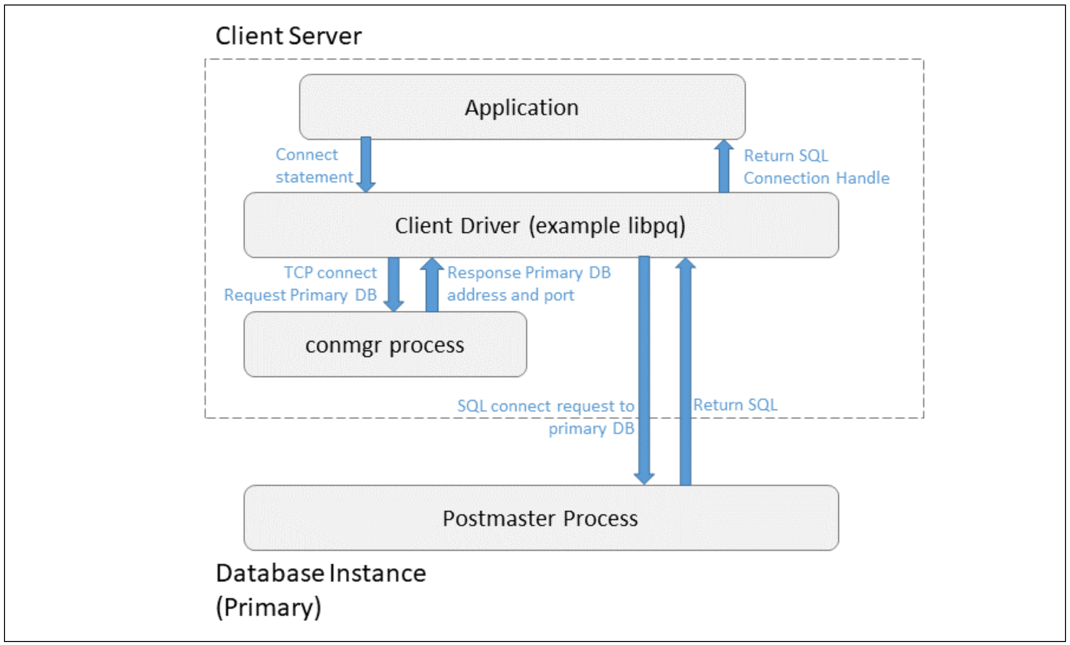

Client Server
Application
Connect statement
Return SQL Connection Handle
Client Driver (example libpq)
TCP connect Request Primary DB
Response Primary DB address and port
conmgr process
SQL connect request to primary DB
Return SQL
Postmaster Process
Database Instance (Primary)

Figure 7-2 Transparent connection flow between application and database

To demonstrate the application and database connectively, we use the `libpq` client driver. The sample code to build the C application is taken from 33.21, "Example Programs", in the *PostgreSQL 12.7 Documentation*. Example 7-35 demonstrates how to verify the client driver.

Example 7-35 Verifying the Connection Manager setting on the database server

```
[fsepclient@rdbkpgr3 ~]$ ls -l
total 12
drwxrwxr-x. 2 fsepclient fsepclient   43 Oct 27 01:20 conmgr
-rw-r--r--. 1 root       root       3571 Oct 27 01:29 testlibpq.c
-rw-r--r--. 1 root       root       1968 Oct 27 01:29 testlibpq_insert.c
-rw-r--r--. 1 root       root       2968 Oct 27 01:29 testlibpq_retry.c
```

For more information about the FUJITSU Enterprise Postgres libpq client driver, see Chapter 4, "C Library (libpq)", of the *FUJITSU Enterprise Postgres 12 on IBM LinuxONE Application Development Guide*.

The sample code is compiled by linking all the dependent libraries. There are two steps to validate the Connection Manager settings:

1. Connect to the application server.
2. Run the **`cm_ctl`** status command to check the Connection Manager, database instances, and application status (Example 7-36).

Example 7-36 Verifying the Connection Manager setting on the database server

```
[fsepclient@rdbkpgr3 ~]$ cm_ctl status -i all -D /home/fsepclient/conmgr
conmgr_status:
status           pid
ready            3705

instance_information:
addr                                    port  database_attr
xxx.xx.xx.158                           27501 standby
xxx.xx.xx.158                        27500 primary

application_information
addr                                 port  connected_time
```

7.2.3 Connection Manager use case examples

In this section, we demonstrate several Connection Manager use cases.

Use case 1: Both database instances are running

Here are the application connection parameters that we used for this use case:

- **`Hostname=localhost`**: The Connection Manager that is running on the client.
- **`port=27456`**: The Connection Manager port number that is configured in the `conmgr.conf` configuration file.
- **`target_session_attrs=read-write`**

By using the sample code in 7.1.3, “C library (libpq)” on page 256, we tested an application connection to the database that uses the Connection Manager (Example 7-37).

The Connection Manager shares the IP address and port of the primary instance because **`target_session_attrs=read-write`**.

Example 7-37 Application connects with target_session_attrs as read-write

```
[fsepclient@rdbkpgr3 ~]$ ./testprog "host=localhost port=27546 dbname=postgres
user=fsepuser password=fsepuser connect_timeout=10
target_session_attrs=read-write"
oid            datname        datdba         encoding       datcollate
datctype       datistemplate  datallowconn   datconnlimit   datlastsysoid
datfrozenxid   datminmxid     dattablespace  datacl

13468         postgres      10           6           C            C
f             t            -1           13467        490          1
1663
1             template1     10           6           C            C
t             t            -1           13467        490          1
1663            {=c/fsepuser,fsepuser=CTc/fsepuser}
13467         template0     10           6           C            C
t             f            -1           13467        490          1
1663            {=c/fsepuser,fsepuser=CTc/fsepuser}
16406         FEPRedbook    10           6           C            C
f             t            -1           13467        490          1
1663
```

Example 7-38 shows the server logs at the primary instance that is named FEP1, indicating that the application connected to it.

Example 7-38 Application connects to the primary instance (FEP1)

```
Nov  2 22:31:34 rdbkpgr3 FEP1[30016]: [8-1] 2020-11-02 22:31:34.560 EST [30016]
LOG:  statement: SELECT pg_catalog.set_config('search_path', '', false) (11077)
Nov  2 22:31:34 rdbkpgr3 FEP1[30016]: [9-1] 2020-11-02 22:31:34.560 EST [30016]
LOG:  statement: BEGIN (11077)
```

```
Nov  2 22:31:34 rdbkpgr3 FEP1[30016]: [10-1] 2020-11-02 22:31:34.560 EST [30016]
LOG:  statement: DECLARE myportal CURSOR FOR select * from pg_database (11077)
Nov  2 22:31:34 rdbkpgr3 FEP1[30016]: [11-1] 2020-11-02 22:31:34.561 EST [30016]
LOG:  statement: FETCH ALL in myportal (11077)
Nov  2 22:31:34 rdbkpgr3 FEP1[30016]: [12-1] 2020-11-02 22:31:34.561 EST [30016]
LOG:  statement: CLOSE myportal (11077)
Nov  2 22:31:34 rdbkpgr3 FEP1[30016]: [13-1] 2020-11-02 22:31:34.561 EST [30016]
LOG:  statement: END (11077)
```

Connection Manager shares the IP address and port of the standby instance if **target_session_attrs** is prefer-read. By using the following application connection parameters, we demonstrate in Example 7-39 how an application connects when **target_session_attrs** is changed to prefer-read:

- **Hostname=localhost**: Connection Manager is running on the client/server.
- **port=27456**: The Connection Manager port number is configured in the conmgr.conf configuration file.
- **target_session_attrs=prefer-read**

Example 7-39 Application connects with target_session_attrs as prefer-read

```
[fsepclient@rdbkpgr3 ~]$ ./testprog "host=localhost port=27546 dbname=postgres
user=fsepuser password=fsepuser connect_timeout=10
target_session_attrs=prefer-read"
oid           datname       datdba        encoding      datcollate
datctype      datistemplate datallowconn  datconnlimit  datlastsysoid
datfrozenxid  datminmxid    dattablespace datacl

13468        postgres      10           6           C            C
f            t            -1           13467        490          1
1663
1            template1     10           6           C            C
t            t            -1           13467        490          1
1663          {=c/fsepuser,fsepuser=CTc/fsepuser}
13467        template0     10           6           C            C
t            f            -1           13467        490          1
1663          {=c/fsepuser,fsepuser=CTc/fsepuser}
16406        FEPRedbook    10           6           C            C
f            t            -1           13467        490          1
1663
```

Example 7-40 shows the server logs at the primary instance that is named FEP2, which indicate that the application was connected to it.

Example 7-40 Application connects to the standby instance (FEP2)

```
Nov  2 22:20:27 rdbkpgr3 FEP2[29882]: [8-1] 2020-11-02 22:20:27.493 EST [29882]
LOG:  statement: SELECT pg_catalog.set_config('search_path', '', false) (11077)
Nov  2 22:20:27 rdbkpgr3 FEP2[29882]: [9-1] 2020-11-02 22:20:27.493 EST [29882]
LOG:  statement: BEGIN (11077)
Nov  2 22:20:27 rdbkpgr3 FEP2[29882]: [10-1] 2020-11-02 22:20:27.493 EST [29882]
LOG:  statement: DECLARE myportal CURSOR FOR select * from pg_database (11077)
Nov  2 22:20:27 rdbkpgr3 FEP2[29882]: [11-1] 2020-11-02 22:20:27.493 EST [29882]
LOG:  statement: FETCH ALL in myportal (11077)
Nov  2 22:20:27 rdbkpgr3 FEP2[29882]: [12-1] 2020-11-02 22:20:27.494 EST [29882]
LOG:  statement: CLOSE myportal (11077)
```

```
Nov  2 22:20:27 rdbkpgr3 FEP2[29882]: [13-1] 2020-11-02 22:20:27.494 EST [29882]
LOG:  statement: END (11077)
```

Use case 2: Database primary instance is down, and the application makes a connection request

In this use case, we demonstrate what happens when the database primary instance is down and an application makes a connection request.

To demonstrate this use case, we performed the following steps and used the sample code in 7.1.3, “C library (libpq)” on page 256:

1. Connect to the primary instance and run the **immediate** shutdown command so that primary database exits immediately (Example 7-41).

Example 7-41 Terminating the database primary instance

```
[fsepuser@rdbkpgr3 ~]$ pg_ctl -w -m immediate stop -D /database/inst1
waiting for server to shut down.... done
server stopped
```

2. Check the status of Connection Manager and database instances. The status of the primary database instance changes from primary to standby, as shown in Example 7-42.

Example 7-42 Verifying the Connection Manager status on the database server

```
[fsepclient@rdbkpgr3 ~]$ cm_ctl status -i all -D /home/fsepclient/conmgr
conmgr_status:
status          pid
ready           3705

instance_information:
addr                                    port  database_attr
xxx.xx.xx.158                           27501 standby
xxx.xx.xx.158                           27500 unknown

application_information
addr                                    port  connected_time
```

3. Run the application with **target_session_attrs** as `read-write`, as shown in Example 7-43.

Example 7-43 Application connects with target_session_attrs as read-write

```
[fsepclient@rdbkpgr3 ~]$ ./testprog_retry "host=localhost port=27546
dbname=postgres user=fsepuser password=fsepuser connect_timeout=10
target_session_attrs=read-write"
Connection to database failed: no target server address from Connection Manager
[fsepclient@rdbkpgr3 ~]$
```

4. While the primary instance is not started, run the application with **target_session_attrs** as `prefer-read`. An application that has a read request connects to the standby instance, FEP2, and displays the results.

   Because the client connection request was to connect to a standby database instance (through prefer-read), the Connection Manager finds that FEP2, the standby database instance, can serve such a connection request, and so serves the connection details to the client.

Based on the **target_session_attrs** parameter, Connection Manager shares the IP address and port number of the database instance. The connection to the database instance is transparent to the application, and it always connects to the Connection Manager. Example 7-44 shows how to verify the Connection Manager setting on the database server.

Example 7-44 Application connects with target_session_attrs as prefer-read

```
[fsepclient@rdbkpgr3 ~]$ ./testprog "host=localhost port=27546 dbname=postgres
user=fsepuser password=fsepuser connect_timeout=10
target_session_attrs=prefer-read"
oid             datname         datdba          encoding        datcollate
datctype        datistemplate   datallowconn    datconnlimit    datlastsysoid
datfrozenxid    datminmxid      dattablespace   datacl

13468          postgres       10             6             C             C
f              t             -1             13467         490           1
1663
1              template1      10             6             C             C
t              t             -1             13467         490           1
1663             {=c/fsepuser,fsepuser=CTc/fsepuser}
13467          template0      10             6             C             C
t              f             -1             13467         490           1
1663             {=c/fsepuser,fsepuser=CTc/fsepuser}
16406          FEPRedbook     10             6             C             C
f              t             -1             13467         490           1
1663
```

5. Promote the standby server to primary by using the command that is shown in Example 7-45.

Example 7-45 Promoting the standby server

```
[fsepuser@rdbkpgr3 ~]$ pg_ctl promote -D /database/inst2
waiting for server to promote.... done
server promoted
```

Note: Connection Manager provides a heartbeat monitoring and transparent connection support feature. It does not include automatic switching from primary to standby. For automatic switchover from primary to standby in a failover, use the MC.

For more information about FUJITSU Enterprise Postgres MC, see Chapter 5, “Managing Mirroring Controller Using WebAdmin, in the *FUJITSU Enterprise Postgres 12 on IBM LinuxONE Cluster Operations Guide*.

6. Check the status of the Connection Manager and database instances. The status of the previous standby database instance (FEP2) changed from standby to primary. We have promoted only the previous standby (FEP2) to the primary role. However, the FEP1 instance is still down.

7. Run the application with **`target_session_attrs`** as `read-write` (Example 7-46). The application connects to the promoted database instance, as indicated by a successful insert.

Example 7-46 Rerunning the application

```
[fsepclient@rdbkpgr3 ~]$ ./testprog_retry "host=localhost port=27546 dbname=postgres
user=fsepuser password=fsepuser connect_timeout=10 target_session_attrs=read-write"
conninfo:host=localhost port=27546 dbname=postgres user=fsepuser password=fsepuser
connect_timeout=10 target_session_attrs=read-write
Wait 10sec. ##########
Insert successful
Exit successfully
```

8. Check the status of the Connection Manager and database instances (Example 7-47). The status of the standby database instance changed from standby to primary.

Example 7-47 Verifying the Connection Manager setting on the database server

```
[fsepclient@rdbkpgr3 ~]$ cm_ctl status -i all -D /home/fsepclient/conmgr
conmgr_status:
status          pid
ready           3705

instance_information:
addr                                    port  database_attr
xxx.xx.xx.158                           27501 primary
xxx.xx.xx.158                           27500 unknown

application_information
addr                                    port  connected_time
```

Use case 3: Application makes a connection request, and the primary instance goes down while the application is connected

In this use case, we demonstrate what happens when an application makes a connection request and the primary instance goes down before the application completes.

To demonstrate this use case, we performed the following steps and used the sample code that was created in 7.1.3, “C library (libpq)” on page 256.

Here are the application connection parameters that we used for this use case:

- **`Hostname = localhost`**: The Connection Manager is running on the client/server.
- **`port = 27456`**: The Connection Manager port number is configured in the `conmgr.conf` configuration file.
- **`target_session_attrs = read-write`**

1. The application connects to the database by using the Connection Manager, as shown in Example 7-48.

Example 7-48 Application connection to available instance through the Connection Manager

```
[fsepclient@rdbkpgr3 ~]$ ./testprog_retry "host=localhost port=27546 dbname=postgres
user=fsepuser password=fsepuser connect_timeout=10 target_session_attrs=read-write"
conninfo:host=localhost port=27546 dbname=postgres user=fsepuser password=fsepuser
connect_timeout=10 target_session_attrs=read-write
Wait 10sec. ##
```

2. The application waits for 10 seconds. During that wait time, the primary instance goes down. To demonstrate this situation, we run a command to terminate the database primary instance, as shown in Example 7-49.

Example 7-49 Terminating the database primary instance

```
[fsepuser@rdbkpgr3 ~]$ pg_ctl -w -m immediate stop -D /database/inst1
waiting for server to shut down.... done
server stopped
```

3. The application insert operation fails because the primary instance is down. We find this information in the Connection Manager and application logs (Example 7-50).

Example 7-50 Checking the logs of Connection Manager and the client application

```
// Connection Manager logs

2020-10-27 04:25:49.605 ERROR:  remote node shuts down heartbeat connection
(25160): host=xxx.xx.xx.158 port=27545 pid=5379
2020-10-27 04:25:49.605 LOG:  attribute of the instance(host='xxx.xx.xx.158',
port=27545) is changed from 'primary' to 'unknown' (25203)
#2020-10-27 04:25:50.606 ERROR:  could not connect due to the following errors
(25143): host=xxx.xx.xx.158 port=27545 pid=5379
2020-10-27 04:25:50.606 ERROR:  Connection refused: host=xxx.xx.xx.158 port=27545
pid=5379

// Application logs

FATAL:  heartbeat connection is shut down eventually.
INSERT failed: retry connection(2)
```

4. The application retries a connection to the database. Meanwhile, we promote the standby instance by using the command that is shown in Example 7-51.

Example 7-51 Promoting the standby server

```
[fsepuser@rdbkpgr3 ~]$ pg_ctl promote -D /database/inst2
waiting for server to promote.... done
server promoted
```

The application connects to the promoted database instance, as shown in Example 7-52.

Example 7-52 Application connects to the promoted instance

```
conninfo:host=localhost port=27546 dbname=postgres user=fsepuser password=fsepuser
connect_timeout=10 target_session_attrs=read-write
Wait 10sec. ##########
Insert successful
Exit successfully
```

7.3 Oracle compatibility features

In this section, we describe Oracle compatibility features that are included in FUJITSU Enterprise Postgres, which enables developers to easily migrate from Oracle to FUJITSU Enterprise Postgres.

There are two options regarding the Oracle compatibility function:

- Use the functions that are compatible with Oracle databases.
- Use the function, orafce, a FUJITSU Enterprise Postgres original feature.

7.3.1 Original Oracle compatibility functions

FUJITSU Enterprise Postgres provides features that are compatible with Oracle databases. These features enable you to easily migrate to FUJITSU Enterprise Postgres and reduce the costs of reconfiguring applications. These features include SQL statements, functions, and packages, which are shown in Table 7-4 and developed by FUJITSU.

Table 7-4 Oracle compatibility functions and features

Functions	Oracle compatibility feature
SQL	outerjoin operator(+)
	Dual table
Built-In Functions	nvl, decode, and substring
PL/SQL packages	**`DBMS_OUTPUT`**
	`UTL_FILE`
	`DBMS_SQL`

To use the features that are compatible with Oracle databases, in the following steps we demonstrate the setup and execution of the FUJITSU compatibility feature for Oracle databases:

1. To use the compatibility feature for Oracle databases, run the command **`CREATE EXTENSION`**, as shown in Example 7-53.

Example 7-53 How to set up the Oracle compatibility function

```
fsepuser@rdbkpgr3 ~]$ psql -d postgres -p 27500
psql (12.1)fs
Type "help" for help.

postgres=# create extension oracle_compatible;
CREATE EXTENSION
postgres=# \dx
```

2. In the file `postgresql.conf`, set the **`search_path`** parameter to `oracle` and `pg_catalog`, as shown in Example 7-54. You must specify `oracle` before `pg_catalog`.

Example 7-54 Setting search_path

```
search_path = '"$user", public, oracle, pg_catalog'
```

3. To demonstrate the features, we create table 1 (`tbl1`) for test data. Our test data is shown in Example 7-55. We verify that the data was inserted with the **`select * from tbl1;`** command.

Example 7-55 Testing data

```
postgres=# create table tbl1(id int, name text, sid int, tid int);
CREATE TABLE
postgres=# insert into tbl1 values(1, 'abc', 3, 8);
INSERT 0 1
postgres=# insert into tbl1 values(2, 'def', 2, 5);
INSERT 0 1
postgres=# insert into tbl1 values(3, 'ghi', 4, 6);
INSERT 0 1
postgres=# insert into tbl1 values(4, 'jkl', 1, 2);
INSERT 0 1
postgres=# insert into tbl1 values(5, null, 0, 0);
INSERT 0 1
postgres=# insert into tbl1 values(6, 'mno', null, null);
INSERT 0 1
postgres=# insert into tbl1 values(7, '', 2, 2);
INSERT 0 1
postgres=# select * from tbl1;
 id | name | sid | tid
----+------+-----+-----
  1 | abc  |   3 |   8
  2 | def  |   2 |   5
  3 | ghi  |   4 |   6
  4 | jkl  |   1 |   2
  5 |      |   0 |   0#name is NULL
  6 | mno  |     |#sid,tid are NULL
  7 |      |   2 |   2#name is ''?string of 0 length?
(Seven rows)
```

Notes on using the features that are compatible with Oracle databases

In this section, we provide some notes and hints when using the features that are compatible with Oracle databases.

SUBSTR

`SUBSTR` extracts part of a character string. **`SUBSTR`** returns the number of characters that are specified in the third argument (starting from the position that is specified in the second argument) from the string that is specified in the first argument. So, when using **`SUBSTR`**, specify the string as the first parameter, the starting position in the second parameter, and the length of the string that is returned in the third parameter, as shown in Example 7-56.

Example 7-56 Running SUBSTR

```
postgres=# select substr('abcdefg', 2);
 substr
--------
 bcdefg
(One row)

postgres=# select substr('abcdefg', 2,3);
 substr
--------
```

```
 bcd
(One row)

postgres=# select substr('abcdefg', 0,2);
 substr
--------
 ab
(One row)

postgres=# select substr('abcdefg', -4,2);
 substr
--------
 de
(One row)

postgres=# select substr('abcdefg', -4,0);
 substr
--------

(One row)
```

DECODE

`DECODE` provides the function that compares values. If they match, `DECODE` returns a corresponding value.

In this section, we provide three examples:

1. Example 7-57 shows using the **`if else`** statement.

Example 7-57 Execution of DECODE: if else statement

```
postgres=# select id, decode(id, 1, 'MAN', 2, 'WOMAN', 'UNKNOWN') "gender" from
tbl1;
id | gender
----+---------
  1 | MAN
  2 | WOMAN
  3 | UNKNOWN
  4 | UNKNOWN
  5 | UNKNOWN
  6 | UNKNOWN
  7 | UNKNOWN
(Seven rows)
```

2. Example 7-58 shows the replacement of null values.

Example 7-58 Execution of DECODE: Replacement of null values

```
postgres=# select sid, decode(sid, null, 'NONE', 1, 'OK', 2, 'OK', 'NG') "check"
from tbl1;
LOG:  statement: select sid, decode(sid, null, 'NONE', 1, 'OK', 2, 'OK', 'NG')
"check" from tbl1; (11077)
 sid | check
-----+-------
   3 | NG
   2 | OK
   4 | NG
   1 | OK
   0 | NG
     | NONE
   2 | OK
(Seven rows)
```

3. Example 7-59 shows changing the order of records.

Example 7-59 Execution of DECODE: Changing the order of records

```
postgres=# select * from tbl1 order by decode(name, null, 0, 'jkl', 1, 'mno', 2,
'abc', 3, 4) ASC;
LOG:  statement: select * from tbl1 order by decode(name, null, 0, 'jkl', 1,
'mno', 2, 'abc', 3, 4) ASC; (11077)
 id | name | sid | tid
----+------+-----+-----
  5 |      |   0 |   0
  4 | jkl  |   1 |   2
  6 | mno  |     |
  1 | abc  |   3 |   8
  3 | ghi  |   4 |   6
  2 | def  |   2 |   5
  7 |      |   2 |   2
(Seven rows)
```

7.3.2 Oracle compatible functions that use orafce

FUJITSU Enterprise Postgres supports orafce 3.8.0 to provide compatibility with Oracle databases.

To use these features, in the file `postgresql.conf`, set the **search_path** parameter to `oracle` and `pg_catalog`, as shown in Example 7-60. You must specify `oracle` before `pg_catalog`.

Example 7-60 Setting search_path

```
search_path = '"$user", public, oracle, pg_catalog'
```

Note: As a best practice, set the parameter **search_path** in the `postgresql.conf` file to make it effective for each instance. For more information, see orafce.

Functions included with orafce

This section demonstrates **SYSDATE**, **TO_DATE**, and **NVL2**, which are included in orafce.

SYSDATE

This function returns the current date and time of the operating system (OS). There are two things to remember about this feature:

- The default time zone of **SYSDATE()** is Greenwich mean time.

  The time zone can be changed by setting the parameter `orafce.timezone` in `postgreql.conf`.
- FUJITSU Enterprise Postgres cannot use seconds after the decimal point.

Example 7-61 shows the commands that are used to verify the PostgreSQL timestamp and date before running the **SYSDATE** function.

Example 7-61 Using SYSDATE

```
//Verify the type of timestamp of PostgreSQL
postgres=# select clock_timestamp();
        clock_timestamp
-------------------------------
 2020-11-13 03:35:28.370585-05
(One row)

//Verify oracle.date of PostgreSQL
postgres=# select cast(clock_timestamp() as oracle.date);
   clock_timestamp
---------------------
 2020-11-13 03:35:42
(One row)

//Execute SYSDATE function
postgres=# select oracle.sysdate();
       sysdate
---------------------
 2020-11-13 08:39:45
(One row)
```

TO_DATE

The **TO_DATE** function is used in Oracle to convert a string to a date. A standard feature of PostgreSQL is that the data type of the return value is `DATE`. By using orafce, the data type of the return value is `TIMESTAMP`.

There are two things to remember about this feature:

- Although Oracle DB can run addition and subtraction of fractions, FUJITSU Enterprise Postgres must cast to numeric type or use the **INTERVAL** function.
- FUJITSU Enterprise Postgres cannot use seconds after the decimal point.

Example 7-62 shows various uses of the **TO_DATE** function.

Example 7-62 Using TO_DATE

```
//execution of to_date
postgres=# select to_date('2020/11/11');
       to_date
---------------------
 2020-11-11 00:00:00
(One row)

//add 1 hour by using the numeric type
postgres=# select to_char(to_date('2020/11/11', 'yyyy/mm/dd') + 1.0/24,
'yyyy/mm/dd hh24:mi:ss');
       to_char
---------------------
 2020/11/11 01:00:00
(One row)
```

```
//add 1 hour by using the INTERVAL function
postgres=# select to_char(oracle.to_date('2020/11/11', 'yyyy/mm/dd') + interval '1 
hour', 'yyyy/mm/dd hh24:mi:ss');
       to_char
---------------------
 2020/11/11 01:00:00
(One row)
```

NVL

This function allows the return of a substitute value based on whether a value is `NULL` or not `NULL`. **`NVL`** substitutes a `NULL` value with a more meaningful alternative in the results of a query. The syntax of the **`NVL()`** function is:

```
NVL(e1,e2)
```

If **`e1`** evaluates to `NULL`, then the function returns **`e2`**.

There are two things to remember about this feature:

1. The first and second parameters must be same data type.
2. A zero length string in the first parameter must be changed to `NULL`.

Example 7-63 demonstrates the use of the **`NVL`** function.

Example 7-63 Examples of the use of NVL

```
// The result from this query is not expected. We expect the result to be x, but 
// the second NULL value returned a blank.
postgres=# select name, nvl(name, 'x') from tbl1;
 name | nvl
------+-----
 abc  | abc
 def  | def
 ghi  | ghi
 jkl  | jkl
      | x
 mno  | mno
      |
(Seven rows)

// This query was used to get the expected result.
postgres=# select name, nvl(nullif(name, ''), 'x') from tbl1;
 name | nvl
------+-----
 abc  | abc
 def  | def
 ghi  | ghi
 jkl  | jkl
      | x
 mno  | mno
      | x
(Seven rows)

// There is no special rule for number.
postgres=# select sid, nvl(sid, 999) from tbl1;
 sid | nvl
```

```
-----+-----
   3 |   3
   2 |   2
   4 |   4
   1 |   1
   0 |   0
     | 999
   2 |   2
(Seven rows)

// This error occurs when the data type of the first and second parameters are
// different.
postgres=# select name, nvl(name, 0) from tbl1;
ERROR:  function nvl(text, integer) does not exist (10494)
LINE 1: select name, nvl(name, 0) from tbl1;
                     ^
HINT:  No function matches the given name and argument types. You might need to
add explicit type casts. (14431)

// solution for when the data type of the first and second parameters are
// different.
postgres=# select name, nvl(nullif(name, ''), '0') from tbl1;
 name | nvl
------+-----
 abc  | abc
 def  | def
 ghi  | ghi
 jkl  | jkl
      | 0
 mno  | mno
      | 0
(Seven rows)
```

7.4 Migrating to FUJITSU Enterprise Postgres

In this section, we describe migration to FUJITSU Enterprise Postgres, the FUJITSU migration tool, and the FUJITSU Enterprise Postgres backup tools that support data migration.

We consider the following migration scenarios:

Database	Migration from open source software (OSS) PostgreSQL to FUJITSU Enterprise Postgres.
Platform	Migration from IA(x86) to IBM Z.

In our lab environment, we are using PostgreSQL 12 and OSS PostgreSQL.

7.4.1 Migrating from OSS PostgreSQL to FUJITSU Enterprise Postgres

OSS PostgreSQL can be migrated to FUJITSU Enterprise Postgres in either an offline or online mode. In an offline migration, the database instance is stopped and a consistent data and schema backup is taken, and in an online migration, PostgreSQL logical replication is used to migrate the changes from OSS PostgreSQL to FUJITSU Enterprise Postgres.

Offline migration is carried out during a migration window and the database is not available for write operations. In an offline migration, the definition of each selected object is read and written in SQL script. This SQL script is used to migrate the Postgres database to FUJITSU Enterprise Postgres.

The total time of offline migration depends on the database size and time that is taken to lift and shift the selected objects. The downtime in the case of an offline backup significantly increases when migrating large databases.

Performing an offline migration from OSS PostgreSQL to FUJITSU Enterprise Postgres

To perform an offline migration, complete the following steps:

1. Run database instance in read-only mode. Change the `postgresql.conf` file parameter **`default_transaction_read_only`** to `true`, as shown in Example 7-64, and reload the configuration.

Example 7-64 Updating the postgresql.conf configuration file

```
// Connection String parameters IPv4 and targetServerType “master”.
[postgres@rdbkpgr3 ~]$ psql -c 'SHOW config_file'
             config_file
------------------------------------
 /var/lib/pgsql/data/postgresql.conf
(One row)

[postgres@rdbkpgr3 ~]$ vi /var/lib/pgsql/data/postgresql.conf

default_transaction_read_only = on
```

2. Take a logical backup by using the PostgreSQL utility **`pg_dump`** to back up a single database, or use **`pg_dumpall`** to back up global objects that are common to the database cluster, as shown in Example 7-65.

Example 7-65 Backing up a single database by using the pg_dumpall command

```
[postgres@rdbkpgr3 ~]$ pg_dumpall > backup_1211.out
```

3. Move the backup file to the target server (Example 7-66).

Example 7-66 Copying the backup file to the FUJITSU Enterprise Postgres instance

```
[postgres@rdbkpgr3 ~]$ scp backup_1211.out fsepuser@xxx.xx.xx.164:/home/fsepuser
Password:
backup_1211.out
```

4. Restore the backup to the FUJITSU Enterprise Postgres instance (Example 7-67).

Example 7-67 Restoring the backup on the FUJITSU Enterprise Postgres instance

```
fsepuser@rdbkpgs3:~> psql -f backup_1211.out postgres
SET
SET
SET
CREATE ROLE
ALTER ROLE
You are now connected to database "template1" as user "fsepuser".
SET
```

```
SET
SET
SET
SET
 set_config
------------

(One row)

SET
SET
SET
SET
You are now connected to database "postgres" as user "fsepuser".
SET
SET
SET
SET
SET
 set_config
------------

(One row)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
COPY 10000001
COPY 100
COPY 0
COPY 1000
```

Performing an online migration from OSS PostgreSQL to FUJITSU Enterprise Postgres

Online migration is used when it is not acceptable to have any system downtime. In this example, we use Postgres logical replication to migrate to FUJITSU Enterprise Postgres. Logical replication is a method of replicating data objects and their changes based on their replication identity (usually a primary key). Logical replication works on the publisher-subscriber model, where the master server is defined as publisher and the destination server as subscriber.

In this publication, we replicate all the tables, so while creating a publication, we choose `ALL TABLES`. You can also create a publication for selected tables when you are planning to migrate a selected set of tables.

Logical replication has some limitations that should be considered while migrating from OSS Postgres to FUJITSU Enterprise Postgres. These limitations include:

- The table must have a primary key or unique key.
- Superuser privileges are needed to add all tables.

To perform an online migration, complete the following steps:

1. Install and initiate a FUJITSU Enterprise Postgres Database instance, as described in 3.2, "FUJITSU Enterprise Postgres 12 server installation on LinuxONE" on page 98.
2. Change the configuration parameters for the source and destination database and restart the instances. Example 7-68 shows the necessary parameter changes.

Example 7-68 Configuration parameter changes

```
// Add to OSS PostgreSQL postgresql.conf file.
listen_addresses='*'
wal_level= logical

// Add to FUJITSU Enterprise Postgres postgresql.conf file.
listen_addresses='*'
port = 27501
wal_level= logical
```

3. Change the `pg_hba.conf` configuration file to allow replication and reload the instances (Example 7-69).

Example 7-69 Changing the pg_hba.conf configuration file

```
host     all     all     0.0.0.0/0    md5
```

4. To demonstrate database migration, we created a set of sample tables with data by using **`pgbench`** on the source OSS PostgreSQL database (Example 7-70).

Example 7-70 Creating sample tables

```
[postgres@rdbkpgr3 data]$ pgbench -i -s 10
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...
100000 of 1000000 tuples (10%) done (elapsed 0.06 s, remaining 0.57 s)
200000 of 1000000 tuples (20%) done (elapsed 0.26 s, remaining 1.03 s)
300000 of 1000000 tuples (30%) done (elapsed 0.37 s, remaining 0.87 s)
400000 of 1000000 tuples (40%) done (elapsed 0.53 s, remaining 0.80 s)
500000 of 1000000 tuples (50%) done (elapsed 0.73 s, remaining 0.73 s)
600000 of 1000000 tuples (60%) done (elapsed 0.85 s, remaining 0.57 s)
700000 of 1000000 tuples (70%) done (elapsed 1.02 s, remaining 0.44 s)
800000 of 1000000 tuples (80%) done (elapsed 1.21 s, remaining 0.30 s)
900000 of 1000000 tuples (90%) done (elapsed 1.33 s, remaining 0.15 s)
1000000 of 1000000 tuples (100%) done (elapsed 1.50 s, remaining 0.00 s)
```

```
vacuuming...
creating primary keys...
done.
[postgres@rdbkpgr3 data]$ psql -c 'select count(*) from pgbench_accounts;'
  count
---------
 1000000
(One row)
```

5. Create a dump of the database schema without data by using the native PostgreSQL tool **`pg_dumpall`** (Example 7-71).

Example 7-71 Creating a schema dump

```
[postgres@rdbkpgr3 ~]$ pg_dumpall -s -f databse_schema.sql
```

6. Import the database schema to FUJITSU Enterprise Postgres (Example 7-72).

Example 7-72 Importing the schema dump

```
[postgres@rdbkpgr3 ~]$ scp databse_schema.sql
fsepuser@xxx.xx.xx.164:/home/fsepuser
Password:
databse_schema.sql

fsepuser@rdbkpgs3:~> psql -d postgres -p 27502 < databse_schema.sql
SET
SET
SET
CREATE ROLE
ALTER ROLE
You are now connected to database "template1" as user "fsepuser".
SET
SET
SET
SET
SET
 set_config
------------

(One row)

SET
SET
SET
SET
You are now connected to database "postgres" as user "fsepuser".
SET
SET
SET
SET
SET
 set_config
------------

(One row)
```

```
SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE


[fsepuser@rdbkpgr3 inst3]$ psql -d postgres -p 27502
psql (12.1)
Type "help" for help.

postgres=# select count(*) from pgbench_accounts;
 count
-------
     0
(One row)
```

7. Submit the command that is shown in Example 7-73 to add a publication to the current database. Using the parameter **`FOR ALL TABLES`** marks the publication as one that replicates changes for all tables in the database, including tables that are created in the future.

Example 7-73 Creating a publication

```
postgres=# CREATE PUBLICATION my_pub FOR ALL TABLES;
CREATE PUBLICATION
```

8. Create a subscription on FUJITSU Enterprise Postgres by submitting the command that is shown in Example 7-74. The subscription represents a replication connection to the publisher. After the subscription is created, the initial snapshot is transferred to FUJITSU Enterprise Postgres.

Example 7-74 Creating a subscription

```
postgres=# CREATE SUBSCRIPTION my_sub CONNECTION 'dbname=postgres host=localhost
user=postgres port=5432' PUBLICATION my_pub;
NOTICE:  created replication slot "my_sub" on publisher
CREATE SUBSCRIPTION
postgres=# 2020-11-15 23:01:07.432 EST [131954] LOG:  logical replication apply
worker for subscription "my_sub" has started
2020-11-15 23:01:07.440 EST [131956] LOG:  logical replication table
synchronization worker for subscription "my_sub", table "pgbench_history" has
started
2020-11-15 23:01:07.450 EST [131958] LOG:  logical replication table
synchronization worker for subscription "my_sub", table "pgbench_branches" has
started
```

```
2020-11-15 23:01:07.451 EST [131956] LOG:  logical replication table
synchronization worker for subscription "my_sub", table "pgbench_history" has
finished
2020-11-15 23:01:07.452 EST [131960] LOG:  logical replication table
synchronization worker for subscription "my_sub", table "pgbench_accounts" has
started
2020-11-15 23:01:07.467 EST [131958] LOG:  logical replication table
synchronization worker for subscription "my_sub", table "pgbench_branches" has
finished
2020-11-15 23:01:07.468 EST [131962] LOG:  logical replication table
synchronization worker for subscription "my_sub", table "pgbench_tellers" has
started
2020-11-15 23:01:07.481 EST [131962] LOG:  logical replication table
synchronization worker for subscription "my_sub", table "pgbench_tellers" has
finished
2020-11-15 23:01:10.765 EST [131960] LOG:  logical replication table
synchronization worker for subscription "my_sub", table "pgbench_accounts" has
finished
postgres=# select count(*) from pgbench_accounts;
  count
---------
 1000000
(One row)
```

9. In Example 7-75, we add data to the OSS PostgreSQL database instance.

Example 7-75 Adding data to the accounts table

```
postgres=# insert into pgbench_accounts values (generate_series(1000001, 1000100), 1, 0,
NULL);
INSERT 0 100
postgres=# select count(1) from pgbench_accounts;
  count
---------
 1000100
(One row)
```

10. After the initial data transfer, logical replication replicates the changes between the source and target databases. We used the queries that are shown in Example 7-76 to validate that the amount of newly added data from the source database is the same on the target database.

Example 7-76 Validating the count and newly added data

```
postgres=# select count(*) from pgbench_accounts;
  count
---------
 1000000
(One row)

postgres=# select count(*) from pgbench_accounts;
  count
---------
 1000100
(One row)
```

11. After the data is synchronized, the workload is switched from OSS Postgres to FUJITSU Enterprise Postgres.

8

Database performance features of FUJITSU Enterprise Postgres

No matter which choices businesses make, IT systems must adapt to keep pace with exponential data growth and quickly analyze that data. Every minute that is saved in ingesting and analyzing data has tremendous value. The demand to quickly perform analysis puts tremendous pressure on IT systems. Business and data analysts are looking for faster ways of processing data aggregation and manipulation to support business leaders to make business decisions.

FUJITSU Enterprise Postgres offers three performance features for enhancing ingesting and analyzing data by fully optimizing the processor, memory, storage, and I/O resources. These performance-improving features are *Vertical Clustered Index (VCI)*, *FUJITSU High-Speed Data Load*, and *Global Meta Cache (GMC)*.

In this chapter, we describe the following items:

- Implementation of the VCI in stand-alone and high availability (HA) environments.
- Usage and configuration of FUJITSU High-Speed Data Load, performance comparison of FUJITSU High-Speed Data Load with open source software (OSS) PostgreSQL COPY on IBM LinuxONE, and performance tuning recommendations.
- Configuration and a use case for GMC and the performance results with GMC Off and GMC On.

This chapter covers the following topics:

- Vertical Clustered Index
- FUJITSU High-Speed Data Load (pgx_loader)
- Global Meta Cache

8.1 Vertical Clustered Index

VCI enables fast aggregation by using its memory-resident and disk-resident functions. VCI has a data structure that is suitable for aggregation of column data, and features for parallel scan and disk compression. The memory-resident feature reduces disk I/O that occurs during aggregation. The feature consists of the preinstall feature that reads VCI data to memory in advance, and the stable buffer feature that suppresses VCI data eviction from memory. This chapter describes how to install and operate the features.

8.1.1 VCI applicability

VCI is a feature for achieving fast aggregation. Its usage depends on certain conditions being met. The conditions in which VCI can be more effective are:

- Single table processing.
- Processing that handles many rows in the table.
- Processing that handles some columns in the table.
- Processing that performs heavy aggregation, such as simultaneous sum and average aggregation.

  VCI cannot be used in the following cases, so it is necessary to determine its effectiveness in advance:

- The data type of the target table or column contains VCI restrictions.
- The SQL statement does not meet the VCI operating conditions.
- VCI is slower based on cost estimation.

8.1.2 Benefits of VCI

- No application change: VCI is enabled as an index so that the user does not have to change the application configuration.
- Crash-safe: The memory and disk has the data so that the user can restart the aggregation with VCI if the transaction fails.
- VCI can be used with data in disk when the data that is used for VCI exceeds the configured memory size by user.
- The function includes disk compression and parallel scan. The VCI data is compressed, and aggregation can be performed in parallel with the power of multiple cores.

8.1.3 VCI architecture

VCI uses write-buffer, row-based Write Optimized Store (WOS) in addition to the columnar data structure Read Optimized Store (ROS). The data is synchronously reflected to the row-based WOS during data update. Then, a certain amount of data is stored in WOS, and the ROS control daemon asynchronously converts it to ROS. This architecture improves performance because converting each update into a columnar index improves the update process response times.

Note: For more information about the VCI architecture, see 2.3.1, “Vertical Clustered Index: Increased aggregation performance” on page 59.

For detailed evaluation criteria for installing VCI, and the resource estimation and configuration steps, see 9.1, “Installing Vertical Clustered Index (VCI)” of the *FUJITSU Enterprise Postgres Operation Guide*.

8.1.4 VCI use case scenarios

In this section, we describe three use case scenarios that use VCI.

Scenario 1: Using VCI in high availability environment

In this scenario, we consider the HA environment that we created in Chapter 5, “High availability and high reliability architectures” on page 165. Figure 8-1 on page 292 shows the HA configuration that is being used for this scenario. The purpose of this scenario is to showcase that when VCI is created on a primary database instance, it is replicated to standby and cascaded database instances. With this scenario, we achieve the benefit of distributing resource-intensive read-only aggregation queries across several cascaded database instances, which reduces the load on the primary server. At the same time, we also achieve the benefit of setting up Hybrid Transactional Analytical Processing (HTAP) on the primary database server. For more information about the HTAP use cases of VCI, see Chapter 2, “Introducing FUJITSU Enterprise Postgres features on IBM LinuxONE” on page 37.

Data flow

In this configuration, data is transferred by streaming replication from the primary database server to the standby database server. The Mirroring Controller (MC) is configured on both the database servers to provide automatic failover. The standby database instance transfers the data to the cascaded read-replica database server and the backup database server. The Server Assistant arbitration server and backup database server are configured on the same virtual machine (VM) (IP address is XXX.XX.XX.172).

There are five database servers that we use in this scenario. To learn how to configure an HA environment as shown in Figure 8-1 on page 292, see Chapter 5, “High availability and high reliability architectures” on page 165.

Here is the list of IP addresses for the various servers:

- Primary server IP address: XXX.XX.XX.156
- Standby server IP address: XXX.XX.XX.161
- Cascaded read replica server IP address: XXX.XX.XX.157
- Backup server IP address: XXX.XX.XX.172
- Server Assistant arbitration server IP address: XXX.XX.XX.172

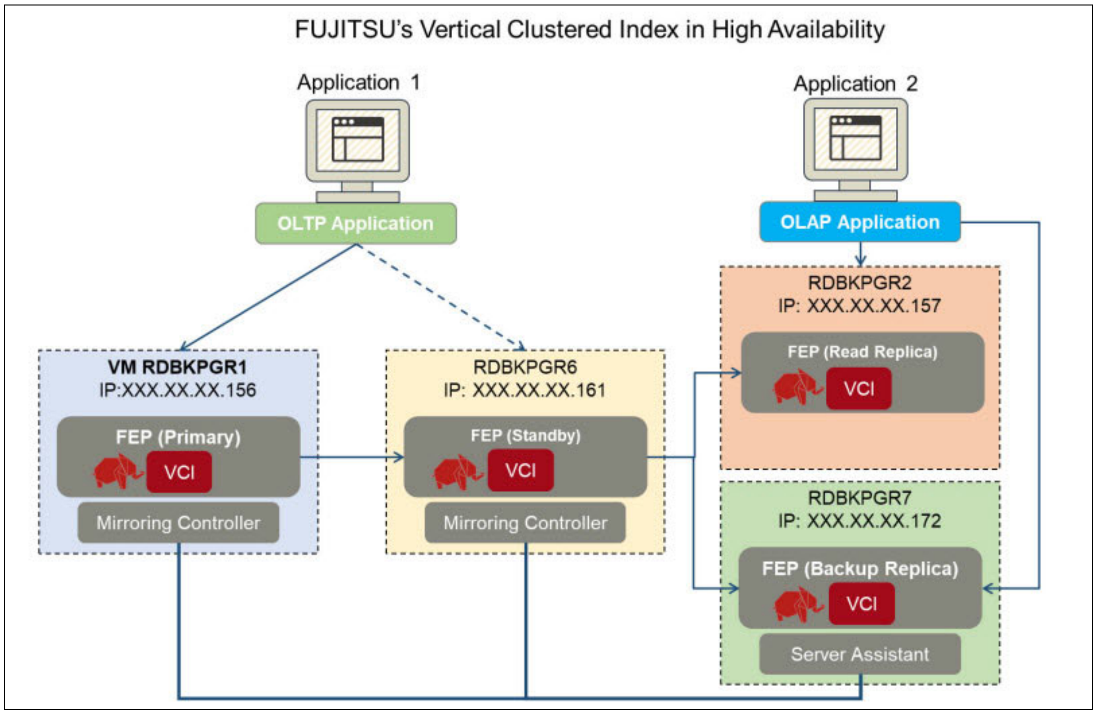


Figure 8-1 VCI in a high availability implementation

In the following section, we configure a VCI on all five database servers:

1. Enable VCI on all five database servers. To do this task, edit the `postgresql.conf` configuration file and update the VCI configuration parameters on all five database servers.

Note: We described the VCI parameters in 2.3.1, "Vertical Clustered Index: Increased aggregation performance" on page 59.

Because VCI is a performance enhancement feature, a few factors are considered for performance tuning the VCI, such as the application workload, transactions per second, service-level agreement (SLA), replication, and partitioning considerations. As a best practice, performance testing should be conducted for each use case that is considered for implementing VCI.

For more information about the evaluation criteria for installing VCI, resource estimation, and configuration steps, see 9.1, "Installing VCI" in the *FUJITSU Enterprise Postgres Operation Guide*.

Example 8-1 on page 293 shows how to configure VCI on the primary, standby, read-replica, and backup database servers. In the lab environment that we used, we set the following parameter:

```
vci.max_parallel_degree = 2
```

For a VM with a high number of virtual Integrated Facility for Linux (IFL) processors, you can set a higher value that is equal to the number of virtual IFL processors x 2 (if symmetric multi-threading is enabled) for **`vci.max_parallel_degree`** to enable more parallel processing.

Example 8-1 Configuration file for primary, standby, cascade, and backup servers

```
$ vim /database/inst1/postgresql.conf
shared_preload_libraries = 'vci, pg_prewarm'
session_preload_libraries = 'vci, pg_prewarm'
shared_buffers = 12GB
reserve_buffer_ratio = 30
vci.control_max_workers = 8
vci.max_parallel_degree = 2
max_worker_processes = 18
```

As shown in Example 8-1, the **shared_buffers** parameter is 12 GB, and **reserve_buffer_ratio** is configured as 30% of **shared_buffers**. So, for this configuration, VCI has a maximum of 3.6 GB of memory that is available for fast in-memory aggregation.

2. After the VCI is configured and set up with the required parameters, restart the database servers, and then create a **pgbench** database in which you create VCI indexes on the tables. The **pgbench** database is initialized with a scale factor of 1000. Example 8-2 shows the steps to restart the database servers and initialize the **pgbench** database.

Example 8-2 Starting servers and initializing pgbench

```
# Restart instance regarding cascade and backup server
$ pg_ctl restart -D /database/inst1
# Stop Mirroring Controller of primary server
$ mc_ctl stop -M /mcdir/inst1 -a
# Start Mirroring Controller of primary server
$ mc_ctl start -M /mcdir/inst1
# Start Mirroring Controller of Standby server
$ mc_ctl start -M /mcdir/inst1
# Initialize the pgbench
$ pgbench -i postgres -s 1000
```

3. After the **pgbench** database is initialized and all the tables are created, run **explain analyze** to see the query plan for the aggregation query on the **pgbench_accounts** table. The purpose of capturing query plan before creating a VCI index is to compare the two query plans before and after creating the VCI index. Example 8-3 shows the **explain analyze** output. The query planner used a parallel sequential scan on the **pgbench_accounts** table to compute the average on the `abalance` column. As shown under `Execution Time`, this query takes 34887.651 ms to return the results.

Example 8-3 Checking the query plan before configuring the VCI

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500 -c “explain analyze
select avg(abalance) from pgbench_accounts”
                                                                            QUERY
PLAN
---------------------------------------------------------------------------------
----------------------------------------------------------------
------------
 Finalize Aggregate  (cost=2161178.55..2161178.56 rows=1 width=32) (actual
time=34885.923..34885.931 rows=1 loops=1)
   Lwlock Wait: buffer_mapping total_waits=145 total_wait_time=0.341
   ->  Gather  (cost=2161178.33..2161178.54 rows=2 width=32) (actual
time=34885.374..34886.495 rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         Lwlock Wait: buffer_mapping total_waits=145 total_wait_time=0.341
```

```
        ->  Partial Aggregate  (cost=2160178.33..2160178.34 rows=1 width=32)
(actual time=34863.714..34863.719 rows=1 loops=3)
              Lwlock Wait: buffer_mapping total_waits=145 total_wait_time=0.341
              ->  Parallel Seq Scan on pgbench_accounts  (cost=0.00..2056011.67
rows=41666667 width=4) (actual time=0.025..17769.595 rows=333333
33 loops=3)
                    Lwlock Wait: buffer_mapping total_waits=145
total_wait_time=0.341
 Planning Time: 0.330 ms
 Execution Time: 34887.651 ms
(Twelve rows)
```

4. Having captured the **`explain analyze`** output (shown in Example 8-3 on page 293), install the VCI extension on the primary database server, as shown in Example 8-4 by using the **`CREATE EXTENSION VCI`** statement.

Example 8-4 Configuring VCI on the primary server

```
# install extension on primary server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500 -c “CREATE EXTENSION vci”
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500 -c “CREATE EXTENSION
pg_prewarm”
# Create VCI on primary server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500 -c “create index
idx_vci_abalance on pgbench_accounts using vci(abalance) WITH
(stable_buffer=true)”
# prewarm index
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500 -c "select
pgx_prewarm_vci('idx_vci_abalance')"
 pgx_prewarm_vci
-----------------
          326393
```

In Example 8-4, we create VCI index with the name `idx_vci_abalance` on column `abalance` of table **`pgbench_accounts`**. We also prewarm the index by using the function **`pgx_prewarm_vci`**, which loads the VCI index keys into memory for faster in-memory query processing.

5. Because the VCI index is created on the primary database server, check whether the VCI index cascaded to the standby, read-replica, and the backup database servers by running an SQL query on the **`indexref`** table on the other three database servers: standby, read-replica, and backup database servers, as shown in Example 8-5.

Example 8-5 Checking the index on the primary, standby, cascade and backup servers

```
# Check the index on each server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500 -c "SELECT indexdef FROM
pg_indexes WHERE indexdef LIKE '%vci%'"
                                    indexdef
--------------------------------------------------------------------------------
 CREATE INDEX idx_vci_abalance ON public.pgbench_accounts USING vci (abalance)
WITH (stable_buffer=’true’)
(One row)
```

In Example 8-5 on page 294, we see that we get the same result across the other three servers, which confirms that the VCI index is configured successfully on the secondary servers.

In Example 8-6, we run **`explain analyze`** on the aggregation query that was used in Example 8-3 on page 293 to check whether VCI is now used by the query planner and whether VCI improved the query execution time.

Example 8-6 Checking the query plan

```
# check query plan on primary server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500 -c “explain analyze 
select avg(abalance) from pgbench_accounts”
                                                                QUERY PLAN
---------------------------------------------------------------------------------
-----------------------------------------------------
 Custom Scan (VCI Aggregate)  (cost=2889345.00..2889345.01 rows=1 width=32) 
(actual time=4424.269..4424.270 rows=1 loops=1)
   Allocated Workers: 2
   ->  Custom Scan (VCI Scan) using idx_vci_abalance on pgbench_accounts  
(cost=0.00..2639345.00 rows=100000000 width=4) (never executed)
 Planning Time: 0.232 ms
 Execution Time: 4431.377 ms
(Five rows)
```

The **`explain analyze`** output shows that the VCI, **`idx_vci_abalance`**, is used for computing the aggregation of column `abalance`. Also, the execution time is considerably less than the execution time that was captured in Example 8-3 on page 293.

In this use case, we showed that VCI and streaming replication together can be used to implement HTAP in a HA configuration and distribute the load of different Online Transaction Processing (OLTP) and Online Analytical Processing (OLAP) applications by running resource-intensive analytical queries on secondary database servers (standby, read-replica, or backup database servers).

Scenario 2: Using VCI with logical replication

In this scenario, we use primary and standby servers to demonstrate the VCI benefits with logical replication. In “Scenario 1: Using VCI in high availability environment” on page 291, VCI was configured on primary, standby, read-replica, and backup servers. However, if there is no requirement to implement HTAP, then we do not have to configure VCI on the primary server, but VCI can still be used on the secondary servers. To achieve this setup, we use logical replication, which we set up by completing the following steps:

1. Enable logical replication, as shown in Example 8-7, by configuring the **`wal_level=logical`** parameter on the primary database server, which enables the primary and standby servers to have streaming and logical replications.

Example 8-7 Configuring logical replication on the primary server

```
# configure the parameter on postgres.conf about primary server
$ vim /database/inst1/postgresql.conf
listen_addresses = '*'
port = 27500
logging_collector = on
wal_level = logical
```

2. After `postgresql.conf` is updated, restart the primary database server, as shown in Example 8-8.

Example 8-8 Restarting the primary server

```
# start the primary server
$ pg_ctl restart -D /database/inst1
```

3. Initialize the **pgbench** database on the primary database server, as shown in Example 8-9. In this example, we use a scale factor of 1000, which creates approximately 15 GB of data.

Example 8-9 Preparing pgbench

```
# Prepare the data by pgbench on primary server
$ pgbench -i postgres -s 1000
```

4. Configure the standby server for logical replication, as shown in Example 8-10, by creating a database instance on the standby server, configuring **wal_level** as `logical`, set up host-based authentication, and restart the standby server to make the changes permanent. Initialize **pgbench** database on the standby database instance.

Example 8-10 Setting up on the standby server

```
# Create instance on Standby server
$ initdb -D /database/inst1 --waldir=/transaction/inst1 --lc-collate="C"
--lc-ctype="C" --encoding=UTF8
# setup postgresql.conf for logical replication on Standby server
$ vim /database/inst1/postgresql.conf
listen_addresses = '*'
port = 27500
logging_collector = on
wal_level = logical
# setup pg_hba.conf
$ vim /database/inst1/pg_hba.conf
host    all            all            XXX.XX.XX.0/24          trust
host    replication   all            XXX.XX.XX.0/24          trust
# Start instance about Standby server
$ pg_ctl start -D /database/inst1
# Prepare the data by pgbench on Standby server
# Note that the command between primary and Standby is different.
$ pgbench -i postgres -I dtvp
```

5. To enable logical replication, create a publisher with the name `inst1_pub` for the table **pgbench_accounts** on the primary database server, as shown in Example 8-11. This action is confirmed by querying the **pg_publication_tables** table.

Example 8-11 Configuring the publication for logical replication on the primary server

```
# Create publication on primary server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500 -c "create publication
inst1_pub for table pgbench_accounts"
# Confirm publication primary server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500 -c "select * from
pg_publication_tables"
  pubname  | schemaname |    tablename
-----------+------------+------------------
```

```
 inst1_pub | public      | pgbench_accounts
(One row)
```

6. Configure the subscription with the name `inst1_sub` on the standby database server, as shown in Example 8-12. Confirm the creation of the subscription by querying the **`pg_subscription`** table on the standby server.

Example 8-12 Configuring the subscription for logical replication on the standby server

```
# Create Subscription on Standby server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.161 -p 27500 -c "create subscription
inst1_sub connection 'host=XXX.XX.XX.156 port=27500 dbname=postgres user=fsepuser'
publication inst1_pub"
# Confirm Subscription Standby server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.161 -p 27500 -c "select * from
pg_subscription"
  oid  | subdbid |  subname  | subowner | subenabled |
subconninfo                              | subslotname | subsynccommit |
subpublications
-------+---------+-----------+----------+------------+----------------------------
---------------------------------+-------------+---------------+-----------------
 16402 |   13468 | inst1_sub |       10 | t          | host=XXX.XX.XX.156
port=27500 dbname=postgres user=fsepuser | inst1_sub   | off           |
{inst1_pub}
(One row)
```

7. We have configured logical replication between the primary and standby database instances. Now, we capture the **`explain analyze`** output for an aggregation SQL query that computes the average of the `abalance` column of the **`pgbench_accounts`** table, as shown in Example 8-13. Before VCI is created, the query planner chose to run a parallel sequential scan to access the data.

Example 8-13 Checking the query plan on the standby server before using VCI

```
# Check index
$ psql -d postgres -U fsepuser -h XXX.XX.XX.161 -p 27500 -c "SELECT indexdef FROM
pg_indexes WHERE indexdef LIKE '%vci%'"
 indexdef
----------
(0 row)
# Check query plan
$ psql -d postgres -U fsepuser -h XXX.XX.XX.161 -p 27500 -c “explain analyze
select avg(abalance) from pgbench_accounts”
                                                                         QUERY
PLAN
---------------------------------------------------------------------------------
---------------------------------------------------------------------------
 Finalize Aggregate  (cost=2161178.80..2161178.81 rows=1 width=32) (actual
time=35183.061..35183.068 rows=1 loops=1)
   Lwlock Wait: buffer_mapping total_waits=72 total_wait_time=0.155
   ->  Gather  (cost=2161178.58..2161178.79 rows=2 width=32) (actual
time=35181.905..35183.217 rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         Lwlock Wait: buffer_mapping total_waits=72 total_wait_time=0.155
         ->  Partial Aggregate  (cost=2160178.58..2160178.59 rows=1 width=32)
(actual time=35157.825..35157.829 rows=1 loops=3)
               Lwlock Wait: buffer_mapping total_waits=72 total_wait_time=0.155
               ->  Parallel Seq Scan on pgbench_accounts  (cost=0.00..2056011.87
rows=41666687 width=4) (actual time=0.038..18107.851 rows=33333333 loops=3)
                     Lwlock Wait: buffer_mapping total_waits=72
total_wait_time=0.155
 Planning Time: 1.839 ms
 Execution Time: 35190.243 ms
(12 row)
```

8. Configure VCI on the standby server, as shown in Example 8-14.

Example 8-14 Configuring the VCI on the standby server

```
# Configure the postgreql.conf on Standby server
$ vim /database/inst1/postgresql.conf
shared_preload_libraries = 'vci, pg_prewarm'
session_preload_libraries = 'vci, pg_prewarm'
shared_buffers = 12GB
reserve_buffer_ratio = 30
vci.control_max_workers = 8
vci.max_parallel_degree = 2
max_worker_processes = 18
# restart Standby server
$ pg_ctl restart -D /database/inst1
# install extension on Standby server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.161 -p 27500 -c “CREATE EXTENSION vci”
$ psql -d postgres -U fsepuser -h XXX.XX.XX.161 -p 27500 -c “CREATE EXTENSION
pg_prewarm”
# Create VCI on Standby server
$ psql -d postgres -U fsepuser -h XXX.XX.XX.161 -p 27500 -c “create index
idx_vci_abalance on pgbench_accounts using vci(abalance) WITH
(stable_buffer=true)”
# prewarm index
$ psql -d postgres -U fsepuser -h XXX.XX.XX.161 -p 27500 -c "select
pgx_prewarm_vci('idx_vci_abalance')"
 pgx_prewarm_vci
-----------------
          326393
```

We configured VCI on the standby server only because we use logical replication.

9. As shown in Example 8-15, we rerun **`explain analyze`** for the same query that was used in Example 8-13 on page 297. In Example 8-15, The query planner chose VCI custom scan to compute aggregation by using the **`idx_vci_abalance`** VCI index with considerably less run time. The use case was not performed to showcase VCI performance.

Example 8-15 Checking the query plan on the standby server

```
# Check the query plan
$  psql -d postgres  -U fsepuser -h XXX.XX.XX.161 -p 27500 -c “explain analyze
select avg(abalance) from pgbench_accounts”
                                                                QUERY PLAN
--------------------------------------------------------------------------------
 Custom Scan (VCI Aggregate)  (cost=2889345.00..2889345.01 rows=1 width=32)
(actual time=4457.552..4457.554 rows=1 loops=1)
```

```
   Allocated Workers: 2
   ->  Custom Scan (VCI Scan) using idx_vci_abalance on pgbench_accounts
(cost=0.00..2639345.00 rows=100000000 width=4) (never executed)
 Planning Time: 0.480 ms
 Execution Time: 4483.392 ms
(Five rows)
```

In this scenario, we used logical replication and VCI to eliminate the need to establish an Extract, Transform, and Load (ETL) process between OLTP and OLAP applications. In this scenario, OLTP application is assumed to be connected to the primary database instance, and the standby database instance is used to run resource-intensive complex analytical SQL queries. By using VCI on the standby database server, we demonstrated that logical replication and VCI together can reduce the over-all batch processing time, which otherwise would exist between OLTP and OLAP systems.

Scenario 3: Using the VCI combination with partitioning

In this scenario, we demonstrate the VCI combination with partitioning on a single server. Table partitioning is a method to distribute the data for faster query performance and easy housekeeping. Often, VCI is required on large partitioned tables, and then users use VCI on partitioned tables that have a large amount of data.

In this scenario, we show the steps to implement VCI on partitioned tables and compare the **explain analyze** results before and after creating VCI for an SQL query for aggregation computations. Our purpose is not to show the performance of VCI, but only the steps.

1. Initialize **pgbench**, as shown in Example 8-16.

Example 8-16 Configuring data with pgbench

```
# Create table
$ pgbench -i postgres -I dtvp
```

2. As shown in Example 8-17, drop the **pgbench_accounts** table and re-create it as a partitioned table that is partitioned by range on the `aid` column. Also, add the primary key by using the partitioning column `aid`.

Example 8-17 Configuring the partitions

```
# Set up partitioning
$ psql -d postgres -c "drop table pgbench_accounts"
$ psql -d postgres -c "create table pgbench_accounts(aid    int not null,bid
int,abalance int,filler char(84)) partition by range(aid)"
$ psql -d postgres -c "create table pgbench_accounts_1 partition of
pgbench_accounts for values from (1) to (50000001) with (fillfactor=100)"
$ psql -d postgres -c "create table pgbench_accounts_2 partition of
pgbench_accounts for values from (50000001) to (100000001) with (fillfactor=100)"
$ psql -d postgres -c "alter table pgbench_accounts add primary key (aid)"
```

3. Populate the **pgbench** tables, as shown in Example 8-18.

Example 8-18 Loading data

```
# load the data with the pgbench command
$ pgbench -i postgres -I gv -s 1000
```

4. After the tables are populated, capture the **explain analyze** output for the aggregation SQL, as shown in Example 8-19 on page 300 before configuring VCI.

Example 8-19 Checking the query plan before configuring VCI

```
# Check index
$ psql -d postgres -c "SELECT indexdef FROM pg_indexes WHERE indexdef LIKE
'%vci%'"
 indexdef
----------
(0 row)
# Check query plan
$ psql -d postgres -c "set max_parallel_workers_per_gather=1;explain select
avg(abalance) from pgbench_accounts"
                                                  QUERY PLAN
----------------------------------------------------------------------------------
------------------------------
 Finalize Aggregate  (cost=2669758.41..2669758.42 rows=1 width=32)
   ->  Gather  (cost=2669758.30..2669758.41 rows=1 width=32)
         Workers Planned: 1
         ->  Partial Aggregate  (cost=2668758.30..2668758.31 rows=1 width=32)
               ->  Parallel Append  (cost=0.00..2521699.40 rows=58823560 width=4)
                     ->  Parallel Seq Scan on pgbench_accounts_1
(cost=0.00..1113790.95 rows=29411795 width=4)
                     ->  Parallel Seq Scan on pgbench_accounts_2
(cost=0.00..1113790.65 rows=29411765 width=4)
(Seven rows)
$ time psql -d postgres -c "set max_parallel_workers_per_gather=1;select
avg(abalance) from pgbench_accounts"
              avg
--------------------------------
 0.00000000000000000000000000000
(One row)
real    0m5.221s
user    0m0.002s
sys     0m0.005s
```

The query planner used a parallel sequential scan to access data from partitions of the **pgbench_accounts** table.

5. As shown in Example 8-20, we configure VCI on column abalance, confirm that was successfully created, and prewarm it by using the **pgx_prewarm_vci** function (Example 8-21 on page 301).

Example 8-20 Configuring the VCI

```
# Create VCI
$ psql -d postgres -c "create index idx_vci_abalance on pgbench_accounts using
vci(abalance) WITH (stable_buffer=true)"
# Check index
$ psql -d postgres -c "SELECT indexdef FROM pg_indexes WHERE indexdef LIKE
'%vci%'"
                                            indexdef
----------------------------------------------------------------------------------
--------------
 CREATE INDEX idx_vci_abalance ON ONLY public.pgbench_accounts USING vci
(abalance) WITH (stable_buffer='true')
 CREATE INDEX pgbench_accounts_1_abalance_idx ON public.pgbench_accounts_1 USING
vci (abalance) WITH (stable_buffer='true')
```

```
 CREATE INDEX pgbench_accounts_2_abalance_idx ON public.pgbench_accounts_2 USING
vci (abalance) WITH (stable_buffer='true')
(Three rows)
```

Example 8-21 Prewarming to use VCI

```
# Prewarm
$ psql -d postgres -c "select pgx_prewarm_vci('idx_vci_abalance')"
 pgx_prewarm_vci
-----------------
          326404
(One row)
```

6. Now, after creating the VCI index **`idx_vci_abalance`**, we run **`explain analyze`** on the aggregation SQL again and capture the query planner output, as shown in Example 8-22. We notice that instead of parallel sequential scan, the query planner chose VCI to access the data and compute it in memory aggregation on column `abalance` by using the **`idx_vci_abalance`** VCI partitions.

Example 8-22 Checking the query plan

```
# Check query plan
$ psql -d postgres -c “set max_parallel_workers_per_gather=0;explain select
avg(abalance) from pgbench_accounts”
                                                                   QUERY PLAN
-------------------------------------------------------------------------------------
-------------------------------------------------------------
 Aggregate  (cost=3389346.91..3389346.92 rows=1 width=32)
   ->  Append  (cost=0.00..3139346.78 rows=100000052 width=4)
         ->  Custom Scan (VCI Scan) using pgbench_accounts_1_abalance_idx on
pgbench_accounts_1  (cost=0.00..1319673.52 rows=50000052 width=4)
               Allocated Workers: 1
         ->  Custom Scan (VCI Scan) using pgbench_accounts_2_abalance_idx on
pgbench_accounts_2  (cost=0.00..1319673.00 rows=50000000 width=4)
               Allocated Workers: 1
(Six rows)
$ time psql -d postgres -c “set max_parallel_workers_per_gather=0;select
avg(abalance) from pgbench_accounts”
              avg
--------------------------------
 0.0000000000000000000000000000
(One row)
real    0m4.951s
user    0m0.002s
sys     0m0.006s
```

In this scenario, we demonstrated that VCI can be used with partitioning and for implementing in-memory aggregation while leveraging the parallel processing benefits of partitioning.

8.2 FUJITSU High-Speed Data Load (pgx_loader)

In this section, we describe the steps to configure and use FUJITSU High-Speed Data Load on a FUJITSU Enterprise Postgres server with Red Hat Enterprise Linux Server 8 on IBM LinuxONE.

This feature enables us to run the **COPY FROM** command in parallel. The feature can perform bulk data load with a parallelism base on the available IFL processors or virtual IFL processors. Mission-critical systems can take advantage of multi-cores by using this feature. This feature runs the **COPY** command with degrees of multiple processes that are configured automatically by resources of the environment.

8.2.1 Verification

In this session, we demonstrate the setup and running of the **pgx_loader** function.

The verification is done by the normal **COPY** command with a core, and FUJITSU High-Speed Data Load with the power of two and three cores. Each verification is performed with an index. The machine that is used for verification has four virtual IFL processors and 32 GB of memory.

Creating the data

The data that is used for the verification is created by the **random** command with 10,000,000 rows and 1.3 GB. The data size is calculated when the data is backed up. In Example 8-23, we show the preparation of the data.

Example 8-23 Preparing the data

```
postgres=# CREATE TABLE tbl (id int, data1 text, data2 text, s_date timestamp,
e_date timestamp, foreign_id int, primary key (id));
CREATE TABLE
postgres=# INSERT INTO tbl SELECT s, md5(clock_timestamp()::text),
md5((random()*1000)::text), now(), clock_timestamp(), random()*1000 FROM
generate_series(1, 10000000) as s;
postgres=# COPY tbl TO '/home/fsepuser/exp_data.csv' WITH csv;
COPY 10000000
```

Configuration

Example 8-24 shows the `postgresql.conf` parameters that are configured to optimize the bulk data load performance.

Example 8-24 Configuring postgresql.conf

```
[fsepuser@rdbkpgr3 ~]$ vi /database/inst4/postgresql.conf
shared_buffers = 8GB      # 25% of memory
work_mem = 16MB
checkpoint_timeout = 30min
checkpoint_completion_target = 0.9
max_wal_size = 4GB
wal_buffers = 256MB
maintenance_work_mem = 1GB
jit = off      # specify off to ignore the effect from JIT
max_prepared_transactions = 4
max_worker_processes = 12
max_parallel_workers = 12
```

Execution

Example 8-25 shows three ways to perform bulk data load. The first command uses the open source PostgreSQL COPY command. The second and third commands use FUJITSU Enterprise Postgres High-Speed Data Load with parallelism. The second command uses two virtual IFL processors, and the third command uses three virtual IFL processors for parallel load operation.

Example 8-25 Running COPY, creating pgx_loader extension, and running pgx_loader

```
[fsepuser@rdbkpgr3 ~]$ psql -d postgres -p 27503 -c "COPY tbl FROM
'/home/fsepuser/exp_data.csv' WITH csv"
[fsepuser@rdbkpgr3 ~]$ psql -d postgres -p 27503 -c "CREATE EXTENSION pgx_loader;”
CREATE EXTENSION
[fsepuser@rdbkpgr3 ~]$  pgx_loader load -j 2 -d postgres -p 27503 -c "COPY tbl
FROM '/home/fsepuser/exp_data.csv' WITH csv”
[fsepuser@rdbkpgr3 ~]$  pgx_loader load -j 3 -d postgres -p 27503 -c "COPY tbl
FROM '/home/fsepuser/exp_data.csv' WITH csv”
```

Table 8-1 compares the average execution time that is measured by the operating system’s (OS) time command for five iterations for a normal **COPY** command, FUJITSU High-Speed Data Load with two virtual IFL processors, and FUJITSU High-Speed Data Load with three virtual IFL processors.

Table 8-1 Load utility execution times with indexes on the table

Condition	Times
Normal **COPY** command	33.18
FUJITSU High-Speed Data Load with two cores	26.22
FUJITSU High-Speed Data Load with three cores	24.58

From the results that are shown in Table 8-1, we notice that bulk load processing is faster when using FUJITSU High-Speed Data Load and optimally using all the available virtual IFL processors.

If the indexes on the table are dropped before initiating the load, the load performance is further improved. After the load completes successfully, the indexes are re-created. Table 8-2 shows the load utility execution times comparison after the indexes were dropped before the load and re-created after a successful load. As a result, the data in Table 8-2 shows that the load processing performance is further improved for FUJITSU High-Speed Data Load when the tables did not have any indexes during load.

Table 8-2 Load utility execution times with indexes dropped from the table before load and re-created after load

Condition	Times
Normal **COPY** command	29.51
FUJITSU High-Speed Data Load with two cores	21.62
FUJITSU High-Speed Data Load with three cores	17.16

Performance tuning key considerations

Writing to Write-Ahead-Log (WAL) and checkpointing often affects performance, and the number of parallel processes is increased. Therefore, to have optimal performance, it is a best practice to configure `postgresql.conf` for your performance goals.

- WAL shipping effect on performance:
  - Allocate separate storage locations for table space (table, WAL, and index).
  - Increase the buffer of WAL to prevent checkpoint from running (**`wal_buffers`** and **`max_wal_size`**). Reduce checkpointing when the load runs.
  - Increase **`checkpoint_timeout`**.
  - Decrease the effect of checkpoint (**`checkpoint_completion_target`**).
- Others:
  - Increase **`shared_buffers`** to access the data pages more efficiently (25 - 30% of database server memory recommended).
  - Load without index.
  - Increase **`max_wal_size`** or **`checkpoint_timeout`**.

8.3 Global Meta Cache

In FUJITSU Enterprise Postgres, query processing involves parsing the query, creating a query plan, running the plan, and other steps. To perform these steps, the PostgreSQL process accesses the system catalog, aggregates the working information that is required to access referenced tables, and caches it in the per-process memory. This cached information is called a *meta cache*. Using this cached information, the query processing time is faster because each process searches for the required data in the meta cache instead of searching the system catalog each time.

This data is cached on a per-process basis, so as the number of connections increases, so does the overall meta cache usage. But in an environment with a large database and large number of connections, there is an increase in memory usage due to the increased usage of meta cache.

The FUJITSU Enterprise Postgres GMC feature enables sharing of meta cache information across the processes by caching it in shared memory. By centrally managing the meta cache information in shared memory, the GMC reduces the overall system memory usage. Figure 8-2 on page 305 shows architecture of the GMC feature.

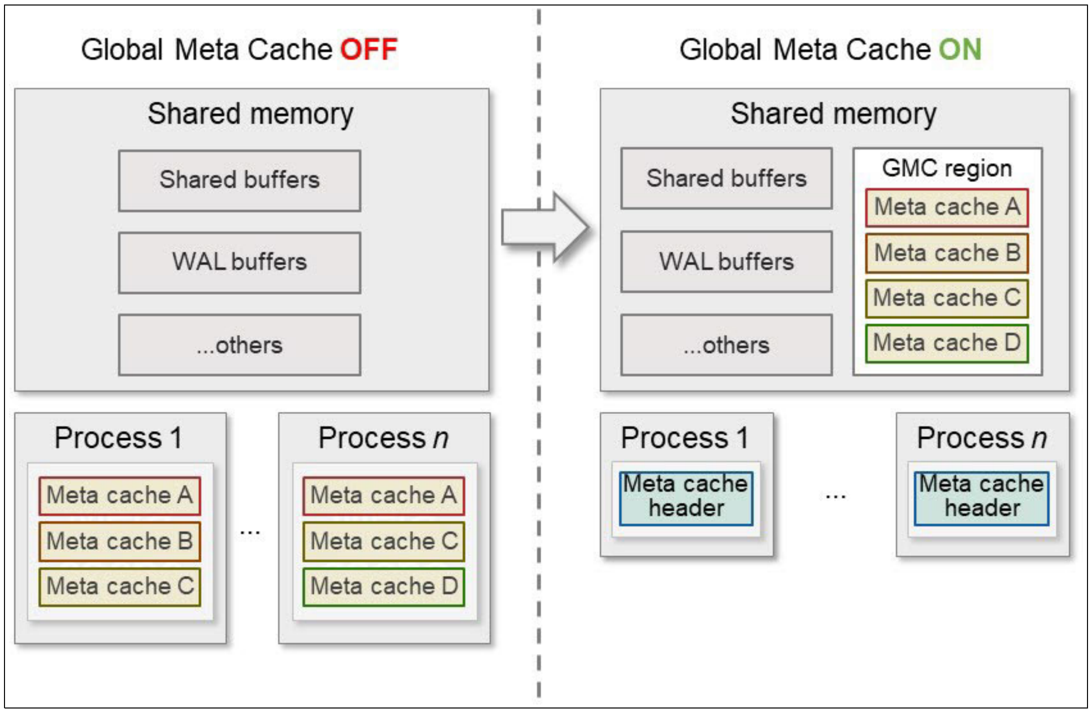


Figure 8-2 Architecture of Global Meta Cache feature

When the GMC feature is ON, the meta cache is cached in the GMC region of the shared memory and the reference to the GMC area is cached as a meta cache header in per-process memory.

Example 8-26 shows how to enable the GMC feature. By default, the GMC feature is disabled.

Example 8-26 Editing the postgresql.conf file and setting the pgx_global_metacache parameter

```
[fsepuser@rdbkpgr3 ~]$ vi /database/inst4/postgresql.conf
pgx_global_metacache = 800MB
```

The **pgx_global_metacache** parameter value specifies the maximum amount of memory for GMC on shared memory. When the value is set to 0, the GMC is disabled.

8.3.1 Using the Global Meta Cache feature

In this section, we describe using the GMC feature and compare memory usage when GMC is on and off. Complete the following steps:

1. We are using **pgbench** to simulate the load on FUJITSU Enterprise Postgres. In Example 8-27, we initialize **pgbench** with a scaling factor of 100, which creates and loads data in to the benchmarking tables.

Example 8-27 Initializing pgbench

```
[fsepuser@rdbkpgr3 ~]$ pgbench -i -s 100 testdb -h XXX.XX.XX.158 -p 27503
dropping old tables...
creating tables...
generating data...
100000 of 10000000 tuples (1%) done (elapsed 0.02 s, remaining 1.98 s)
```

```
…
9900000 of 10000000 tuples (99%) done (elapsed 9.73 s, remaining 0.10 s)
10000000 of 10000000 tuples (100%) done (elapsed 9.90 s, remaining 0.00 s)
```

2. We run **pgbench** with 800 clients (processes) and 100 transactions per client. In Example 8-28, we edit the `postgresql.conf` file and set the required parameters to accept 800 connections and disable GMC.

Example 8-28 Editing the postgresql.conf file

```
[fsepuser@rdbkpgr3 ~]$ vi /database/inst4/postgresql.conf
max_connections = 820
pgx_global_metacache = 0MB
#pgx_global_metacache = 800MB
```

3. For `postgresql.conf` file changes to be effective, we restart the FUJITSU Enterprise Postgres instance, as shown in Example 8-29.

Example 8-29 Restarting the FUJITSU Enterprise Postgres instance

```
[fsepuser@rdbkpgr3 ~]$ pg_ctl restart -D /database/inst4
server stopped
server started
```

4. We take measurements by running the **ps** command during **pgbench** execution, and then average the result for better comparison.

   Example 8-30 shows the steps to run the **pgbench** command and simulate the load on the database instance.

Example 8-30 The pgbench command

```
[fsepuser@rdbkpgr3 ~]$ pgbench -h XXX.XX.XX.158 -p 27503 -c 800 -t 100 testdb;
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 100
query mode: simple
number of clients: 800
number of threads: 1
number of transactions per client: 100
number of transactions actually processed: 80000/80000
latency average = 228.401 ms
tps = 3502.607545 (including connections establishing)
tps = 3503.204862 (excluding connections establishing)
```

5. Open another terminal and run the **ps** command to capture system resources while the **pgbench** command runs, as shown in Example 8-31.

Example 8-31 Running the ps command five times every second to capture system resources

```
[lnxadmin@rdbkpgr3 ~]$ ps auxf | grep postgres | grep -v grep >> ps_result.txt
```

6. Run **pgbench** again with GMC ON.

   Edit the `postgresql.conf` file and enable GMC, as shown in Example 8-32 on page 307.

Example 8-32 Editing the postgresql.conf file.

```
[fsepuser@rdbkpgr3 ~]$ vi /database/inst4/postgresql.conf
max_connections = 820
#pgx_global_metacache = 0MB
pgx_global_metacache = 800MB
```

For the `postgresql.conf` file changes to be effective, restart the FUJITSU Enterprise Postgres instance (Example 8-33).

Example 8-33 Restarting the FUJITSU Enterprise Postgres instance.

```
[fsepuser@rdbkpgr3 ~]$ pg_ctl restart -D /database/inst4
server stopped
server started
```

7. Run **pgbench** and simulate the load on the database instance after initializing the **pgbench**, as shown in Example 8-34.

Example 8-34 Running the pgbench command

```
[fsepuser@rdbkpgr3 ~]$ pgbench -h XXX.XX.XX.158 -p 27503 -c 800 -t 100 testdb;
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 100
query mode: simple
number of clients: 800
number of threads: 1
number of transactions per client: 100
number of transactions actually processed: 80000/80000
latency average = 230.453 ms
tps = 3471.423153 (including connections establishing)
tps = 3472.096708 (excluding connections establishing)
```

8. Open another terminal and run the **ps** command to capture system resources while the **pgbench** command runs (Example 8-35).

Example 8-35 Running the ps command five times every second to capture system resources

```
[lnxadmin@rdbkpgr3 ~]$ ps auxf | grep postgres | grep -v grep >>
ps_result_GlobalMetaCache.txt
```

Based on the output of the **ps** command, we compare the results with GMC ON and GMC OFF, as shown in Table 8-3 on page 308 and Table 8-4 on page 308.

Table 8-3 Memory usage with GMC OFF

GMC OFF	Total memory usage	Number of processes	Average memory usage
	16,174,880	808	20,018
	24,561,640	808	30,398
	26,009,828	808	32,190
	29,076,200	808	35,985
	34,981,632	808	43,294
Overall average	26,160,836	808	32,377

Table 8-4 Memory usage with GMC ON

GMC ON	Total memory usage	Number of processes	Average memory usage
	12,609,220	808	15,605
	21,771,272	808	26,945
	25,128,468	808	31,100
	29,749,604	808	36,819
	33,648,600	808	41,644
Overall average	24,581,433	808	30,423

The overall average with GMC ON is 30,423, and the overall average with GMC OFF is 32,377. The memory usage is less when the GMC feature is enabled, and overall 6% less memory is required to cache table definition and system catalog (meta cache) information (approximately 2 KB per process).

To demonstrate this feature, we are working with a relatively small database, but on large databases and with more connections, the shared memory usage is much more effective and significant, which helps reduce the system memory usage per process.

Depending on the database size and number of connections, the FUJITSU Enterprise Postgres GMC feature is enabled and memory for the GMC area is allocated in the shared memory. The GMC usage is tracked when **`track_gmc`** is set to `ON`. Tracking can be disabled by setting this parameter to `OFF`. By default, this parameter is set to `ON`.

This parameter enables the collection of statistics for **`pgx_stat_gmc`**. The **`pgx_stat_gmc`** command displays statistics that are related to the GMC hit ration and size of the GMC area in the shared memory.

Example 8-36 displays the output of **`pgx_stat_gmc`**.

Example 8-36 Output of pgx_stat_gmc view

```
[fsepuser@rdbkpgr3 ~]$ psql -d postgres -p 27503 -c "select * from pgx_stat_gmc"
  searches  |    hits    | size  |          stats_reset
-----------+------------+--------+-------------------------------
 1509755787 | 1509466534 | 696568 | 2020-11-17 04:16:26.247632-05
(One row)
```

These statistics show that the cache hit ratio is 99.9%, and the size of the GMC in the shared memory is approximately 697 KB.

9

Backup, recovery, and monitoring

In Chapter 5, “High availability and high reliability architectures” on page 165 and Chapter 6, “Connection pooling and load balancing with Pgpool-II” on page 233, we designed and implemented a high availability (HA) FUJITSU Enterprise Postgres Advanced Edition database cluster that has load distribution across read-replica databases and disaster preparedness that is supported by a backup database instance. We configured this architecture on two IBM LinuxONE machines across three logical partitions (LPARs) that have eight virtual machines (VMs) that are running as guests in z/VM. In this chapter, we demonstrate the use cases for backup and typical recovery scenarios in an HA configuration.

This chapter discusses the following topics:

- Backup and restore concepts for FUJITSU Enterprise Postgres
- Different types of backup methods that are available with FUJITSU Enterprise Postgres
- The **`pg_rman`** open source software (OSS) peripheral tool for backup and easy point-in-time recovery (PITR)
- Use cases of backup and recovery by using **`pgx_dmpall`** and **`pg_rman`** utilities for full and incremental backups
- Monitoring of database instances by using **`pgbadger`** and **`pg_statsinfo`**
- Database performance monitoring by using **`pg_hint_plan`** and **`pg_dbms_stats`**

This chapter covers the following topics:

- Backup and recovery overview
- Optimizer hints
- Locked statistics
- Monitoring

9.1 Backup and recovery overview

In this section, we provide an overview of methods that can be used to back up and recover your databases.

9.1.1 Logical and physical backups

There are two types of backup methods that are available with PostgreSQL:

- Logical backup: A method to take the backup of the data that is stored in the logical database objects by extracting the data and then storing it on a file. This type of backup method provides the flexibility of backing up data from the database objects and using it for restoring data on other database instances.
- Physical backup: A method to back up all the database files that make up the database.

Depending upon the use case, both logical and physical backup methods are used. It is important to understand the differences and capabilities of the two backup methods to plan and implement a backup strategy in your organization:

- While when using logical backups, you can recover an entire database. The recovery process can be slow because the database instance must run all the queries that are required to restore the data.
- Physical backups are faster compared to logical backups when the entire database is recovered because a physical backup is at the file level, which makes the recovery faster compared to the logical backup.
- The performance of both backup types depends on the available computer, memory, and type of storage, and the network speed if the backup and recovery operation occurs over the network.

We now look at different ways in which logical and physical backup methods, as shown in Figure 9-1 on page 313, can be used with FUJITSU Enterprise Postgres.

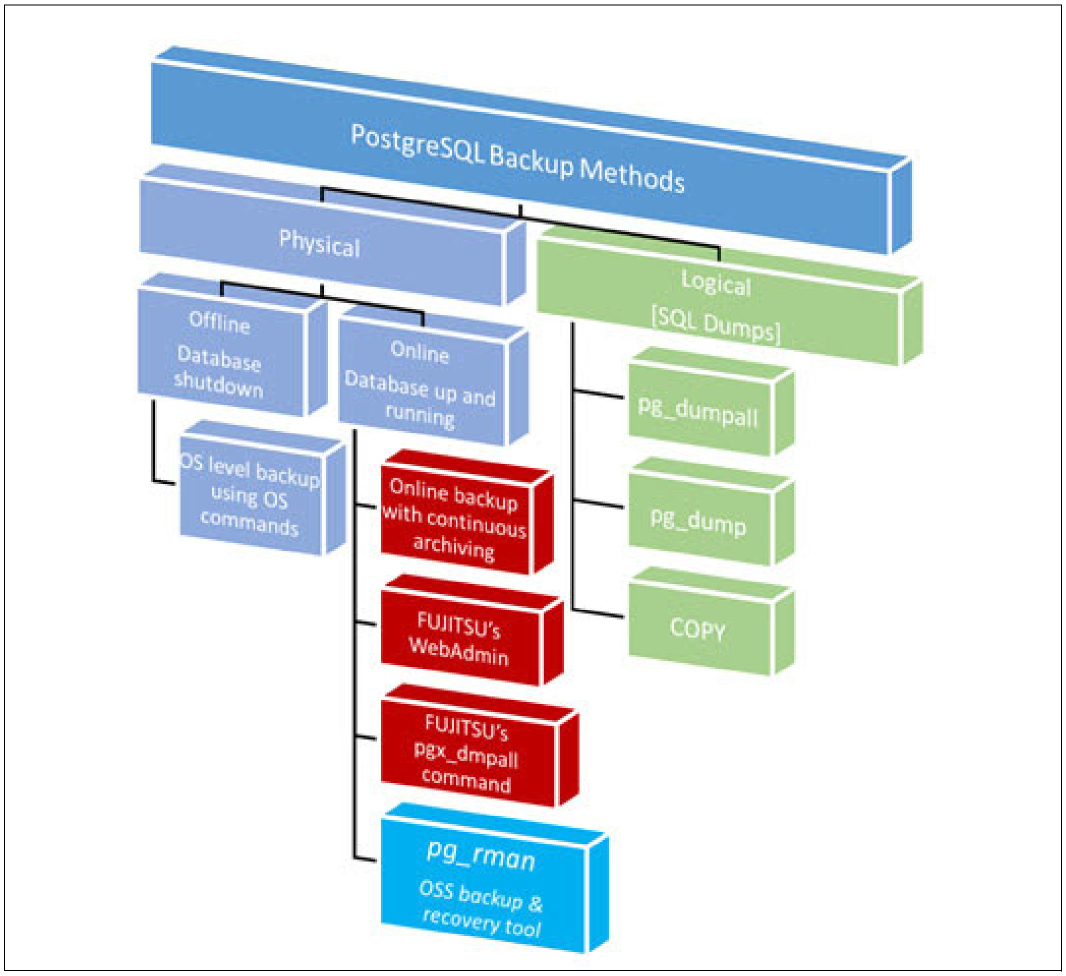

PostgreSQL Backup Methods
Physical
Logical
[SQL Dumps]
Offline
Database shutdown
Online
Database up and running
OS level backup using OS commands
Online backup with continuous archiving
FUJITSU's WebAdmin
FUJITSU's pgx_dmpall command
pg_rman
OSS backup & recovery tool
pg_dumpall
pg_dump
COPY

Figure 9-1 FUJITSU Enterprise Postgres and PostgreSQL backup methods

9.1.2 Logical backup

A logical backup works while the database is running. Because it works at the object level, the availability of the database instance is important for taking a logical backup.

A logical backup can be performed by using any of three different methods:

- By using the **pg_dumpall** command:
  - The **pg_dumpall** command is available as part of PostgreSQL that back ups the whole database cluster in SQL format.
  - The recovery is done by using the **psql** command.
- By using the **pg_dump** command:
  - The **pg_dump** command can be used to back up a specified database in SQL format. With **pg_dump**, the backup is always consistent because it uses a single transaction to back up the data, so it is not impacted by the transactions that are initiated later.
  - The recovery method is either running a **psql** command (a backup in script file format) or the **pg_restore** command (backup in archive file format).

- By using the `COPY` command:
  - The `COPY` command can be used to move data from database objects to a file in different formats of your choice, and the data can be moved from the file to the database objects.
  - The recovery method uses the `COPY` command for copying the data back from the files to the database instance.
  - The `COPY` command takes only data dumps; it does not capture schema definitions.

Note: When the logical backup is taken with any of these methods, the recovery is always to the backup point and not PiT.

9.1.3 Physical backup

As the name suggests, a physical backup is a copy of the database files. Unlike logical backup, a physical backup can either be performed in offline or online mode.

Offline backup

With this backup method, while the FUJITSU Enterprise Postgres instance is stopped, the entire database is backed up by copying all the files and by using operating system (OS) level commands such as `cp`, `scp`, or `rsync`. To have a consistent backup with this method, you must stop the database instance.

The recovery of the backup data is done by using the OS-level commands to the point when the backup was performed, and then the database instance is started again.

Online backup

It may not always be possible to stop the database and create an offline backup. For those scenarios, you can use online backup methods. In an online backup, the database instance is not stopped and it keeps running. PostgreSQL uses a combination of full database backup and transaction logs for capturing a consistent online backup of the running database instance.

The transaction logs (also known as Write-Ahead-Logs (WALs) are archived as *archive logs*.

The recovery of the database (for example, in the event of data corruption) is performed by recovering the data by using the full base backup followed by applying the archive logs and the WAL records.

Online backup with continuous archiving is a feature in PostgreSQL. In this method, the database instance is switched to backup mode, and then the database files are copied by using the OS-level copy function. After the copy completes, the database instance is returned from the backup mode. For the recovery of the data, it is possible to recover the database instance to a PiT by using the procedure for PITR.

Note: For more information about PITR, see Recovering Using a Continuous Archive Backup.

In addition to the online backup function of PostgreSQL, FUJITSU Enterprise Postgres uses the following two features that are supported by the WebAdmin GUI and the Fujitsu developed command **`pgx_dmpall`**:

- WebAdmin: The FUJITSU WedAdmin GUI can perform online backups with continuous archiving with a single click. Furthermore, after the backup is complete, the WAL is duplicated and stored at two locations that are defined on the data storage destination disk and the backup data storage destination disk. WebAdmin also provides the facility to recover the database PiT with its GUI.

Note: For more information about using WebAdmin for backup, see 3.2.1, "Using WebAdmin", in *FUJITSU Enterprise Postgres Operation Guide*.

- The **`pgx_dmpall`** command: The FUJITSU **`pgx_dmpall`** command uses the backup of the FUJITSU Enterprise Postgres data directories, table spaces, and configuration files. The backup data is stored in the directory that is specified by the **`backup_destination`** parameter in the `postgresql.conf` file. The command also deletes the archive logs that are no longer required after the backup completes. The process for **`pgx_dmpall`** is shown in Figure 9-2.

  The **`pgx_dmpall`** command can also be integrated with automation software to perform periodic backups. To support recovery, the FUJITSU **`pgx_rcvall`** command can be used to perform PITR by using a restore point.

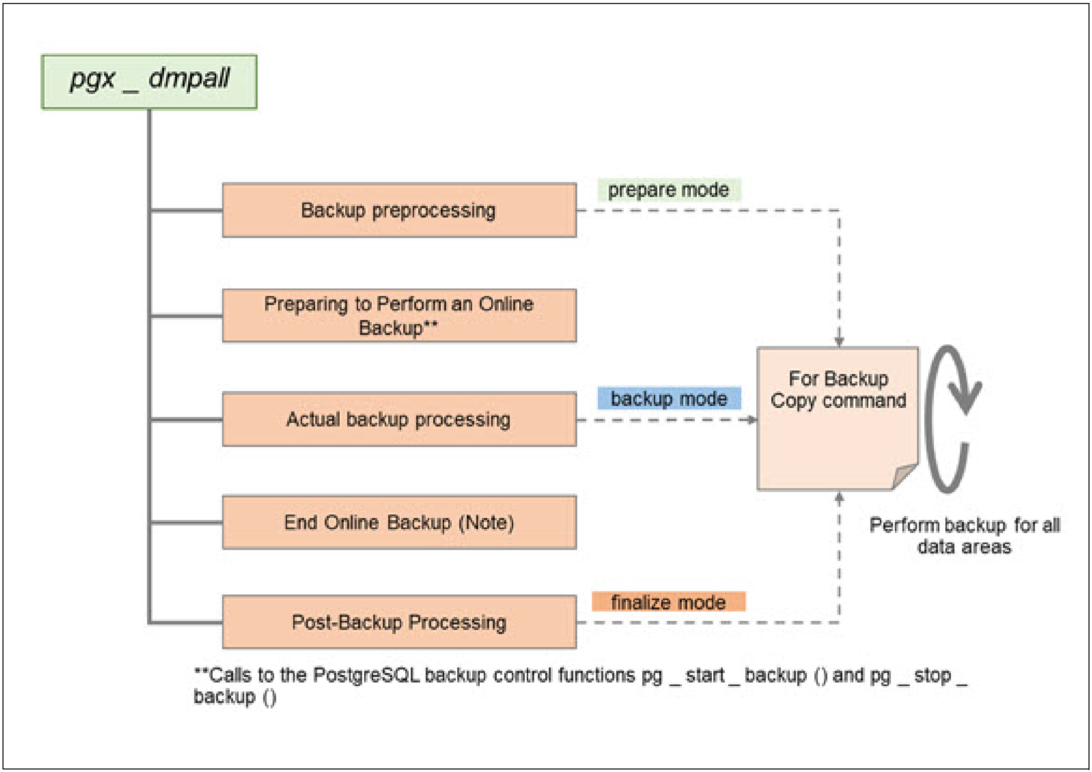

pgx _ dmpall
Backup preprocessing
prepare mode
Preparing to Perform an Online Backup**
Actual backup processing
backup mode
For Backup Copy command
End Online Backup (Note)
Perform backup for all data areas
Post-Backup Processing
finalize mode
**Calls to the PostgreSQL backup control functions pg _ start _ backup () and pg _ stop _ backup ()

Figure 9-2 FUJITSU pgx_dmpall backup processing steps

In this section, we briefly described the backup and recovery methods and the tools and commands that support logical and physical backup and recovery.

FUJITSU Enterprise Postgres also supports the OSS backup and recovery tool **`pg_rman`**, which comes packaged with FUJITSU Enterprise Postgres.

9.1.4 The pg_rman command for backup and recovery

The **pg_rman** command is an open source software utility for PostgreSQL backup and recovery. This utility can take a physical online backup of the database cluster, archive logs, and the server logs. PITR is a convenient feature that allows you to freely set a recovery point, but it is cumbersome to prepare for recovery. By using **pg_rman** to perform PITR, this process is simplified.

The pg_rman features

Seven functions, which are shown in Figure 9-3, are available through the **pg_rman** command. The following features of the **pg_rman** utility use those functions:

- Online backup and recovery with one command.
- Provides options to take incremental backups with data compression.
- Organizes and manages backup versions, and provides a list of the backup that is performed with backup metadata.
- Provides the capability for storage snapshots to reduce backup time.

A single **pg_rman** command can manage the backup and recovery and the backup/recovery metadata.

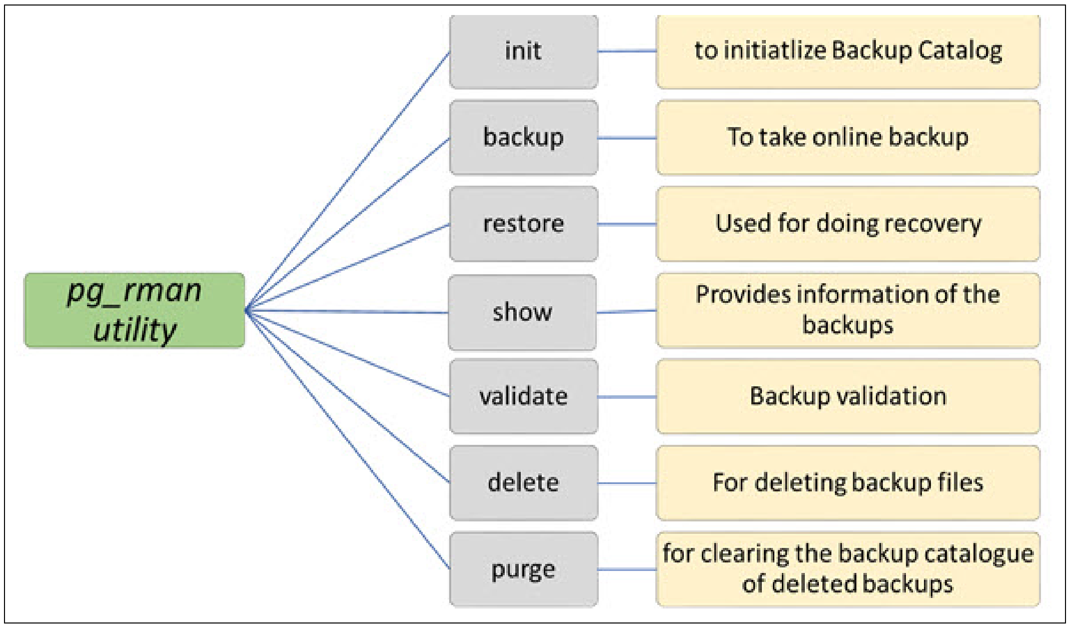

pg_rman utility
init
to initiatlize Backup Catalog
backup
To take online backup
restore
Used for doing recovery
show
Provides information of the backups
validate
Backup validation
delete
For deleting backup files
purge
for clearing the backup catalogue of deleted backups

Figure 9-3 The pg_rman command options for various functions

Figure 9-4 on page 317 shows a typical scenario when an application with frequent updates that is being backed up as a **pg_rman** full backup and with incremental functions suffers a failure. Figure 9-4 on page 317 shows the recovery process, which involves combining full backup, incremental backup, archive logs, and the latest WAL records to run recovery to the latest state just before the failure occurred. The **pg_rman** command also provides the facility to recover the data to a specific PiT known as PITR.

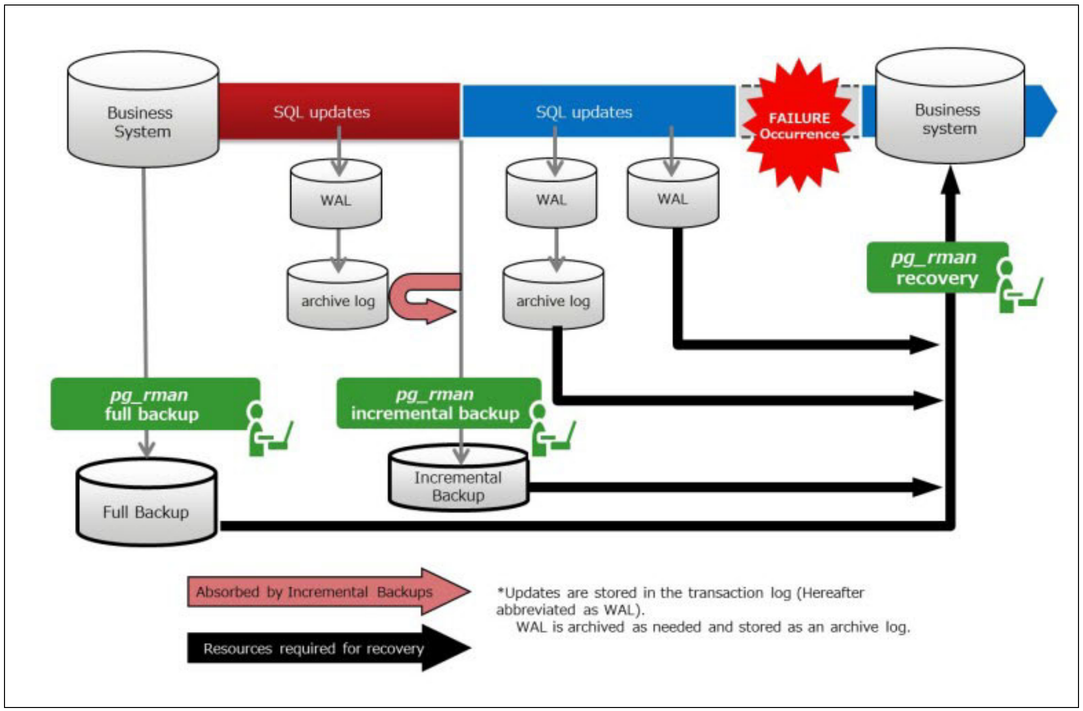

Business System
SQL updates
SQL updates
FAILURE Occurrence
Business system
WAL
WAL
WAL
archive log
archive log
pg_rman recovery
pg_rman full backup
pg_rman incremental backup
Full Backup
Incremental Backup
Absorbed by Incremental Backups
Resources required for recovery
*Updates are stored in the transaction log (Hereafter abbreviated as WAL).
WAL is archived as needed and stored as an archive log.

Figure 9-4 The pg_rman backup and recovery process with online full backup and incremental backup

Comparing pg_rman with PostgreSQL

Table 9-1 compares OSS PostgreSQL and the **pg_rman** backup and recovery procedure.

Table 9-1 Comparing pg_rman with the PostgreSQL backup and recovery method

Process	PostgreSQL standard features	pg_rman
Backup	1. Make a base backup by using the **pg_basebackup** command. 2. Verify that the base backup data was generated. 3. Check whether a WAL was generated.	1. Make a full (Increment) backup and verification by using the **pg_rman** command. 2. Use the **pg_rman** command to check the backup status.
Recovery	1. Restore the base backup data. 2. Remove the restored old WAL. 3. Restore the unarchived WALs. 4. Create a recovery signal. 5. Start the database server. 6. Check whether the data is current.	1. Restore by running the pg_rman command. 2. Start the database server. 3. Check whether the data is current.

Here are the advantages of **pg_rman** over the OSS PostgreSQL backup and recovery functions:

- With **pg_rman**, both the backup and recovery are possible by using a single command.
- PITR with **pg_rman** is simple and straightforward.
- The **pg_rman** command provides both a full online backup and incremental backup in a single utility command with options.
- The **pg_rman** command provides the information about backups that are taken and the ability to manage the clearing of obsolete backup information from the catalog.

9.1.5 Performing backup and recovery in a FUJITSU Enterprise Postgres high availability environment

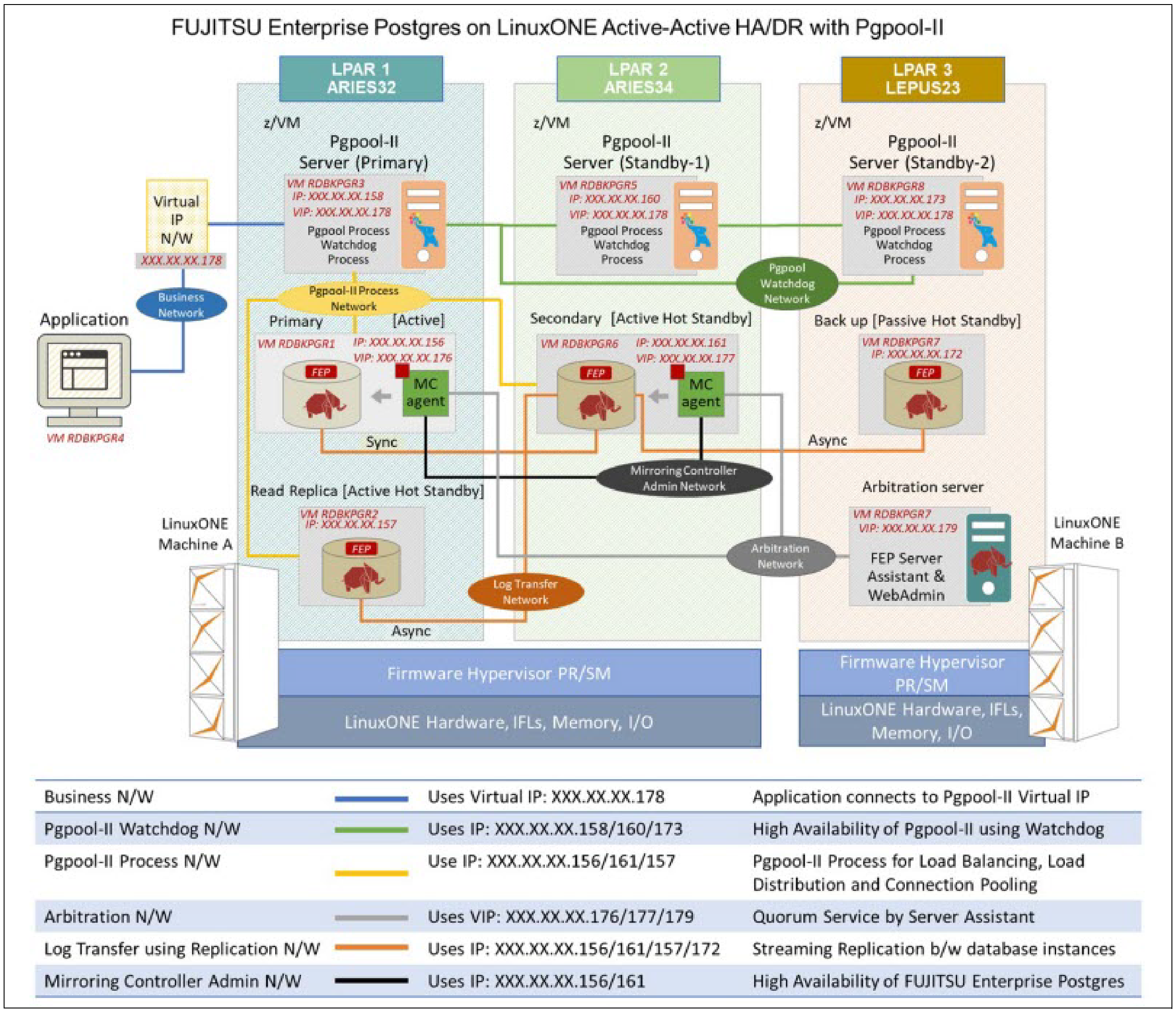
In this section, we show two uses cases of backup and recovery in a HA environment.

In the first use case, we describe the steps to configure a backup with continuous archiving, an online full backup that uses the **`pgx_dmpall`** backup command, and a PITR that uses the **`pgx_rcvall`** recovery command.

In the second use case, we describe the steps to configure a full and incremental backup by using **`pg_rman`**, and conduct the PITR by using the **`pg_rman`** restore command option.

Lab environment

We continue using the FUJITSU Enterprise Postgres HA environment that we built in Chapter 5, “High availability and high reliability architectures” on page 165 and Chapter 6, “Connection pooling and load balancing with Pgpool-II” on page 233.Figure 9-5 shows the HA environment that is used to demonstrate the backup and recovery use cases.



Business N/W	🟦	Uses Virtual IP: XXX.XX.XX.178	Application connects to Pgpool-II Virtual IP
Pgpool-II Watchdog N/W	🟩	Uses IP: XXX.XX.XX.158/160/173	High Availability of Pgpool-II using Watchdog
Pgpool-II Process N/W	🟨	Use IP: XXX.XX.XX.156/161/157	Pgpool-II Process for Load Balancing, Load Distribution and Connection Pooling
Arbitration N/W	⬜	Uses VIP: XXX.XX.XX.176/177/179	Quorum Service by Server Assistant
Log Transfer using Replication N/W	🟧	Uses IP: XXX.XX.XX.156/161/157/172	Streaming Replication b/w database instances
Mirroring Controller Admin N/W	⬛	Uses IP: XXX.XX.XX.156/161	High Availability of FUJITSU Enterprise Postgres

Figure 9-5 FUJITSU Enterprise Postgres high availability configuration example on IBM LinuxONE

Our configuration consists of the following items:

- Two IBM LinuxONE machines: Machine A and Machine B.
- Three LPARs: LPAR 1, 2, and 3, each running z/VM:
  - LPAR 1: Three VMs running Red Hat Enterprise Linux Server guests.
  - LPAR 2: Two VMs running Red Hat Enterprise Linux Server guests.
  - LPAR 3: Two VMs running Red Hat Enterprise Linux Server guests.
- FUJITSU Enterprise Postgres Advanced Edition HA with the following configuration:
  - Primary database server with Mirroring Controller (MC) (labeled as MC Agent) on VM RDBKPGR1 on LPAR1 ARIES32 on IBM LinuxONE Machine A.
  - Secondary database server with MC (labeled as MC Agent) on VM RDBKPGR6 on LPAR2 ARIES34 on IBM LinuxONE Machine A in Hot Standby mode with synchronous streaming replication with a primary database instance.
  - Arbitration server running FUJITSU Enterprise Postgres Server Assistant software on VM RDBKPGR7 on LPAR3 LEPUS23 on IBM LinuxONE Machine B for providing quorum service to primary and secondary database instances.
  - A backup database instance on VM RDBKPGR7 on LPAR3 LEPUS23 and IBM LinuxONE with Machine B in a passive hot standby mode. Because the instance is running in passive mode, the application does not have connectivity to the backup database instance. The instance is also a hot-standby instance because it is connected to the secondary database instance through asynchronous streaming replication. The arbitration server is also installed on VM RDBKPGR7.
- A read-replica database instance on VM RDBKPGR2 on LPAR1 ARIES32 on IBM LinuxONE Machine A. The read-replica is load balanced with the secondary database instance for read-only queries. Load balancing is provided by Pgpool-II.
- Pgpool-II HA configuration: Pgpool-II is configured in a three-unit HA architecture:
  - Pgpool-II primary installed and configured on VM RDBKPGR3 on LPAR1 ARIES32 on IBM LinuxONE Machine A.
  - Pgpool-II standby-1 installed and configured on VM RDBKPGR5 on LPAR2 ARIES34 on IBM LinuxONE Machine A.
  - Pgpool-II standby-2 installed and configured on VM RDBKPGR8 on LPAR3 LEPUS23 on IBM LinuxONE Machine B.

Note: This chapter assumes that you implemented the FUJITSU Enterprise Postgres HA architecture that is described in Chapter 5, “High availability and high reliability architectures” on page 165.

Backup and recovery by using the pgx_dmpall and pgx_rcvall commands in an HA database cluster

In this section, we describe an application running on a FUJITSU Enterprise Postgres Advanced Edition database cluster. To create a recovery situation, we simulate a data loss by truncating an application table to create a requirement to recover the database. Outside of the lab environment, you use the recovery method for recovering the whole FUJITSU Enterprise Postgres Database cluster or there might be database corruption across the cluster.

Here are the steps to prepare the FUJITSU Enterprise Postgres Advanced Edition database cluster for backup and recovery:

1. On the primary database instance on VM RDBKPGR1 with an IP address of xxx.xx.xx.156 on LPAR ARIES32, we configured the backup with continuous archiving by updating the **postgresql.conf file** for parameters **backup_destination**, **archive_mode**, and **archive_command**, as shown in Example 9-1.

Example 9-1 Configuring a backup with continuous archiving: Update parameters (1 of 2)

```
$ vim /database/inst1/postgresql.conf
backup_destination = '/backup/inst1'
archive_mode = on
archive_command = '/opt/fsepv12server64/bin/pgx_walcopy.cmd "%p"
"/backup/inst1/archived_wal/%f"'
```

2. On the secondary database instance on VM RDBKPGR6 with IP as xxx.xx.xx.161 on LPAR ARIES34, we configured the backup with continuous archiving by updating the `postgresql.conf` file for parameters **backup_destination**, **archive_mode**, and **archive_command**, as shown in Example 9-2.

Example 9-2 Configuring a backup with continuous archiving: Update parameters (2 of 2)

```
$ vim /database/inst1/postgresql.conf
backup_destination = '/backup/inst1'
archive_mode = on
archive_command = '/opt/fsepv12server64/bin/pgx_walcopy.cmd "%p"
"/backup/inst1/archived_wal/%f"'
```

3. Stop the primary and secondary database instances by running the MC **stop** command on the primary database instance on VM RDBKPGR1. The **-a** option stops both the primary and secondary database instances from the primary database node by using the following command:

   ```
   $ mc_ctl stop -M /mcdir/inst1 -a
   ```

4. Start the primary database instance by running the MC **start** command on the primary database VM RDBKPGR1 by using the following command:

   ```
   $ mc_ctl start -M /mcdir/inst1
   ```

5. Start the secondary database instance by running the MC **start** command on the secondary database VM RDBKPGR6 by using the following command:

   ```
   $ mc_ctl start -M /mcdir/inst1
   ```

6. On the primary database instance, initialize the **pgbench** database by using the following command. The **pgbench** database is used for performing backup and recovery.

   ```
   $ pgbench -i postgres -s 10
   ```

7. After the **pgbench** database is populated, verify the number of rows in the **pgbench_accounts** table, as shown in Example 9-3 on page 321. Do this task first on the primary database instance on VM RDBKPGR1, and then on the secondary, read-replica, and backup database instances.

Example 9-3 Verifying the number of rows

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "select count(*) from 
pgbench_accounts"
  count
---------
 1000000
(One row)
```

Note: Run the command that is shown in Example 9-3 on the secondary, read-replica, and the backup database instances to verify that the data was replicated to all standby instances.

8. On the primary database instance (VM RDBKPGR1) and secondary database instance (VM RDBKPGR6), take a full database backup by using the **`pgx_dmpall`** command, as shown in Example 9-4.

Example 9-4 Taking a full database backup

```
$ pgx_dmpall -D /database/inst1
starting backup of database and configuration files...
starting backup mode...
copying data files...
waiting for necessary transaction log files to be archived...
backup of database and configuration files completed successfully (16063)
```

Note: Run the command that is shown in Example 9-4 on the secondary database standby instance to back up the database and configuration files because it is not definite as to which server runs as the primary server. You want to ensure that the **`pgx_dmpall`** command is run periodically on all servers so that all the data is backed up.

9. On the primary database instance (VM RDBKPGR1) and secondary database instance (VM RDBKPGR6), we back up the MC configuration by using the OS copy command:

   ```
   cp -p /mcdir/inst1/*.conf /backup/
   ```

10. On the primary database instance (VM RDBKPGR1), we create a restore point by using **`pg_create_restore_point`** with the name `BeforeAppMaintenance`, as shown in Example 9-5.

Example 9-5 Creating a restore point

```
Create a restore point
$ psql -d postgres -U fsepuser -h xxx.xx.xx.156 -p 27500  -c "select 
pg_create_restore_point('BeforeAppMaintenance');"
 pg_create_restore_point
-------------------------
 E/6EB4A1E0
(One row)
```

Note: The function **`pg_create_restore_point`** returns the log sequence number that is represented by a discrete data type `pg_lsn`, which is used in various WAL operations. For more information, see WAL Internals.

11. Simulate a data loss by truncating the **`pgbench_accounts`** table by running the following command on the primary database instance RDBKPGR1:

    ```
    $ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "truncate
    pgbench_accounts"
    ```

12. Confirm that the data loss that is simulated in step 11 is replicated across the standby database instances, which are secondary-standby (VM RDBKPGR6), read-replica (VM RDBKPGR2), and backup-standby (VM RDBKPGR7) database instances by selecting rows from the **`pgbench_accounts`** table, as shown in Example 9-6.

Example 9-6 Confirming data

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "select count(*) from
pgbench_accounts"
 count
-------
     0
(One row)
```

Note: We run the SQL SELECT in Example 9-6 on all the database instances in the FUJITSU Enterprise Postgres Database cluster.

13. Having confirmed that there is data loss, perform database recovery by using the full backup that was taken in step 8 on page 321. To start the recovery process, stop the MC and the primary and secondary database instances by running the following command on the primary database VM RDBKPGR1. The **`-a`** option in the **`mc_ctl stop`** command conducts the MC stop on the secondary database instance too.

    ```
    $ mc_ctl stop -a -M /mcdir/inst1
    ```

14. Stop the read-replica (on VM RDBKPGR2) and backup-standby database instance (on VM RDBKPGR7). We run the following command one by one on both the read-replica and backup-standby database instances:

    ```
    $ pg_ctl stop -D /database/inst1
    ```

15. You are now ready to perform the recovery by using the **`pgx_rcvall`** command on the primary database server, RDBKPGR1, as shown in Example 9-7.

Example 9-7 Performing the recovery

```
$ pgx_rcvall -D /database/inst1 -B /backup/inst1 -n BeforeAppMaintenance
starting recovery of database...
copying data files...
waiting for server to start.... done
server started
waiting for server to shut down.... done
server stopped
Recovery of database completed successfully. (16090)
```

16. After the recovery completes successfully, start the MC and database instance on the primary database server, RDBKPGR1, by running the following command:

    ```
    $ mc_ctl start -M /mcdir/inst1
    ```

17. Confirm that the data was successfully recovered by running the SQL **`SELECT`** on the **`pgbench_accounts`** table on the primary database instance, as shown in Example 9-8 on page 323.

Example 9-8 Confirming that the data was recovered

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "select count(*) from
pgbench_accounts"
  count
---------
 1000000
(One row)
```

After recovering the primary database instance from full physical backup, the existing configurations on the secondary-standby, read-replica, and the backup-standby instances become obsolete. To bring them in sync with the primary database instance, build the standby database instances from scratch.

18. On the secondary-standby, read-replica, and the backup-standby database instances, remove all the obsolete databases, transactions, and backup files by using the command that is shown in Example 9-9.

Example 9-9 Removing obsolete databases

```
$ rm -rf /database/inst1/*
$ rm -rf /transaction/inst1/*
$ rm -rf /backup/inst1/*
```

19. Rebuild the HA configuration for adding the standby database instances, as it was before the recovery by performing the steps in “Installing and configuring the secondary database server on an IBM LinuxONE guest” on page 193 and “Installing and configuring a read-replica database instance and backup database instance with a cascading secondary database (standby) instance” on page 197.
20. Confirm that the restored data can be queried from the standby instances too, as shown in Example 9-10.

Example 9-10 Confirming the data

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "select count(*) from
pgbench_accounts"
  count
---------
 1000000
(One row)
```

In this section, we described the use case for the Fujitsu online physical backup utility **pgx_dmpall** and the recovery utility **pgx_rcvall** in a HA environment while using a FUJITSU Enterprise Postgres Advanced Edition database cluster.

Backup and recovery by using pg_rman in a FUJITSU Enterprise Postgres HA database cluster

In this section, we describe a use case of an application running on a FUJITSU Enterprise Postgres Advanced Edition HA database cluster. We use **pg_rman** for performing periodic physical online full backup and incremental backups. To create a recovery scenario, we simulate data loss by truncating an application table. In a real world scenario, you use the recovery method for recovering the whole FUJITSU Enterprise Postgres Database cluster because there might be database corruption across the cluster.

To prepare the FUJITSU Enterprise Postgres Advanced Edition database cluster for backup and recovery by using **pg_rman**, complete the following steps.

> **Note:** This chapter assumes that you implemented the FUJITSU Enterprise Postgres HA architecture that is described in “Installing and configuring the arbitration server on an IBM LinuxONE guest” on page 188, “Installing and configuring the secondary database server on an IBM LinuxONE guest” on page 193, and “Installing and configuring a read-replica database instance and backup database instance with a cascading secondary database (standby) instance” on page 197.

1. After you create the FUJITSU Enterprise Postgres HA database cluster, on the primary database instance on RDBKPGR1 with IP as xxx.xx.xx.156 on LPAR ARIES32, configure the **pg_rman** utility as shown in the following command:

   ```
   # cp -r /opt/fsepv12server64/OSS/pg_rman/* /opt/fsepv12server64
   ```

2. Next, on the primary (VM RDBKPGR1) and secondary database instances (VM RDBKPGR6), we prepare the **pg_rman** directories, as shown in Example 9-11.

Example 9-11 Preparing the pg_rman directories

```
# mkdir -p /rman_backup/inst1
# chown -R fsepuser:fsepuser /rman_backup
# chmod 700 /rman_backup/inst1
# mkdir -p /archive/inst1
# chown -R fsepuser:fsepuser /archive
# chmod 700 /archive/inst1
```

3. On the primary (RDBKPGR1) and secondary (RDBKPGR6) database instances, configure the `postgresql.conf` parameters **log_directory**, **archive_mode**, and **archive_command**, as shown in Example 9-12.

Example 9-12 Configuring parameters

```
$ vim /database/inst1/postgresql.conf
log_directory = 'log'
archive_mode = on
archive_command = 'cp "%p" "/archive/inst1/%f"'
```

4. Stop the primary and secondary database instances by running the MC **stop** command on the primary database instance on RDBKPGR1, as shown by the following command. The **-a** option stops both the primary and secondary database instances from the primary database node.

   ```
   $ mc_ctl stop -M /mcdir/inst1 -a
   ```

5. Start the primary database instance by running the MC **start** command on the primary database VM RDBKPGR1 by using the following command:

   ```
   $ mc_ctl start -M /mcdir/inst1
   ```

6. Start the secondary database instance by running the MC **start** command on the secondary database RDBKPGR6 by using the following command:

   ```
   $ mc_ctl start -M /mcdir/inst1
   ```

7. On the primary database server (RDBKPGR1), initialize the **pg_rman** backup catalog, as shown in Example 9-13 on page 325.

Example 9-13 Initializing the backup catalog

```
$ pg_rman init --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
INFO: ARCLOG_PATH is set to '/archive/inst1'
INFO: SRVLOG_PATH is set to '/database/inst1/log'
```

8. On the primary database instance, initialize the **pgbench** database by using the following command. The **pgbench** database is used as a sample application database for demonstrating backup and recovery.

   ```
   $ pgbench -i postgres -s 10
   ```

In the following steps, we simulate the use case for backup and recovery for the following events spread across three business days. The timelines and events are:

- Day 0:
  - Application updates that use **pgbench** on day 0
  - Full backup that uses the **pg_rman** backup option on day 0
- Day 1: Application updates that use **pgbench** and incremental backup on day 1
- Day 2:
  - Application updates that use **pgbench** and incremental backup on day 2
  - Data loss that is simulated by using a truncated table on day 2
  - PITR using **pg_rman** on day 2
  - Re-creating an HA database cluster through replication from the primary database instance

1. Day 0: On the primary database instance, we initiate the **pgbench** database by using the following command:

   ```
   $ pgbench -c 10 -t 100 postgres -n
   ```

2. Day 0: On the primary database instance, confirm that the **pgbench_history** table is populated with the data that is shown in Figure 9-15 on page 339.

Example 9-14 Confirming the population of the table

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "select count(*) from
pgbench_history"
 count
-------
  1000
(One row)
```

3. Day 0: Take a full online physical backup of the primary database instance by using **pg_rman**, as shown in Figure 9-16 on page 339.

Example 9-15 Taking a full online backup

```
$ pg_rman backup --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
--backup-mode=full -d postgres -p 27500
INFO: copying database files
INFO: copying archived WAL files
INFO: backup complete
INFO: Run 'pg_rman validate' to verify that the files are correctly copied.
```

4. Day 0: Validate the full online physical backup that was taken in Figure 9-16 on page 339 on the primary database instance by using **pg_rman**. Validation of backup is an important step, without which the backup data cannot be used for recovery by **pg_rman**. The backup that is taken at 2020-12-01 00:00:00 is validated (see Figure 9-17 on page 340).

Example 9-16 Validating a backup

```
$ pg_rman validate --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
INFO: validate: "2020-12-01 00:00:00" backup and archive log files by CRC
INFO: backup "2020-12-01 00:00:00" is valid
```

5. Day 0: Read the backup catalog to verify the backup details by using the **pg_rman show detail** command, as shown in Example 9-17.

Example 9-17 Reading the backup catalog

```
$ pg_rman show detail --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
===============================================================================
 StartTime           EndTime              Mode    Data  ArcLog  SrvLog   Total
Compressed  CurTLI  ParentTLI  Status
===============================================================================
2020-12-01 00:00:00  2020-12-01 00:00:02  FULL   182MB   184MB    ----   354MB
false       1          0  OK
```

6. Day 1: Simulate application transactions by using **pgbench** to add 1000 records to the pgbench_history table by using the following command:

   `$ pgbench -c 10 -t 100 postgres -n`

7. Day 1: Confirm that the row count is increased by 1000 records in the **pgbench_history** table that is shown in Example 9-16.

Example 9-18 Confirming the row count

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "select count(*) from
pgbench_history"
 count
-------
  2000
(One row)
```

8. Day 1: Perform an incremental backup by using the **pg_rman incremental mode** option for the backup, as shown in Example 9-19.

Example 9-19 Command to run an incremental backup with results

```
$ pg_rman backup --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
--backup-mode=incremental -d postgres -p 27500
INFO: copying database files
INFO: copying archived WAL files
INFO: backup complete
INFO: Run 'pg_rman validate' to verify that the files are correctly copied.
```

9. Day 1: Validate the full online physical backup that was taken in Example 9-19 on the primary database instance by using **pg_rman**. The validation of the backup is an important step, without which the backup data cannot be used for recovery by **pg_rman**. The backup that ws taken at 2020-12-02 00:00:00 is validated.

Example 9-20 Validating the backup

```
$ pg_rman validate --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
INFO: validate: "2020-12-02 00:00:00" backup and archive log files by CRC
INFO: backup "2020-12-02 00:00:00" is valid
```

10. Day 1: Read the backup catalog to verify the backup details by using the **pg_rman show detail** command option, as shown in Example 9-21.

Example 9-21 Reading the backup catalog

```
$ $ pg_rman show detail --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
=================================================================================
====================================
 StartTime           EndTime              Mode    Data  ArcLog  SrvLog   Total
Compressed  CurTLI  ParentTLI  Status
=================================================================================
====================================
2020-12-02 00:00:00  2020-12-02 00:00:02  INCR   158MB    33MB   ----   174MB
false       1          0  OK
2020-12-01 00:00:00  2020-12-01 00:00:05  FULL   182MB   184MB   ----   354MB
false       1          0  OK
```

Note: There are now two backup entries: one for a full backup that is taken on day 0 and an incremental backup that is taken on day 1.

11. Day 2: Simulate application transactions by using **pgbench** to add another 1000 records to the **pgbench_history** table by using the following command:

```
$ pgbench -c 10 -t 100 postgres -n
```

12. Day 2: Confirm that the row count is increased by 1000 records in the **pgbench_history** table, and that the count is now 3000 on day 2, as shown in Example 9-22.

Example 9-22 Confirming the row count

```
$ psql -d postgres -U fsepuser -h xxx.xx.xx.156 -p 27500  -c "select count(*) from
pgbench_history"
 count
-------
  3000
(One row)
```

13. Day 2: Perform an incremental backup by using the **pg_rman incremental mode** option for the backup, as shown in Example 9-23.

Example 9-23 Incremental backup command and results

```
pg_rman backup --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
--backup-mode=incremental -d postgres -p 27500
INFO: copying database files
INFO: copying archived WAL files
INFO: backup complete
INFO: Run 'pg_rman validate' to verify that the files are correctly copied.
```

14. Day 2: Validate the full online physical backup that was taken in Example 9-23 on page 327 on the primary database instance by using **pg_rman**. Validation of the backup is an important step, without which the backup data cannot be used for recovery by **pg_rman**. The backup that was taken at 2020-12-03 00:00:00 is validated, as shown in Example 9-24.

Example 9-24 Validating the backup

```
$ pg_rman validate --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
INFO: validate: "2020-12-03 00:00:00" backup and archive log files by CRC
INFO: backup "2020-12-03 00:00:00" is valid
```

15. Day 2: Read the backup catalog to verify the backup details by using the **pg_rman show detail** command option, as shown in Example 9-25.

Example 9-25 Reading the backup catalog

```
$ pg_rman show detail --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
==================================================================================
====================================
 StartTime           EndTime              Mode    Data  ArcLog  SrvLog   Total
Compressed  CurTLI  ParentTLI  Status
==================================================================================
====================================
2020-12-03 00:00:00  2020-12-03 00:00:03  INCR   159MB    33MB    ----   218MB
false       1          0  OK
2020-12-02 00:00:00  2020-12-02 00:00:02  INCR   158MB    33MB    ----   174MB
false       1          0  OK
2020-12-01 00:00:00  2020-12-01 00:00:05  FULL   182MB   184MB    ----   354MB
false       1          0  OK
```

Note: There are now three backup entries: one for a full backup taken on day 0, and two incremental backups taken on day 1 and day 2.

16. Day 2: Simulate a situation of data loss by truncating the **pgbench_history** table by running the following command on the primary database instance (RDBKPGR1):

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "truncate
pgbench_history"
```

17. Day 2: Confirm that the simulated data loss is replicated across the standby database instances, which are the secondary-standby (RDBKPGR6), the read-replica (RDBKPGR2), and the backup-standby (RDBKPGR7) database instances by selecting rows from the **pgbench_history** table on each one, as shown in Example 9-26.

Example 9-26 Confirming that the simulated data loss is replicated

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "select count(*) from
pgbench_history"
 count
-------
     0
(One row)
```

Note: Run the SQL that is shown in Example 9-26 on all database instances in the FUJITSU Enterprise Postgres Database cluster.

Having confirmed that there is data loss, perform a database recovery with the full backup and incremental backup to a PiT by using the **recovery-target-time** option of the **pg_rman restore** command that was taken on Day 0, Day 1, and Day 2 in the previous steps:

1. Day 2: To start the recovery process, stop the MC and the primary and secondary database instances by running the following command on the primary database VM RDBKPGR1. The **-a** option in the **mc_ctl stop** command stops the MC on the secondary database instance as well.

   ```
   $ mc_ctl stop -a -M /mcdir/inst1
   ```

2. Day 2: Stop the read-replica (RDBKPGR2) and backup-standby database instance (RDBKPGR7). Run the following command on each database instance:

   ```
   $ pg_ctl stop -D /database/inst1
   ```

3. Day 2: Run the recovery by using the **pg_rman restore** command on the primary database server (RDBKPGR1), as shown in Example 9-27. You are recovering to the latest consistent backup that is available. In this example, '2020-12-03 00:00:03' is used as the recovery target time, which corresponds closely to the last incremental backup taken on Day 2.

Example 9-27 Recovery to PiT

```
$ pg_rman restore --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
--recovery-target-time '2020-12-03 00:00:03'
INFO: the recovery target timeline ID is not given
INFO: use timeline ID of current database cluster as recovery target: 1
INFO: calculating timeline branches to be used to recovery target point
INFO: searching latest full backup that can be used as restore start point
INFO: found the full backup can be used as base in recovery: "2020-12-01 00:00:00"
INFO: copying online WAL files and server log files
INFO: clearing restore destination
INFO: validate: "2020-12-01 00:00:00" backup and archive log files by SIZE
INFO: backup "2020-12-01 00:00:00" is valid
INFO: restoring database files from the full mode backup "2020-12-01 00:00:00"
INFO: searching incremental backup to be restored
INFO: validate: "2020-12-02 00:00:00" backup and archive log files by SIZE
INFO: backup "2020-12-02 00:00:00" is valid
INFO: restoring database files from the incremental mode backup "2020-12-02
00:00:00"
INFO: validate: "2020-12-03 00:00:00" backup and archive log files by SIZE
INFO: backup "2020-12-03 00:00:001" is valid
INFO: restoring database files from the incremental mode backup "2020-12-03
00:00:00"
INFO: searching backup that contained archived WAL files to be restored
INFO: backup "2020-12-03 00:00:00" is valid
INFO: restoring WAL files from backup "2020-12-03 00:00:00"
INFO: restoring online WAL files and server log files
INFO: add recovery-related options to postgresql.conf
INFO: generating recovery.signal
INFO: restore complete
HINT: Recovery will start automatically when the PostgreSQL server is started.
```

4. After the recovery completes successfully, start the MC and database instance on the primary database server (VM RDBKPGR1) by using the following command:

   ```
   $ mc_ctl start -M /mcdir/inst1
   ```

5. Day 2: Confirm that the data was successfully recovered by running the **select count** SQL statement on the **pgbench_history** table on the primary database instance, as shown in Example 9-28. It should show the number of rows as 3000.

Example 9-28 Verifying that the data was recovered

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "select count(*) from
pgbench_history"
 count
-------
  3000
(One row)
```

6. Day 2: After confirming that the recovery was successful, open the database instance operations by resuming the WAL replay and by using the function **pg_wal_replay_resume()**, as shown in Example 9-29. This step finishes the recovery mode of the primary database instance.

Example 9-29 Resuming WAL

```
$  psql -d postgres -U fsepuser -h XXX.XX.XX.156 -p 27500  -c "select
pg_wal_replay_resume()"
 pg_wal_replay_resume
----------------------
(One row)
```

7. Day 2: Perform a full physical backup by using the **pg_rman** command on the primary database instance, as shown in Example 9-30.

Example 9-30 Running a full physical backup

```
$ pg_rman backup --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
--backup-mode=full -d postgres -p 27500
INFO: copying database files
INFO: copying archived WAL files
INFO: backup complete
INFO: Run 'pg_rman validate' to verify that the files are correctly copied.
```

8. Day 2: Validate the full online physical backup on the primary database instance by using the **pg_rman** command. Validation of the backup is an important step, without which the backup data cannot be used for recovery by **pg_rman**. In Example 9-31, the backup that was taken at 2020-12-03 12:00:00 was validated.

Example 9-31 Validating the backup

```
$ pg_rman validate --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
INFO: validate: "2020-12-03 12:00:00" backup and archive log files by CRC
INFO: backup "2020-12-03 12:00:00" is valid
```

9. Day 2: Read the backup catalog to verify the backup details by using the **pg_rman show detail** command option, as shown in Example 9-32 on page 331.

Example 9-32 Verifying the backup details

```
$ pg_rman show detail --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
==================================================================================
 StartTime           EndTime              Mode    Data  ArcLog  SrvLog   Total
Compressed  CurTLI  ParentTLI  Status
==================================================================================
2020-12-03 12:00:00  2020-12-03 12:00:05  FULL   183MB   134MB   ----   306MB
false       2          1  OK
2020-12-03 00:00:00  2020-12-03 00:00:03  INCR   159MB    33MB   ----   218MB
false       1          0  OK
2020-12-02 00:00:00  2020-12-02 00:00:02  INCR   158MB    33MB   ----   174MB
false       1          0  OK
2020-12-01 00:00:00  2020-12-01 00:00:05  FULL   182MB   184MB   ----   354MB
false       1          0  OK
```

Note: There should now be four backup entries: Two are the full backups that were taken on days 0 and 2, and two are incremental backups that were taken on days 1 and 2.

After recovering the primary database instance from a full physical backup, the existing configurations on the secondary-standby, read-replica, and the backup-standby instances become obsolete. To bring them in sync with the primary database instance, we build the standby database instances from scratch by completing the following steps:

1. Day 2: On the secondary-standby database instance, remove all the obsolete databases, transactions, backups, and archived log files by using the **rm** OS command, as shown in Example 9-33.

Example 9-33 Removing obsolete files

```
$ rm -rf /database/inst1/*
$ rm -rf /transaction/inst1/*
$ rm -rf /rman_backup/inst1/*
$ rm -rf /archive/inst1/*
```

2. Day 2: On the read-replica and the backup-standby database instance, remove all of the obsolete database and transactions files by using the *‘rm´* OS command as shown Example 9-34.

Example 9-34 Removing more obsolete files

```
$ rm -rf /database/inst1/*
$ rm -rf /transaction/inst1/*
```

3. Rebuild the HA configuration to add the standby database instances as they were before the recovery by doing the steps in “Installing and configuring the secondary database server on an IBM LinuxONE guest” on page 193 and “Installing and configuring a read-replica database instance and backup database instance with a cascading secondary database (standby) instance” on page 197. After the secondary-standby, read-replica, and backup-standby database instances are added, build the HA FUJITSU Enterprise Postgres Database cluster by confirming that the restored data can be queried from the standby instances, as shown in Example 9-35. Replace IP address xxx.xx.xx.xxx with the IP addresses of the secondary-standby, read-replica, and backup standby database instances.

Example 9-35 Confirming that the restored data can be queried from the standby instances

```
$ psql -d postgres -U fsepuser -h XXX.XX.XX.XXX -p 27500  -c "select count(*) from 
pgbench_history"
 count
-------
  3000
(One row)
```

In this section, we described the use of the **pg_rman** open source backup and recovery utility in the same HA environment. We showed a typical recovery scenario that might happen in an organization and used the various **pg_rman** options for backups, management, and monitoring of the backup and recovery process.

9.1.6 Managing FUJITSU Enterprise Postgres backups with IBM Spectrum Protect

In this section, we show how to manage FUJITSU Enterprise Postgres backups with IBM Spectrum Protect with the S3 API.

IBM Spectrum Protect is a centralized and automated data protection platform that helps to reduce data loss and manage compliance with data retention and availability requirements. It provides scalable data protection for physical file servers, applications, and virtual environments.

Organizations can scale up to manage billions of objects per backup server. They can also reduce backup infrastructure costs with built-in data efficiency capabilities and store data on tape, on-premises object storage, and public cloud services. IBM Spectrum Protect can also be a data offload target for IBM Spectrum Protect Plus to leverage your existing investment for long-term data retention and disaster recovery (DR).

Figure 9-6 on page 333 shows IBM Spectrum Protect Operations Center, which provides an advanced visual dashboard, built-in analytics, and integrated workflow automation features to dramatically simplify backup administration.

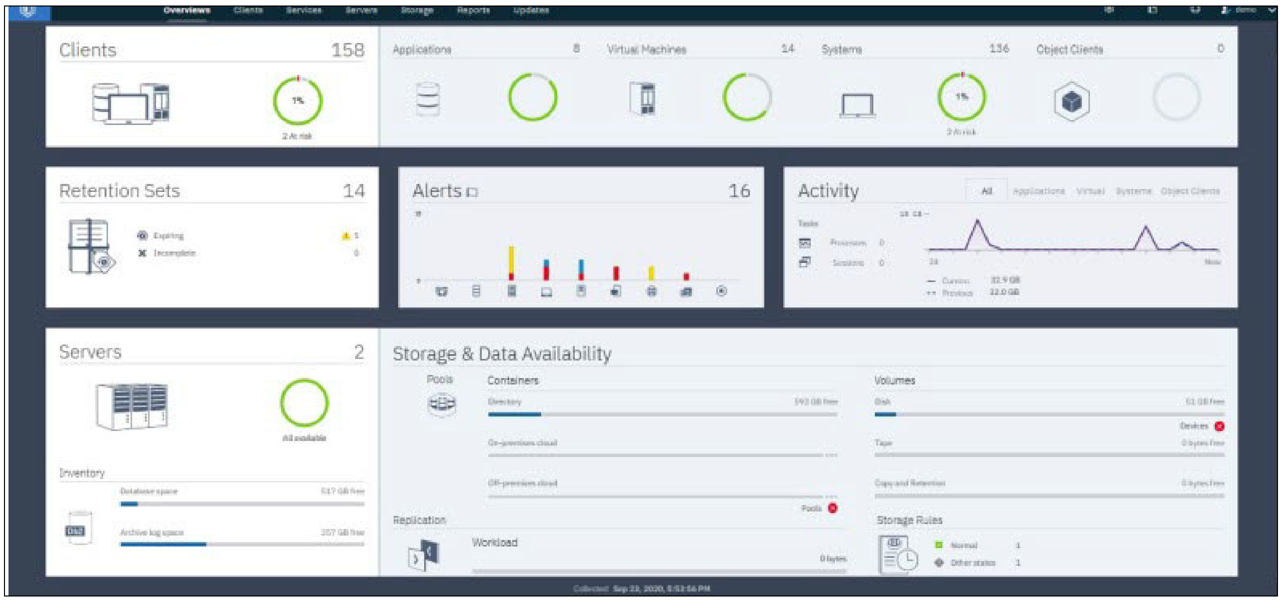


Figure 9-6 IBM Spectrum Protect Operations Center overview

To learn more about IBM Spectrum Protect and IBM Data Protection solutions, see IBM Spectrum Protect.

Managing IBM Spectrum Protect by using the Operations Center allows ease of use and breakthrough visibility to administrators that are tasked with backing up databases, which reduces the level of expertise that is required.

Integrating FUJITSU Enterprise Postgres and IBM Spectrum Protect helps database administrators to have an off-site backup of the database. IBM Spectrum Protect provides extra support to back up and restore by using the S3 client applications, which allows administrators to use existing backup tools and manage the database backups through S3 API calls.

Lab environment

To demonstrate this use case, we continue with **`pg_rman`** for performing the periodical physical backups of FUJITSU Enterprise Postgres and using the S3 client to manage the backups on IBM Spectrum Protect.

The lab environment that is used to demonstrate the IBM Spectrum Protect use case consists of the following items:

- Two IBM LinuxONE servers: IBM Spectrum Protect is running on one machine, and FUJITSU Enterprise Postgres is running on the other machine.
- IBM Spectrum Protect V8.1.11 server: IBM Spectrum Protect V8.1.11 supports data copy from the S3 client application to IBM Spectrum Protect. The S3 API support for S3 client applications is added in IBM Spectrum Protect V8.1.11.
- AWS command-line interface (CLI): AWS CLI as an S3 client application.
- FUJITSU Enterprise Postgres 12 Server.

Note: This chapter assumes that you configured **`pg_rman`** to performing physical backups, as described in “Backup and recovery by using pg_rman in a FUJITSU Enterprise Postgres HA database cluster” on page 323.

Managing the backups by using the AWS CLI and IBM Spectrum Protect

In this section, we describe the steps to configure IBM Spectrum Protect and manage database backups by using the S3 API through AWS CLI to IBM Spectrum Protect.

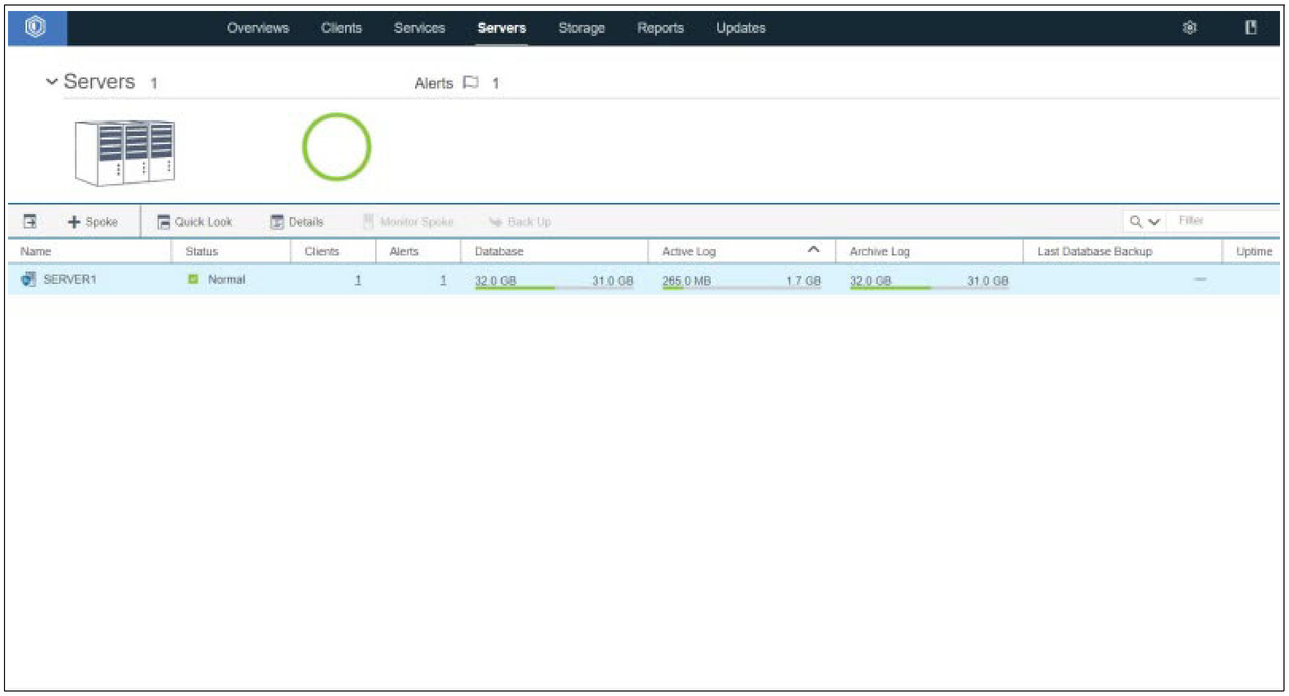
As part of the IBM Spectrum Protect configuration, we create an object agent on the IBM Spectrum Protect server and register an S3 client as an object client to the IBM Spectrum Protect server.

Note: The chapter topics assume that you installed an IBM Spectrum Protect server and are familiar with IBM Spectrum Protect Operation Center and CLI.

Complete the following steps:

1. On the IBM Spectrum Protect server, set up an *object agent* that is the gateway code that receives a request from an object client and converts it into commands that are understandable by IBM Spectrum Protect:

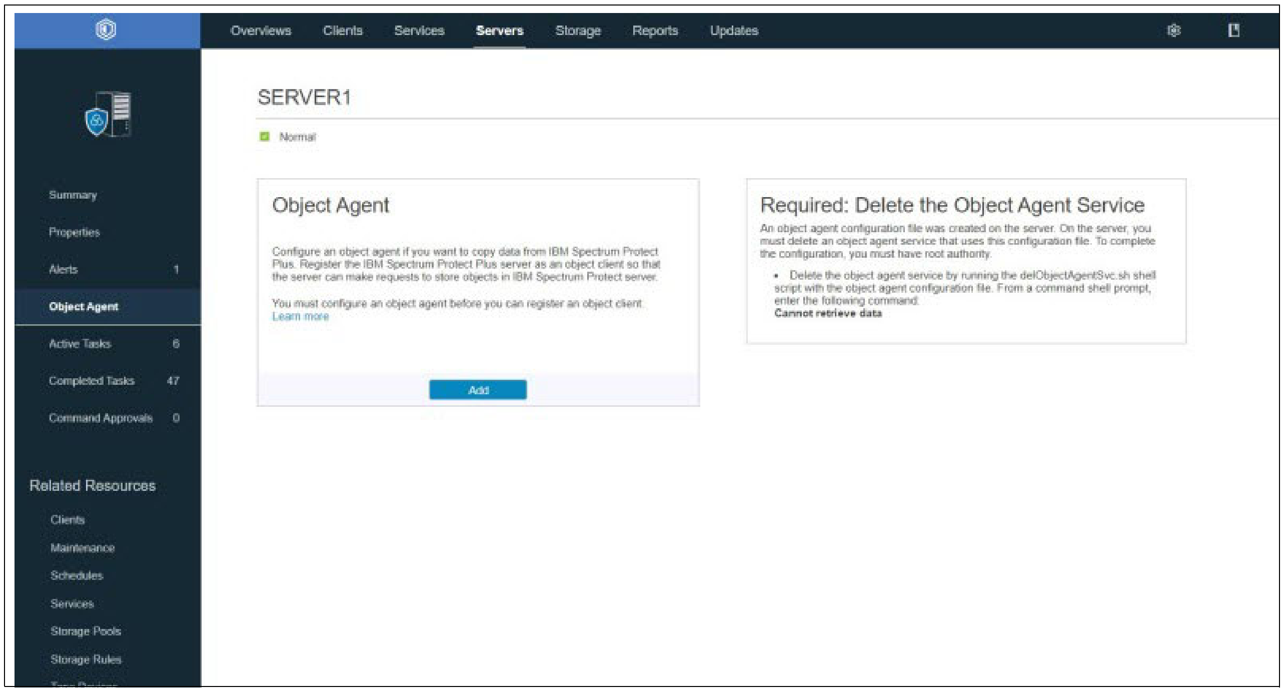
   a. As shown in Figure 9-7, select the server to start the creation of an object agent. Go to the **Servers** tab and select the server from the list.



Name	Status	Clients	Alerts	Database		Active Log		Archive Log		Last Database Backup	Uptime
SERVER1	Normal	1	1	32.0 GB	31.0 GB	265.0 MB	1.7 GB	32.0 GB	31.0 GB	—	

Figure 9-7 Selecting the server to create an object agent

   b. On the left side, click **Add** in the Object Agent pane, as shown in Figure 9-8 on page 335.



Object Agent

Configure an object agent if you want to copy data from IBM Spectrum Protect Plus. Register the IBM Spectrum Protect Plus server as an object client so that the server can make requests to store objects in IBM Spectrum Protect server.

You must configure an object agent before you can register an object client.

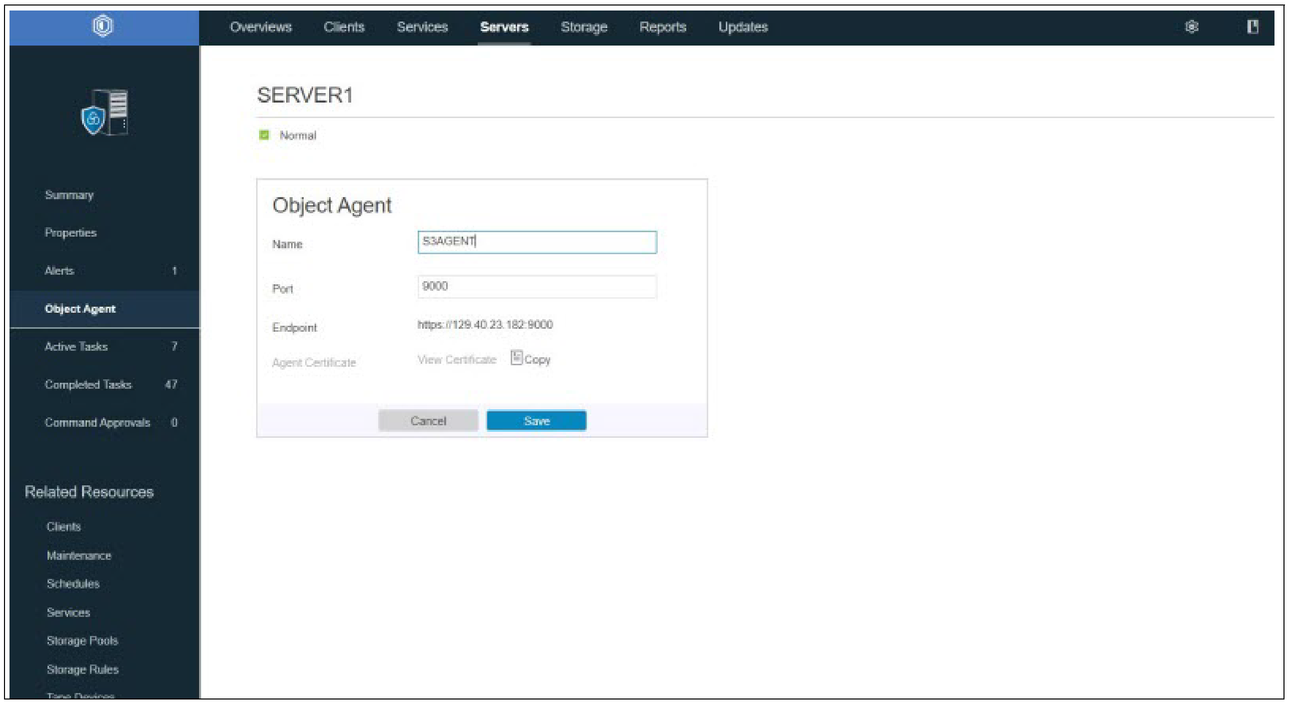
Required: Delete the Object Agent Service

An object agent configuration file was created on the server. On the server, you must delete an object agent service that uses this configuration file. To complete the configuration, you must have root authority.

- Delete the object agent service by running the delObjectAgentSvc.sh shell script with the object agent configuration file. From a command shell prompt, enter the following command:
  Cannot retrieve data

Figure 9-8 Adding an object agent to the selected server

c. Enter the object agent details (Figure 9-9). Our example shows that the object name is S3AGENT. Keep the default port of 9000 and click **Save**.



Object Agent

Name: S3AGENT

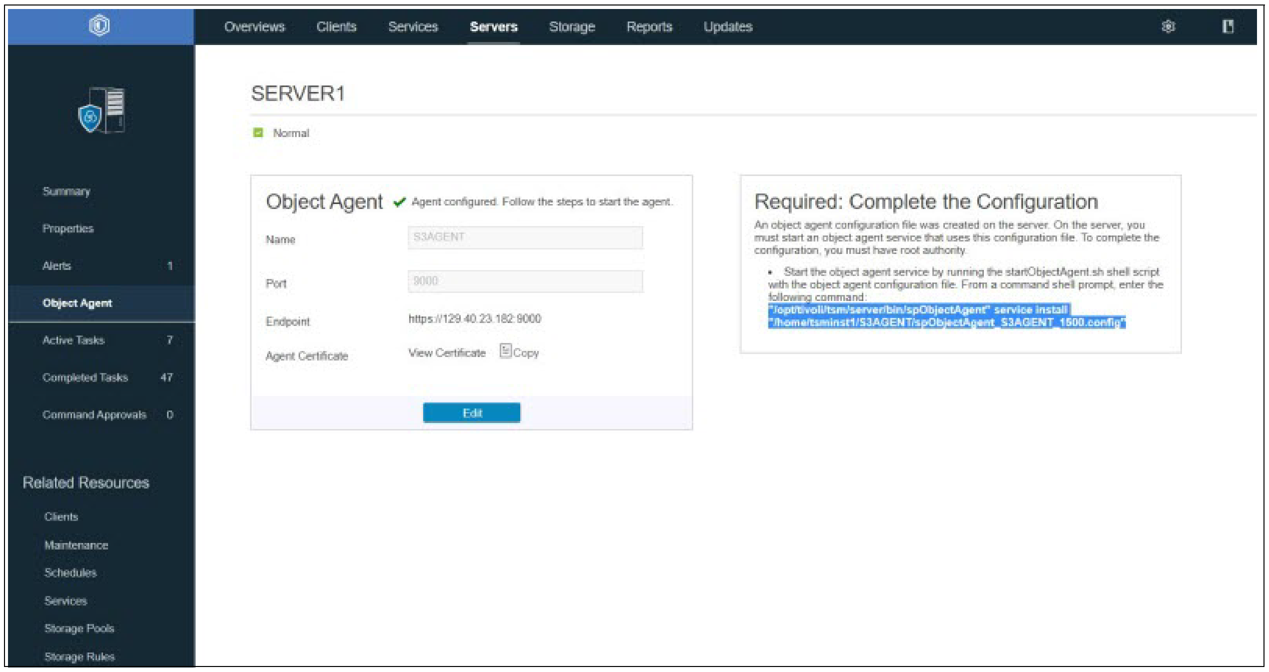
Port: 9000

Endpoint: https://129.40.23.182:9000

Agent Certificate

Figure 9-9 Details of an object agent

d. After creating the object agent, copy the command that is shown in the right pane (highlighted in Figure 9-10) to complete the configuration.



SERVER1

Normal

Object Agent ✓ Agent configured. Follow the steps to start the agent.

Name: S3AGENT

Port: 9000

Endpoint: https://129.40.23.182:9000

Agent Certificate

Required: Complete the Configuration

An object agent configuration file was created on the server. On the server, you must start an object agent service that uses this configuration file. To complete the configuration, you must have root authority.

- Start the object agent service by running the startObjectAgent.sh shell script with the object agent configuration file. From a command shell prompt, enter the following command:

```
"/opt/tivoli/tsm/server/bin/spObjectAgent" service install
"/home/tsminst1/S3AGENT/spObjectAgent_S3AGENT_1500.config"
```

Figure 9-10 Object agent configured

2. Start the object agent services to provide an endpoint that can send and receive data from the server. Connect to the IBM Spectrum Protect server and run the copied command on the IBM Spectrum Protect server. You must have superuser access to run this command. There can be only one object agent per IBM Spectrum Protect server.

Example 9-36 Running the command from Figure 9-10

```
# sudo "/opt/tivoli/tsm/server/bin/spObjectAgent" service install
"/home/tsminst1/S3AGENT/spObjectAgent_S3AGENT_1500.config"
2021-01-24 00:51:37.848863 I | Installed and started system service as
spoa9000S3AGENT.
```

3. Register an object client. The object client makes S3 requests to store and retrieve objects from the server.

   a. To register an object client, click **Clients** and then click **+ Client**. There can be multiple object clients per IBM Spectrum Protect server (Figure 9-11 on page 337).

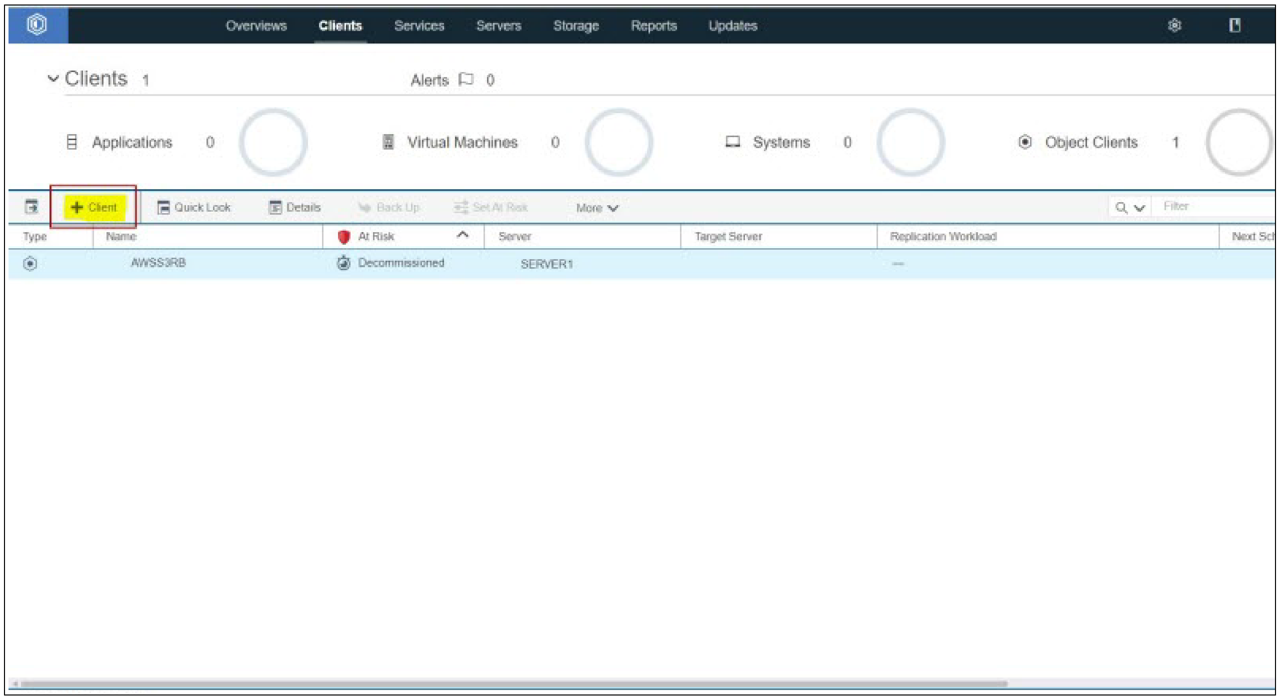
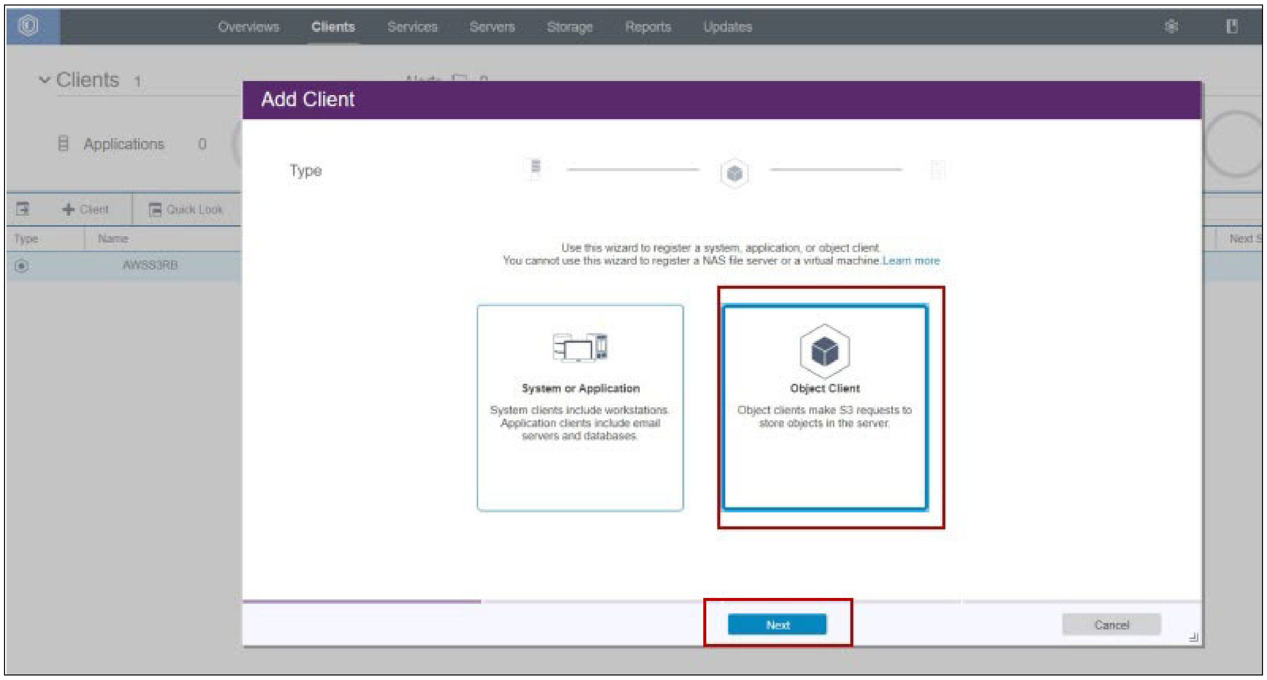


Figure 9-11 Adding a client

b. Select the object client and click **Next**, as shown in Figure 9-12.



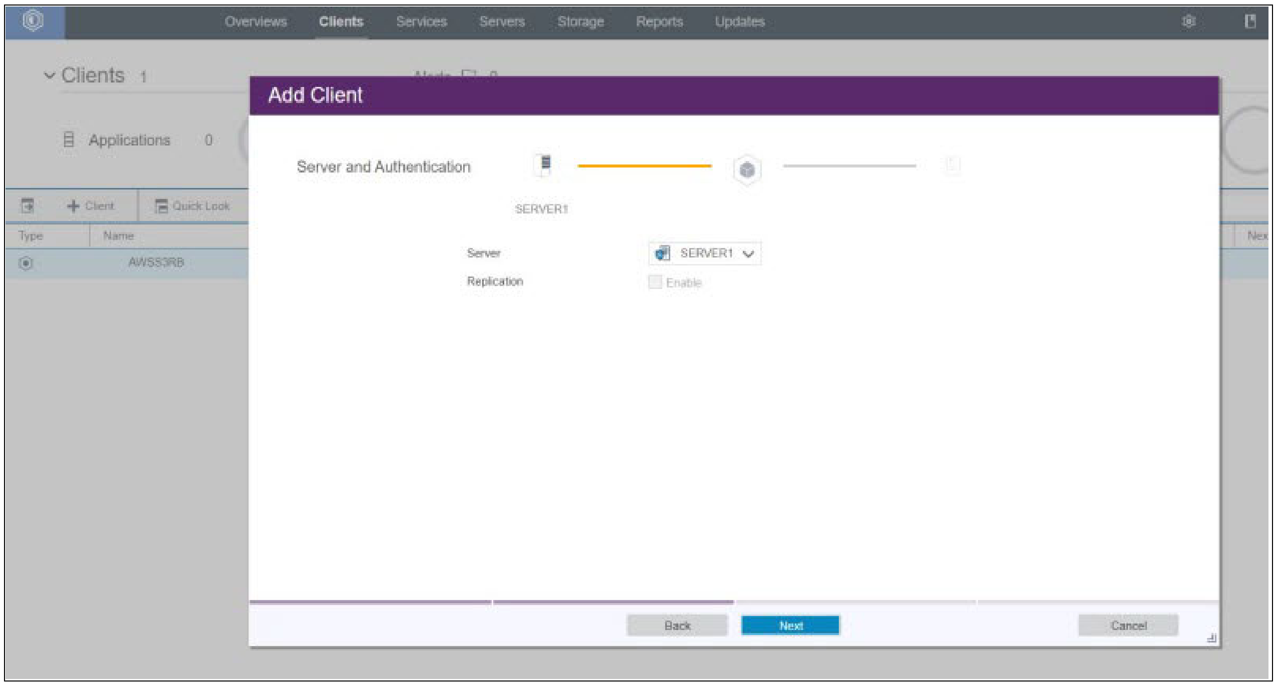
Use this wizard to register a system, application, or object client.
You cannot use this wizard to register a NAS file server or a virtual machine. Learn more

System clients include workstations. Application clients include email servers and databases.

Object clients make S3 requests to store objects in the server.

Figure 9-12 Available client options

c. Choose the IBM Spectrum Protect server and click **Next** (Figure 9-13).



Server and Authentication

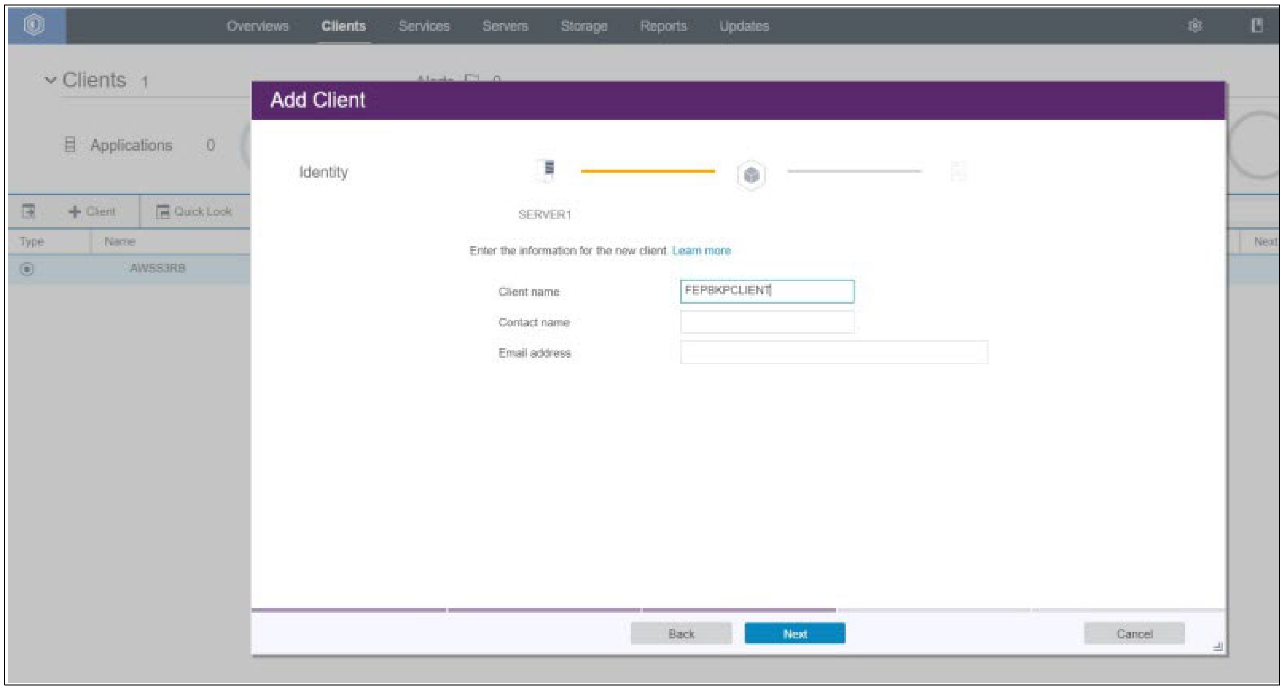
SERVER1

Server: SERVER1

Replication: Enable

Figure 9-13 Registering an object client to the server

d. Provide a client name. In our case, we used `FEPBKPClient`. Click **Next** (Figure 9-14).



Identity

SERVER1

Enter the information for the new client. Learn more

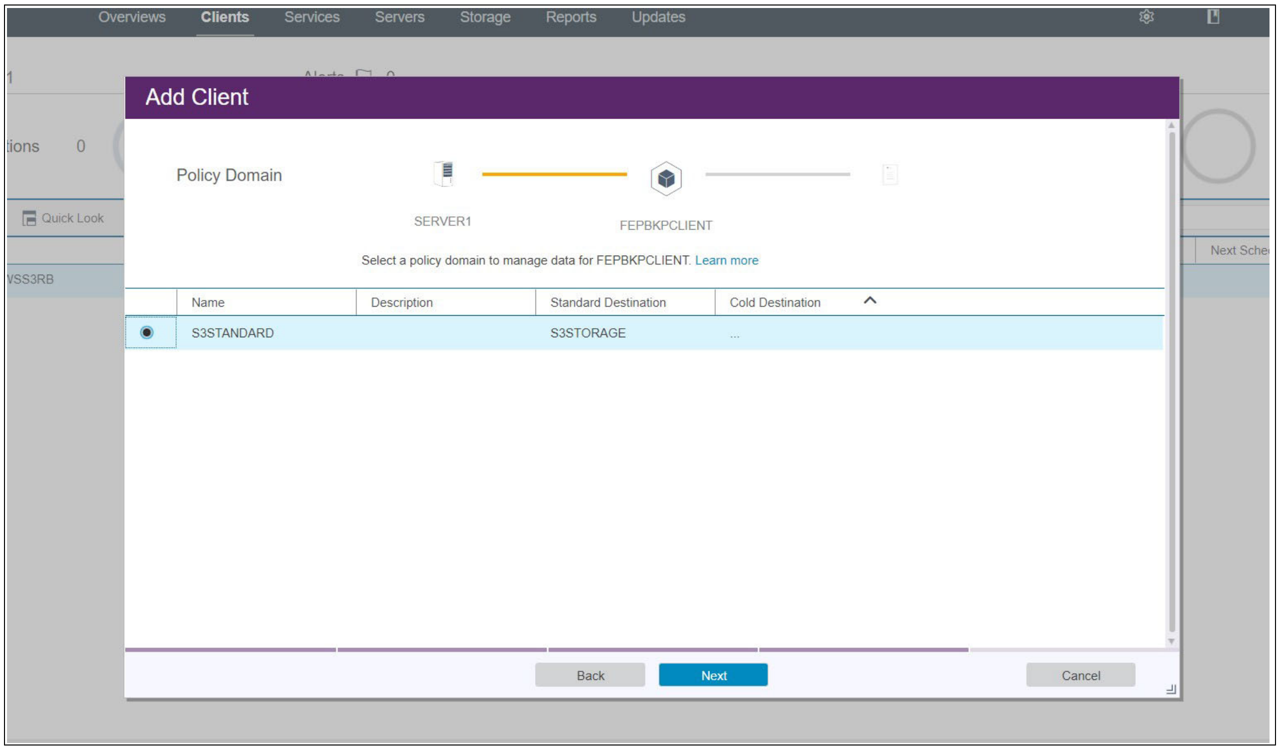
Client name: FEPBKPCLIENT

Contact name:

Email address:

Figure 9-14 Entering the object client parameters

e. We select an S3Standard domain. If there is nothing that is listed, you must create a domain and then click **Next** (Figure 9-15 on page 339).

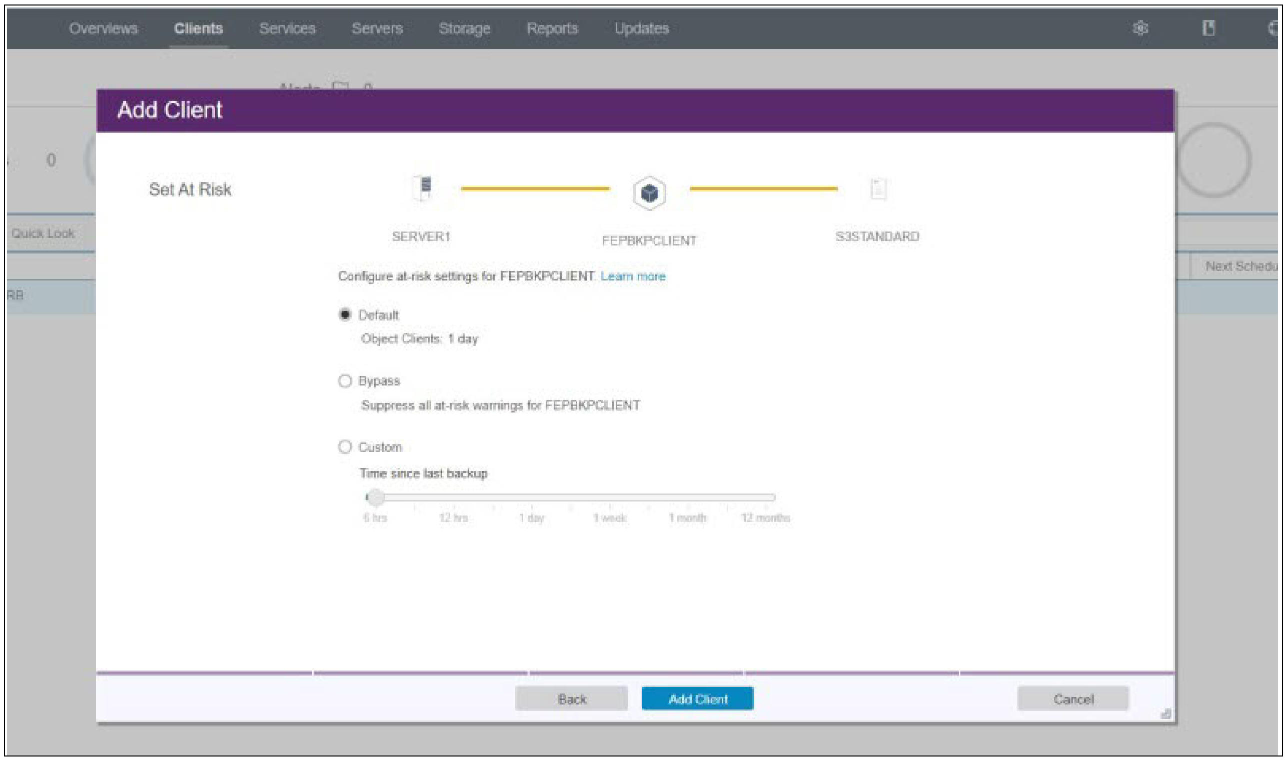


Select a policy domain to manage data for FEPBKPCLIENT. Learn more

	Name	Description	Standard Destination	Cold Destination
●	S3STANDARD		S3STORAGE	...

Figure 9-15 Configuring the object client parameters

f. Choose the default At Risk settings and then click **Add Client** (Figure 9-16).



Configure at-risk settings for FEPBKPCLIENT. Learn more

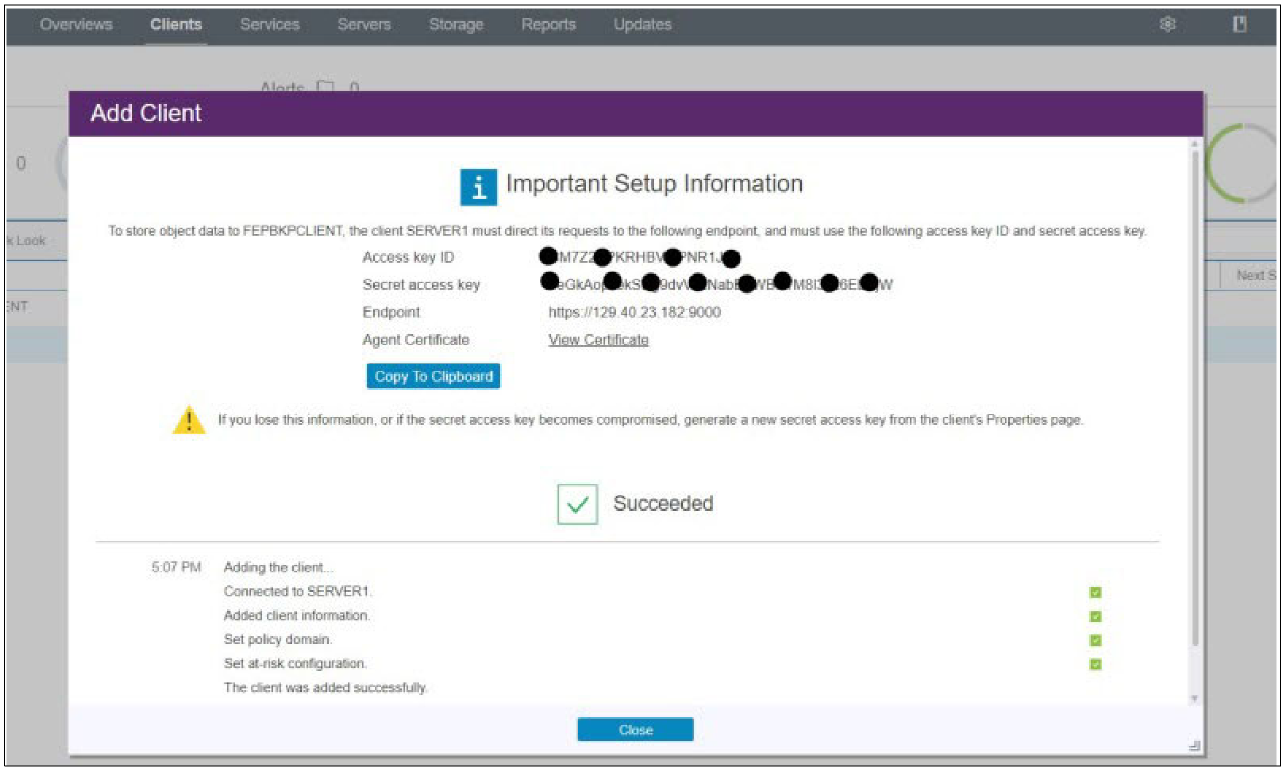
● Default
Object Clients: 1 day

○ Bypass
Suppress all at-risk warnings for FEPBKPCLIENT

○ Custom
Time since last backup

Figure 9-16 Selecting the At Risk settings for the object client

g. Save the ID, secret access key, and the agent certificate that are shown in Figure 9-17. This window is the only place where you can access the secret access key. If you lose it, you must regenerate the secret access key.



Add Client

Important Setup Information

To store object data to FEPBKPCLIENT, the client SERVER1 must direct its requests to the following endpoint, and must use the following access key ID and secret access key.

Access key ID	[redacted]M7Z2[redacted]KRHBV[redacted]PNR1J[redacted]
Secret access key	[redacted]aGkAo[redacted]kS[redacted]9dV[redacted]NabI[redacted]WE[redacted]M8I[redacted]6E[redacted]W
Endpoint	https://129.40.23.182:9000
Agent Certificate	View Certificate

If you lose this information, or if the secret access key becomes compromised, generate a new secret access key from the client's Properties page.

Succeeded

5:07 PM Adding the client...
Connected to SERVER1.
Added client information.
Set policy domain.
Set at-risk configuration.
The client was added successfully.

Figure 9-17 Object client connection information

4. Connect to the FUJITSU Enterprise Postgres server and configure the AWS client. In this publication, we use the AWS CLI as an S3 application, but you can use any other S3 compatible application to connect to IBM Spectrum Protect by using the secret key ID and secret access code. To get the list of S3 API calls that are supported by IBM Spectrum Protect at the time of writing, see Does IBM Spectrum Protect Plus support S3 compatible Object Storage?

Note: If the AWS CLI is not installed on the server, you can install it by using the following command:

```
yum install awscli
```

Example 9-37 Configuring the AWS client

```
# aws configure
AWS Access Key ID [None]: 8HAJZP6899999ZSG5G3K
AWS Secret Access Key [None]: Sje1D9999gnh9CN6hoFr4H9999HX8sQXERXOXtZ9
Default region name [None]:
Default output format [None]:
```

5. Copy the database backup that is created by using **pg_rman** to the IBM Spectrum Protect server. We are using AWS CLI to create a bucket and then copying the backup files on the IBM Spectrum Protect server. You can also integrate the backup by using **pg_rman** code and sending the backup files after the backup completes.

The directory tree for the backups is defined by the backup start time:

`<pg_rman backup path>/<start date>/<start time>`

Example 9-38 shows an example of our backups that used **`pg_rman`**.

Example 9-38 Command to back up

```
$ pg_rman show -B /rman_backup/inst1/
=====================================================================
 StartTime           EndTime              Mode    Size   TLI  Status
=====================================================================
2021-01-20 18:54:54  2021-01-20 18:55:12  INCR    67MB     2  OK
2021-01-14 08:58:10  2021-01-14 08:58:13  FULL   257MB     2  OK
```

The directory tree for the backups is shown in Table 9-2.

Table 9-2 Backup directory trees

Backup	Start time	Directory tree
INCR	2021-01-20 18:54:54	`/rman_backup/20210120/185454`
FULL	2021-01-14 08:58:10	`/rman_backup.20210114/085810`

Use the command that is shown in Example 9-39 to verify the directories.

Example 9-39 Verifying the directories

```
$ ls -ltr 202101*/
20210114/:
total 0
drwx------. 5 fsepuser fsepuser 133 Jan 14 08:58 085810
20210120/:
total 0
drwx------. 5 fsepuser fsepuser 133 Jan 20 18:55 185454
[fsepuser@rdbkpgr1 inst1]$ pwd
/rman_backup/inst1
```

6. Create an S3 bucket on the IBM Spectrum Protect server. In IBM Spectrum Protect, an S3 bucket is represented by a file space. To do this task, query the file space before creating a bucket, as shown in Example 9-40.

Example 9-40 Querying the file space

```
# Query file space before creating a S3 bucket.

Protect: SERVER1>q fi FEPBKPCLIENT *
ANR2034E QUERY FILESPACE: No match found using this criteria.
ANS8001I Return code 11.
```

a. After it is established that the S3 bucket does not exist, you can create the bucket by issuing the command that is shown in Example 9-41.

Example 9-41 Creating the bucket

```
# Create bucket by using the AWS client.
# aws s3 mb s3://fepbackup  --endpoint-url=https://xxx.xx.xx.182:9000
--no-verify-ssl
/usr/lib/python3.6/site-packages/urllib3/connectionpool.py:847:
InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate
verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning)
make_bucket: fepbackup
```

b. Verify that the bucket was created by querying the file space, as shown in Example 9-42.

Example 9-42 Verifying bucket creation

```
# Query file space after creating a S3 bucket.

Protect: SERVER1>q fi FEPBKPCLIENT *
Node Name           File space       FSID     Platform     Filespac-     Is
Filesp-        Capacity      Pct
               Name                         e Type      ace Unico-
Util
                                                                          de?
---------------    -----------    ----    --------    ---------    ----------
-----------     -----
FEPBKPCLIENT      fepbackup        1    Linux/s-    BUCKET          No
0 bytes       0.0
```

7. Take a full database backup by using **pg_rman** and copy the database files on to the IBM Spectrum Protect server, as shown in Example 9-43.

Example 9-43 Taking a full backup

```
$  pg_rman backup --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
--backup-mode=full -d postgres -p 27500
INFO: copying database files
INFO: copying archived WAL files
INFO: backup complete
INFO: Run 'pg_rman validate' to verify that the files are correctly copied.
```

a. You are instructed to run the command **pg_rman validate** to verify that the files were correctly copied. This command is shown in Example 9-44.

Example 9-44 Verifying that the files are copied

```
$  pg_rman validate --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
INFO: validate: "2021-01-24 02:18:42" backup and archive log files by CRC
INFO: backup "2021-01-24 02:18:42" is valid
$ pg_rman show --backup-path=/rman_backup/inst1
=====================================================================
 StartTime           EndTime              Mode    Size   TLI  Status
=====================================================================
2021-01-24 02:18:42  2021-01-24 02:19:04  FULL   223MB     2  OK
```

```
2021-01-20 18:54:54  2021-01-20 18:55:12  INCR    67MB     2  OK
2021-01-14 08:58:10  2021-01-14 08:58:13  FULL   257MB     2  OK
```

A new FULL backup is available with a start time of 2021-01-24 02:18:42. Based on the start time, we know that the backup directory is `/rman_backup/inst1/20210124/021842` (**`<pg_rman backup path>/<backup start date>/<backup start time>`**).

b. Use the AWS client to copy the database backup directory to the IBM Spectrum Protect server, as shown in Example 9-45.

Example 9-45 Copying the database backup directory

```
$ aws s3 cp /rman_backup/inst1/20210124/021842 s3://fepbackup/20210124/021842
--endpoint-url=https://xxx.xx.xx.182:9000 –recursive
upload: 20210124/021842/backup.ini to s3://fepbackup/20210124/021842/backup.ini
upload: 20210124/021842/arclog/000000020000000000000019.00000060.backup to
s3://fepbackup/20210124/021842/arclog/000000020000000000000019.00000060.backup
upload: 20210124/021842/database/PG_VERSION to
s3://fepbackup/20210124/021842/database/PG_VERSION
upload: 20210124/021842/database/backup_label to
s3://fepbackup/20210124/021842/database/backup_label
upload: 20210124/021842/database/backup_label.old to
s3://fepbackup/20210124/021842/database/backup_label.old
upload: 20210124/021842/database/base/1/112 to
s3://fepbackup/20210124/021842/database/base/1/112
upload: 20210124/021842/database/base/1/113 to
s3://fepbackup/20210124/021842/database/base/1/113
upload: 20210124/021842/database/base/1/1247 to
s3://fepbackup/20210124/021842/database/base/1/1247
upload: 20210124/021842/database/base/1/1247_fsm to
s3://fepbackup/20210124/021842/database/base/1/1247_fsm
upload: 20210124/021842/database/base/1/1247_vm to
s3://fepbackup/20210124/021842/database/base/1/1247_vm
upload: 20210124/021842/database/base/1/1249 to
s3://fepbackup/20210124/021842/database/base/1/1249
upload: 20210124/021842/database/base/1/1249_fsm to
s3://fepbackup/20210124/021842/database/base/1/1249_fsm
........................
upload: 20210124/021842/database/global/6002 to
s3://fepbackup/20210124/021842/database/global/6002
upload: 20210124/021842/database/pg_multixact/members/0000 to
s3://fepbackup/20210124/021842/database/pg_multixact/members/0000
upload: 20210124/021842/database/global/6115 to
s3://fepbackup/20210124/021842/database/global/6115
upload: 20210124/021842/database/global/4186 to
s3://fepbackup/20210124/021842/database/global/4186
```

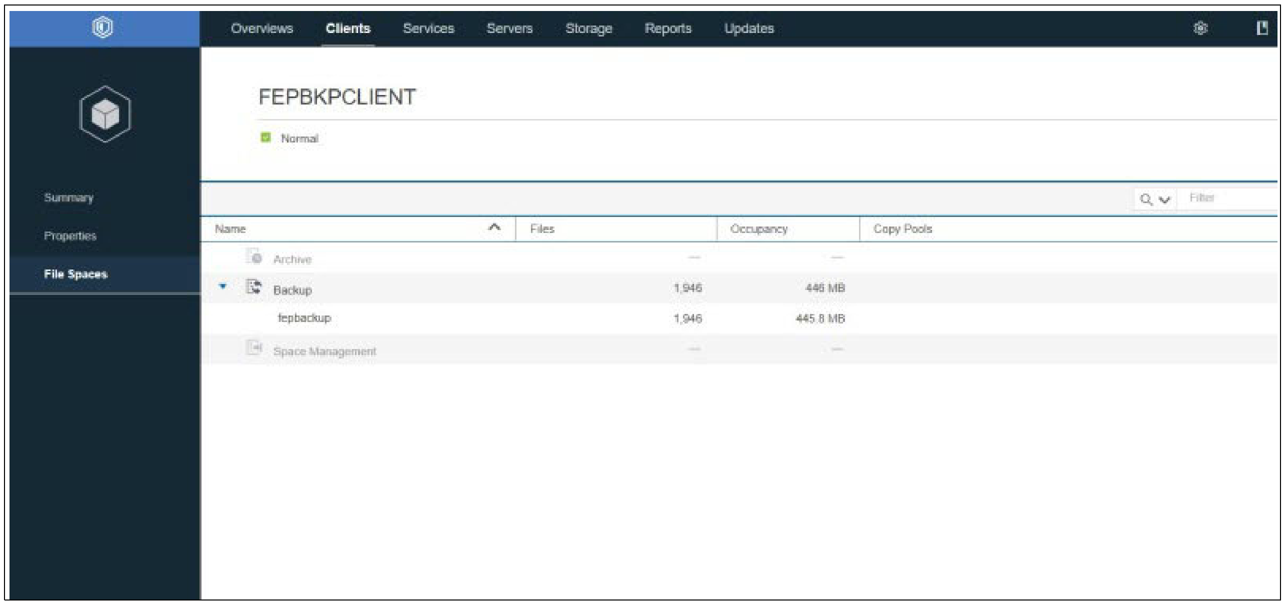
Note: We are using AWS CLI to copy the backup files from FUJITSU Enterprise Postgres server to IBM Spectrum Protect Server. You can integrate these commands with **`pg_rman validate`** as part of validating that a script moves the backup files to the IBM Spectrum Protect server.

8. Validate that all the files are copied and present on the IBM Spectrum Protect server. The command to use, and the results, are shown in Example 9-46.

Example 9-46 Validating the files

```
$ aws s3 ls s3://fepbackup --endpoint-url=https://xxx.xx.xx.182:9000 --recursive
2021-01-23 21:59:51        431 20210124/021842/backup.ini
2021-01-23 21:59:51   16777216 20210124/021842/arclog/000000020000000000000017
2021-01-23 21:59:51   16777216 20210124/021842/arclog/000000020000000000000019
2021-01-23 21:59:51   16777216 20210124/021842/arclog/000000020000000000000018
2021-01-23 21:59:51        347
20210124/021842/arclog/000000020000000000000019.00000060.backup
2021-01-23 21:59:51          3 20210124/021842/database/PG_VERSION
2021-01-23 21:59:51        233 20210124/021842/database/backup_label
2021-01-23 21:59:51        233 20210124/021842/database/backup_label.old
2021-01-23 21:59:51         92 20210124/021842/database/base/1/112
2021-01-23 21:59:51         92 20210124/021842/database/base/1/113
2021-01-23 21:59:51      75524 20210124/021842/database/base/1/1247
2021-01-23 21:59:51      24576 20210124/021842/database/base/1/1247_fsm
2021-01-23 21:59:51         36 20210124/021842/database/base/1/1247_vm
2021-01-23 21:59:51     440044 20210124/021842/database/base/1/1249
2021-01-23 21:59:51      24576 20210124/021842/database/base/1/1249_fsm
2021-01-23 21:59:52         36 20210124/021842/database/base/1/1249_vm
2021-01-23 21:59:52      24576 20210124/021842/database/base/1/1255_fsm
2021-01-23 21:59:52     641620 20210124/021842/database/base/1/1255
2021-01-23 21:59:52      83136 20210124/021842/database/base/1/1259
2021-01-23 21:59:52         36 20210124/021842/database/base/1/1255_vm
2021-01-23 21:59:52      24576 20210124/021842/database/base/1/1259_fsm
......
```

Figure 9-18 show file space details and the size of the backup data that are available on the selected file space.



FEPBKPCLIENT

Normal

Name	Files	Occupancy	Copy Pools
Archive	—	—	
Backup	1,946	446 MB	
fepbackup	1,946	445.8 MB	
Space Management	—	—	

Figure 9-18 File spaces and backup data that are available for FEPBKPCLIENT

9. Delete the last FULL backup from the FUJITSU Enterprise Postgres server and then retrieve it from IBM Spectrum Protect by using the AWS client, as shown in Example 9-47.

Example 9-47 Process to retrieve the full backup from IBM Spectrum Protect

```
$ pwd
/rman_backup/inst1/20210124
$ ls
021842
$ rm -rf 021842/
$ pg_rman show -B /rman_backup/inst1/
=====================================================================
 StartTime           EndTime              Mode    Size   TLI  Status
=====================================================================
2021-01-20 18:54:54  2021-01-20 18:55:12  INCR    67MB     2  OK
2021-01-14 08:58:10  2021-01-14 08:58:13  FULL   257MB     2  OK
```

10. Copy the backup folder from IBM Spectrum Protect, as shown in Example 9-48.

Example 9-48 Copying the backup folder

```
$ aws s3 ls s3://fepbackup/20210124/ --endpoint-url=https://xxx.xx.xx.182:9000
                           PRE 021842/
$ aws s3 cp s3://fepbackup/20210124/021842 /rman_backup/inst1/20210124/021842
--endpoint-url=https://xxx.xx.xx.182:9000 –recursive
download: s3://fepbackup/20210124/021842/database/backup_label.old to
rman_backup/inst1/20210124/021842/database/backup_label.old
download: s3://fepbackup/20210124/021842/backup.ini to
rman_backup/inst1/20210124/021842/backup.ini
download: s3://fepbackup/20210124/021842/database/backup_label to
rman_backup/inst1/20210124/021842/database/backup_label
download: s3://fepbackup/20210124/021842/database/base/1/112 to
rman_backup/inst1/20210124/021842/database/base/1/112
download:
s3://fepbackup/20210124/021842/arclog/000000020000000000000019.00000060.backup to
rman_backup/inst1/20210124/021842/arclog/000000020000000000000019.00000060.backup
download: s3://fepbackup/20210124/021842/database/PG_VERSION to
rman_backup/inst1/20210124/021842/database/PG_VERSION
download: s3://fepbackup/20210124/021842/database/base/1/113 to
rman_backup/inst1/20210124/021842/database/base/1/113
download: s3://fepbackup/20210124/021842/database/base/1/1247_vm to
rman_backup/inst1/20210124/021842/database/base/1/1247_vm
download: s3://fepbackup/20210124/021842/database/base/1/1247 to
rman_backup/inst1/20210124/021842/database/base/1/1247
download: s3://fepbackup/20210124/021842/database/base/1/1247_fsm to
rman_backup/inst1/20210124/021842/database/base/1/1247_fsm
download: s3://fepbackup/20210124/021842/database/base/1/1249_vm to
rman_backup/inst1/20210124/021842/database/base/1/1249_vm
download: s3://fepbackup/20210124/021842/database/base/1/1255_fsm to
rman_backup/inst1/20210124/021842/database/base/1/1255_fsm
download: s3://fepbackup/20210124/021842/database/base/1/1249_fsm to
rman_backup/inst1/20210124/021842/database/base/1/1249_fsm
download: s3://fepbackup/20210124/021842/database/base/1/1259 to
rman_backup/inst1/20210124/021842/database/base/1/1259
download: s3://fepbackup/20210124/021842/database/base/1/1259_vm to
rman_backup/inst1/20210124/021842/database/base/1/1259_vm
```

```
download: s3://fepbackup/20210124/021842/database/base/1/1255 to
rman_backup/inst1/20210124/021842/database/base/1/1255
.........
```

11. After the restore, list all the backups by using the **pg_rman show** command, as shown in Example 9-49.

Example 9-49 Listing all the backups

```
$ pg_rman show -B /rman_backup/inst1/
=====================================================================
 StartTime           EndTime              Mode    Size   TLI  Status
=====================================================================
2021-01-24 02:18:42  2021-01-24 02:19:04  FULL   223MB     2  OK
2021-01-20 18:54:54  2021-01-20 18:55:12  INCR    67MB     2  OK
2021-01-14 08:58:10  2021-01-14 08:58:13  FULL   257MB     2  OK
```

Note: There is a possibility that execute permissions are missing for the `mkdirs.sh` file. If so, you must give execute permissions to `mkdirs.sh` by using the following command:

```
chmod 777 mkdirs.sh
```

12. Delete the `pg_control` file from the database data directory to simulate a recovery, as shown in Example 9-50.

Example 9-50 Deleting the pg_control file

```
$ pg_ctl -D /database/inst1/ stop
waiting for server to shut down.... done
server stopped
$ cd /database/inst1/global/
$ rm -rf pg_control
$ cd /rman_backup/inst1/20210124/021842
$ ls -ltr
total 72
-rw-rw-r--. 1 fsepuser fsepuser   830 Jan 23 22:11 mkdirs.sh
-rw-rw-r--. 1 fsepuser fsepuser 56959 Jan 23 22:11 file_database.txt
-rw-rw-r--. 1 fsepuser fsepuser  2244 Jan 23 22:11 file_arclog.txt
drwxrwxr-x. 2 fsepuser fsepuser   150 Jan 24 05:03 arclog
drwxrwxr-x. 9 fsepuser fsepuser  4096 Jan 24 05:03 database
-rw-rw-r--. 1 fsepuser fsepuser   431 Jan 24 05:10 backup.ini
$ chmod 700 mkdirs.sh
$ ls -ltr
total 72
-rwx------. 1 fsepuser fsepuser   830 Jan 23 22:11 mkdirs.sh
-rw-rw-r--. 1 fsepuser fsepuser 56959 Jan 23 22:11 file_database.txt
-rw-rw-r--. 1 fsepuser fsepuser  2244 Jan 23 22:11 file_arclog.txt
drwxrwxr-x. 2 fsepuser fsepuser   150 Jan 24 05:03 arclog
drwxrwxr-x. 9 fsepuser fsepuser  4096 Jan 24 05:03 database
-rw-rw-r--. 1 fsepuser fsepuser   431 Jan 24 05:10 backup.ini
$ pg_rman restore --backup-path=/rman_backup/inst1 --pgdata=/database/inst1
WARNING: pg_controldata file "/database/inst1/global/pg_control" does not exist
INFO: the recovery target timeline ID is not given
INFO: use timeline ID of latest full backup as recovery target: 2
INFO: calculating timeline branches to be used to recovery target point
INFO: searching latest full backup, which can be used as restore start point
```

```
INFO: found the full backup can be used as base in recovery: "2021-01-24 02:18:42"
INFO: copying online WAL files and server log files
INFO: clearing restore destination
INFO: validate: "2021-01-24 02:18:42" backup and archive log files by SIZE
INFO: backup "2021-01-24 02:18:42" is valid
INFO: restoring database files from the full mode backup "2021-01-24 02:18:42"
INFO: searching incremental backup to be restored
INFO: searching backup, which contained archived WAL files to be restored
INFO: backup "2021-01-24 02:18:42" is valid
INFO: restoring WAL files from backup "2021-01-24 02:18:42"
INFO: restoring online WAL files and server log files
INFO: add recovery-related options to postgresql.conf
INFO: generating recovery.signal
INFO: restore complete
HINT: Recovery will start automatically when the PostgreSQL server is started.
$ pg_ctl -D /database/inst1/ start
waiting for server to start....2021-01-24 05:16:41.629 EST [375591] LOG:  starting
PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (GCC) 8.3.1 20190507 (Red
Hat 8.3.1-4), 64-bit
2021-01-24 05:16:41.630 EST [375591] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2021-01-24 05:16:41.631 EST [375591] LOG:  listening on IPv6 address "::", port
27500
2021-01-24 05:16:41.632 EST [375591] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2021-01-24 05:16:41.643 EST [375591] LOG:  redirecting log output to logging
collector process
2021-01-24 05:16:41.643 EST [375591] HINT:  Future log output will appear in
directory "log".
 done
server started
$ psql -d postgres -p 27500
psql (12.1)
Type "help" for help.
postgres=# show data_directory;
 data_directory
-----------------
 /database/inst1
(One row)
```

In this example, we restored the lost backup from IBM Spectrum Protect and performed the recovery of FUJITSU Enterprise Postgres Database by using the restored backups.

9.2 Optimizer hints

In this section, we provide an overview of FUJITSU Enterprise Postgres optimizer hints and describe the steps to configure and use the **pg_hint_plan** module to manage statistical information to influence a query execution plan. The **pg_hint_plan** module reads “hinting” phrase or phrases within a comment in a special form that is given with a target SQL statement. The special form starts with the character sequence “/*+” and ends with a “*/”.

9.2.1 Overview

FUJITSU Enterprise Postgres integrated the open source module **`pg_hint_plan`** to provide optimizer hints for query plans. Optimizer hints estimate the cost of each possible execution plan that is available for any SQL statement and pick the low-cost execution plan because it is considered the best plan. Optimizer hints help to assist the query planner in selecting one plan over another because the query planner sometimes does not know some of the details of a plan, such as associations between columns.

Optimizer hints enable FUJITSU Enterprise Postgres to stabilize the query plan in mission-critical systems. You can use **`pg_hint_plan`** to specify a query plan as a hint in each individual SQL statement.

Functional overview

The main query plans that can be specified by using this feature are:

- Query methods
- Join methods
- Join sequences

Optimizer hints can be specified by the methods that are shown in Table 9-3.

Table 9-3 Methods that are used with a query plan to query a table

Query plan	**Method**
Query methods	SeqScan, BitMapScan, IndexScan, IndexOnlyScan, TidScan, NoSeqScan, NoTidScan, NoIndexScan, NoIndexOnlyScan, and NoBitMapScan
Join methods	NestLoop, MergeJoin, HashJoin, NoNestLoop, NoMergeJoin, and NoHashJoin
Join sequences	Leading (table names)

For more information about the FUJITSU Enterprise Postgres optimizer hints feature, see 9.1.1, “Optimizer Hints”, in the *FUJITSU Enterprise Postgres on IBM LinuxONE Application Development Guide*.

9.2.2 How to use optimizer hints

In this section, we describe how to set up optimizer hints in your environment and demonstrate the use of optimizer hints by completing the following steps:

1. Copy the set of files that are related to **`pg_hint_plan`** in to the installation directory. These files are in the installation directory of FUJITSU Enterprise Postgres under the `OSS/pg_hint_plan` subdirectory. Run the following command to do this task:

   ```
   [fsepuser@rdbkpgr4 ~]$ cp -r /opt/fsepv12server64/OSS/pg_hint_plan/*
   /opt/fsepv12server64
   ```

2. The FUJITSU Enterprise Postgres Database server should be configured with the following parameter in the configuration file `postgresql.conf`:

   ```
   Shared_preload_libraries= ‘pg_hint_plan’
   ```

3. Create an extension inside the database session, as shown in Example 9-51 on page 349.

Example 9-51 Command to create a pg_hint_plan module

```
postgres=# create extension pg_hint_plan;
CREATE EXTENSION
```

The following command sets **pg_hint_plan.debug_print** to on in a session and prints details about which hints are used, which are not used, which hints are duplicated, and which are error hints:

```
postgres=# set pg_hint_plan.debug_print =on;
```

In the following examples, we show how to make the query planner pick up an index scan for the **pgbench_accounts** table. In the absence of a hint, a sequence scan is picked up by the query planner, and with hints turned on, a query planner index scan is picked up.

Example 9-52 demonstrates a query without using the command hint during a query of a table.

Example 9-52 Querying without a hint

```
postgres=#  explain select * from pgbench_branches b join pgbench_accounts a on
b.bid=a.bid;
                                    QUERY PLAN
----------------------------------------------------------------------------------
 Hash Join  (cost=1.45..59150.94 rows=1999999 width=461)
   Hash Cond: (a.bid = b.bid)
  ->  Seq Scan on pgbench_accounts a  (cost=0.00..52786.99 rows=1999999 width=97)
  ->  Hash  (cost=1.20..1.20 rows=20 width=364)
        ->  Seq Scan on pgbench_branches b  (cost=0.00..1.20 rows=20 width=364)
(Five rows)
```

By using the same query, Example 9-53 tells the query planner to pick up an index scan for the **pgbench_accounts** table.

Example 9-53 Picking up an index scan by using the IndexScan hint

```
postgres=# /*+ IndexScan(a) */ explain select * from pgbench_branches b join
pgbench_accounts a on b.bid=a.bid;
                                             QUERY PLAN
----------------------------------------------------------------------------------
------------------
 Hash Join  (cost=1.88..91130.36 rows=1999999 width=461)
   Hash Cond: (a.bid = b.bid)
   ->  Index Scan using bidindex on pgbench_accounts a  (cost=0.43..84766.41
rows=1999999 width=97)
   ->  Hash  (cost=1.20..1.20 rows=20 width=364)
         ->  Seq Scan on pgbench_branches b  (cost=0.00..1.20 rows=20 width=364)
(Five rows)
```

Example 9-54 provides an example of the command that requests a hint for two tables: **emp ta** and **emp tb**.

Example 9-54 Hint for each table

```
postgres=# WITH /*+ SeqScan(ta) IndexScan(tb) */ over100 AS (SELECT empno FROM emp
ta WHERE salary > 1000000)SELECT * FROM emp tb, over100 WHERE tb.empno =
over100.empno AND tb.age < 30;
```

Example 9-55 provides an example of the command to provide a hash join method. Hash join is selected as the joining method and scans the table **`pgbench_accounts`**.

Example 9-55 Form to use the hash join method

```
postgres=# /*+ HashJoin(a b)*/ explain select * from pgbench_branches b join
pgbench_accounts a on b.bid=a.bid;
                                    QUERY PLAN
--------------------------------------------------------------------------------
-
 Hash Join  (cost=1.45..59150.94 rows=1999999 width=461)
   Hash Cond: (a.bid = b.bid)
   ->  Seq Scan on pgbench_accounts a  (cost=0.00..52786.99 rows=1999999 width=97)
   ->  Hash  (cost=1.20..1.20 rows=20 width=364)
         ->  Seq Scan on pgbench_branches b  (cost=0.00..1.20 rows=20 width=364)
(Five rows)
```

Use case

By using an optimizer hint, the query planner changes the plan that is chosen. In Example 9-56, we use tables **`dept`** and **`emp`** to demonstrate how hinting helps to improve the query plan by picking up an index scan on the table.

Example 9-56 Commands: Indexes and number of rows in tables

```
postgres=# \d emp
                Table "public.emp"
 Column |  Type   | Collation | Nullable | Default
-------+---------+-----------+----------+---------
 empno  | integer |           | not null |
 name   | text    |           |          |
 age    | integer |           |          |
 deptno | integer |           |          |
 salary | integer |           |          |
Indexes:
    "emp_pkey" PRIMARY KEY, btree (empno)
    "emp_age_index" btree (age)

postgres=# \d dept
                Table "public.dept"
 Column |  Type   | Collation | Nullable | Default
-------+---------+-----------+----------+---------
 deptno | integer |           |          |
 name   | text    |           |          |
Indexes:
    "dept_deptno_index" UNIQUE, btree (deptno)

postgres=# select count(*) from emp;
 count
------
  2000
(One row)

postgres=# select count(*) from dept;
 count
------
    30
```

```
(One row)
postgres=# analyze;
ANALYZE
```

In Example 9-57, we show a query execution without hints.

Example 9-57 Query without hints

```
postgres=# EXPLAIN ANALYZE WITH age30 as (SELECT * FROM emp WHERE age BETWEEN 30
AND 39)SELECT * FROM age30, dept WHERE age30.deptno = dept.deptno ;
                                                  QUERY PLAN
---------------------------------------------------------------------------------
--------------------------
 Hash Join  (cost=1.68..46.64 rows=655 width=32) (actual time=0.046..1.062
rows=656 loops=1)
   Hash Cond: (emp.deptno = dept.deptno)
  ->  Seq Scan on emp  (cost=0.00..43.00 rows=655 width=22) (actual
time=0.006..0.426 rows=656 loops=1)
         Filter: ((age >= 30) AND (age <= 39))
         Rows Removed by Filter: 1344
  ->  Hash  (cost=1.30..1.30 rows=30 width=10) (actual time=0.031..0.032 rows=30
loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 10kB
        ->  Seq Scan on dept  (cost=0.00..1.30 rows=30 width=10) (actual
time=0.002..0.014 rows=30 loops=1)
 Planning Time: 0.654 ms
 Execution Time: 1.348 ms
(Ten rows)
```

Example 9-58 shows a query execution with hints.

Example 9-58 Query with hints

```
postgres=# EXPLAIN ANALYZE WITH age30 as (SELECT * FROM emp WHERE age = 33)SELECT
* FROM age30, dept WHERE age30.deptno = dept.deptno ;
                                              QUERY PLAN
---------------------------------------------------------------------------------
 Hash Join  (cost=1.95..14.27 rows=5 width=32) (actual time=0.058..0.071 rows=5
loops=1)
   Hash Cond: (emp.deptno = dept.deptno)
  ->  Index Scan using emp_age_index on emp  (cost=0.28..12.58 rows=5
width=22) (actual time=0.011..0.019 rows=5 loops=1)
         Index Cond: (age = 33)
  ->  Hash  (cost=1.30..1.30 rows=30 width=10) (actual time=0.036..0.037 rows=30
loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 10kB
        ->  Seq Scan on dept  (cost=0.00..1.30 rows=30 width=10) (actual
time=0.005..0.018 rows=30 loops=1)
 Planning Time: 0.238 ms
 Execution Time: 0.137 ms
(Nine rows)

postgres=# EXPLAIN ANALYZE WITH /*+ IndexScan(emp emp_age_index) */ age30 as
(SELECT * FROM emp WHERE age BETWEEN 30 AND 39) SELECT * FROM age30, dept WHERE
age30.deptno = dept.deptno ;
                                              QUERY PLAN
```

```
-------------------------------------------------------------------------------
 Hash Join  (cost=1.95..52.61 rows=655 width=32) (actual time=0.058..0.992
rows=656 loops=1)
   Hash Cond: (emp.deptno = dept.deptno)
  ->  Index Scan using emp_age_index on emp  (cost=0.28..48.98 rows=655
width=22) (actual time=0.014..0.343 rows=656 loops=1)
         Index Cond: ((age >= 30) AND (age <= 39))
  ->  Hash  (cost=1.30..1.30 rows=30 width=10) (actual time=0.035..0.036 rows=30
loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 10kB
        ->  Seq Scan on dept  (cost=0.00..1.30 rows=30 width=10) (actual
time=0.004..0.017 rows=30 loops=1)
 Planning Time: 0.307 ms
 Execution Time: 1.305 ms
(Nine
rows)
```

9.3 Locked statistics

In this section, we provide an overview of and describe the steps to configure and use the **pg_dbms_stats** module to manage statistical information to influence a query execution plan.

9.3.1 Overview

Locked statistics is a FUJITSU Enterprise Postgres feature that locks the various relational statistics in custom views. During query plan generation, the optimizer component of the server calculates the costs based on these locked statistics. With this action, the query plan generation is stabilized because these locked statistics are used every time to generate the plan. This situation reduces the risk of a sudden execution plan change in the customer environment. This function is achieved by installing and using the module **pg_dbms_stats**. Locked statistics also helps to reproduce the same query plan in both a testing environment and a production environment.

Functional overview

PostgreSQL aggregates values that are sampled from the table or index by the **ANALYZE** command and maintains statistical information. The optimizer calculates the cost of a query by using this statistical information and chooses the lowest cost execution plans.

There are three methods to lock the relation statistics and use them during plan generation:

- Backup and restore statistics (Table 9-4 on page 353)
- Lock and unlock statistics (Table 9-5 on page 353)
- Export and import statistics (Table 9-6 on page 354)

Table 9-4 Backup/Restore statistics

Operation	Summary	How to use	Detail
Backup	To reproduce the current execution plan in the future, and back up the current planner statistics.	Run: `backup_<object unit>_stats() SQL function`	Statistical information to be included in the backup can be specified for the database (currently connected), schema, table, or particular column of the table. For example, if you want to save statistics for all tables and their columns in the schema, specify the schema name in the function. For a large backup, specify the database or schema.
Restore	Restore earlier backup statistics plan.	Run one of the following commands: ► **`restore_stats()`** ► **`restore_<object unit>_stats() SQL function`**	Restore the information of an object that is selected at backup without affecting other objects' statistics. Backed up statistical information can be restored by using two ways: 1. Backup ID Restore all statistical information that is contained in the backup by passing a backup ID to the **`restore_stats()`** function. It is simpler for the regular basis backup operations by specifying the database or schema. If you have **`pg_dbms_stats`** in more than one database, the backup ID is unique to each database. 2. Object + Timestamp Use the **`restore_<object unit>_stats()`** function to restore the statistical information of an object, either of the database, schema, table, or column, at the specified PiT. Used for backup statistics with an older timestamp, and it has a wider range than restoring.

Table 9-5 Lock/Unlock statistics

Operation	Summary	How to use	Details
Lock Statistics	Locks the statistics and prevents the changes in the statistical information due to **`ANALYZE`**.	Run: `lock_<object unit>_stats() SQL function`	Specify the database (currently connected), schema, table, or column to lock its statistical information.
Unlock Statistics	To restore the PostgreSQL execution plan criteria to their original state, unlock the lock that was acquired earlier.	Run: `unlock_<object unit>_stats() SQL function`	Release the lock on the **`pg_class`** and **`pg_statistic`** specified database (currently connected), schemas, table, or column. Also, you can unlock by specifying a different lock unit.

Table 9-6 Export/Import statistics

Operation	Summary	How to use	Details
Export Statistics	Export the current statistics in the external file.	Create a **`COPY`** statement that references the sample SQL file (`export__stats-.sql.sample`), and run the statement.	It uses the **`COPY`** command to export. You must specify the directory where the PostgreSQL user has permission to export the file. Depending on the application, you can export statistical information by using two methods: 1. PostgreSQL statistics The **`pg_dbms_stats`** module copies statistical information from the PostgreSQL storage in **`pg_class`** and **`pg_statistic`** tables. Do not install **`pg_dbms_stats`** because it is intended to copy statistics from one environment to another one. It is used only for performance tuning and analysis. 2. Currently valid statistics You can look at the statistics planner by using **`pg_dbms_stats`**. It can be edited and turned back into production statistical information, and it also can be included in OS file backup. A sample file is installed in the extension subdirectory. You can display this directory by running **`pg_config --docdir`**.
Import Statistics	Restore the statistical information from the external file that was created by the export.	Run the **`import_<object unit>_stats()`** SQL function.	Statistical information can be imported for databases (currently connected), schema, tables, or a column of a table. Place the file that you want to import into a directory that is readable by a PostgreSQL running user.

For more information about the FUJITSU Enterprise Postgres locked statistics feature, see 9.1.2, “Locked Statistics” in the *FUJITSU Enterprise Postgres on IBM LinuxONE Application Development Guide*.

Statistics source and destination

In this section, we answer two commonly asked questions about locked statistics:

- What statistics are locked?

  The **`pg_dbms_stats`** command locks the following statistics, and the optimizer uses this locked statistical information instead of the actual system catalog statistics during query plan generation:

  – Values in the row that were sampled by **`ANALYZE`**, that is, per columns stats of a **`relation(pg_catalog.pg_statistic)`**.

  – Estimated number of rows when **`ANALYZE(pg_catalog.pg_class.reltuples)`** runs.

  – File size at **`ANALYZE`** time, that is, the number of pages in **`relation(pg_catalog.pg_class.relpages)`**.

  – File size of the execution plan creation.

- Where are the statistics locked?

  The locked statistics are held in various custom views like **`dbms_stats.relation_stats_locked`** and **`dbms_stats.relation_stats_backup`** by using SQL statements, as shown Example 9-59 on page 355.

Example 9-59 Location of locked statistics

```
INSERT INTO dbms_stats.relation_stats_locked
        SELECT $1, dbms_stats.relname(nspname, c.relname),
               v.relpages, v.reltuples, v.relallvisible, v.curpages,
               v.last_analyze, v.last_autoanalyze
          FROM pg_catalog.pg_class c,
               pg_catalog.pg_namespace n,
               dbms_stats.relation_stats_effective v
         WHERE c.oid = $1
           AND c.relnamespace = n.oid
           AND v.relid = $1;
```

9.3.2 How to use locked statistics

In this section, we describe the steps that are used to set up and use locked statistics:

1. Copy the set of files that is related to **`pg_dbms_stats`** to the installation directory. These files are in the installation directory of FUJITSU Enterprise Postgres under the `OSS/pg_dbms_stats` subdirectory. Use the following command to copy the **`pg_dbms_stats`** files:

   ```
   [root@rdbkpgr4 fsepuser]# cp -r /opt/fsepv12server64/OSS/pg_dbms_stats/*
   /opt/fsepv12server64
   ```

2. The FUJITSU Enterprise Postgres Database server should be configured with the following parameter in the configuration file `postgresql.conf`:

   ```
   shared_preload_libraries = 'pg_dbms_stats'
   ```

3. Create an extension inside the database session, as shown in Example 9-60.

Example 9-60 Creating the pg_dbms_stats module

```
postgres=# create extension pg_dbms_stats
CREATE EXTENSION
```

Example use cases

In this section, we provide examples of how to use locked statistics.

Locking statistics

Example 9-61 shows the delete operation of 500 records on a locked table that does not change the number of rows of statistics in an **`explain`** plan. The number of rows (`rows=100000`) shows the count before the **`delete`** operation, which is the number that is retained at the time when the statistics were locked.

Example 9-61 Command to lock statistics

```
postgres=# explain analyze select * from pgbench_accounts;
                                                       QUERY PLAN
---------------------------------------------------------------------------------
 Seq Scan on pgbench_accounts  (cost=0.00..2640.00 rows=100000 width=97) (actual
time=0.012..49.327 rows=100000 loops=1)
 Planning Time: 0.075 ms
 Execution Time: 88.896 ms
(Three rows)
```

```
postgres=# select dbms_stats.lock_table_stats('public.pgbench_accounts');
 lock_table_stats
-----------------
 pgbench_accounts
(One row)

postgres=# delete from pgbench_accounts where aid<=500;
DELETE 500
postgres=# analyze;
ANALYZE

postgres=# explain analyze select * from pgbench_accounts;
                                                       QUERY PLAN
---------------------------------------------------------------------------------
 Seq Scan on pgbench_accounts  (cost=0.00..2640.00 rows=100000 width=97) (actual
time=0.036..42.408 rows=99500 loops=1)
 Planning Time: 0.595 ms
 Execution Time: 81.814 ms
(Three rows)
postgres=# select * from dbms_stats.relation_stats_locked;
 relid |        relname        | relpages | reltuples | relallvisible | curpages
|         last_analyze          |       last_autoanalyze
------+-------------------------+----------+-----------+---------------+----------
 16485 | public.pgbench_accounts |     1640 |    100000 |          1640 |     1640
| 2020-11-11 20:37:31.211904-05 | 2020-11-11 20:20:14.129794-05
(One row)
```

Unlocking statistics

Unlock the database object to start updating statistical information. In Example 9-62, the **analyze** command updates the table statistics after the unlock operation. So, the number of rows (rows=99500) is updated and reflected in the query plan.

Example 9-62 Commands to unlock statistics

```
postgres=# select dbms_stats.unlock_table_stats('public.pgbench_accounts');
 unlock_table_stats
------------------
 pgbench_accounts
(One row)
postgres=# analyze;
ANALYZE
postgres=# explain analyze select * from pgbench_accounts;
                                                      QUERY PLAN
---------------------------------------------------------------------------------
 Seq Scan on pgbench_accounts  (cost=0.00..2635.00 rows=99500 width=97) (actual
time=0.013..42.419 rows=99500 loops=1)
 Planning Time: 0.939 ms
 Execution Time: 81.993 ms
(Three rows)
```

Backing up statistics

Backing up statistics back up the current statistics. Example 9-63 on page 357 shows the command to back up database statistics.

Example 9-63 Command to back up database statistics

```
postgres=# select dbms_stats.backup_database_stats('Backup1');
 backup_database_stats
----------------------
                     1
(One row)
postgres=# select * from pg_catalog.pg_stat_database;
 datid |  datname  | numbackends | xact_commit | xact_rollback | blks_read |
blks_hit | tup_returned | tup_fetched | tup_inserted | tup_updated | tup_deleted |
conflicts | temp_files | temp
_bytes | deadlocks | checksum_failures | checksum_last_failure | blk_read_time |
blk_write_time |         stats_reset
------+-----------+-------------+-------------+---------------+-----------+-------
0 |          |          0 |          0 |            0 |      101 |    85320 |
31439 |       18018 |           4 |           0 |           0 |          0 |
0 |
    0 |       0 |                |                 |          0 |
0 | 2020-11-08 19:38:00.610541-05
 13468 | postgres  |           2 |       9433 |             7 |      7021 |
444516 |     6542572 |      91104 |      101558 |       2249 |       6123 |
0 |          4 |    2
031616 |       0 |                |                 |          0 |
0 | 2020-11-08 19:38:00.610528-05
     1 | template1 |           0 |           0 |             0 |         0 |
0 |           0 |           0 |           0 |           0 |           0 |
0 |          0 |
    0 |       0 |                |                 |          0 |
0 |
 13467 | template0 |           0 |           0 |             0 |         0 |
0 |           0 |           0 |           0 |           0 |           0 |
0 |          0 |
    0 |       0 |                |                 |          0 |
0 (four rows)
```

Restoring statistics

Statistics can be restored from a backup and then locked. Example 9-64 shows how to restore database statistics from a backup that was taken earlier. Restore statistics need a timestamp from when the backup was taken. Example 9-64 includes details about how to get the timestamp details of a backup.

Example 9-64 Example command to restore database statistics

```
postgres=# select * from pg_catalog.pg_stat_database;
 datid |  datname  | numbackends | xact_commit | xact_rollback | blks_read |
blks_hit | tup_returned | tup_fetched | tup_inserted | tup_updated | tup_deleted |
conflicts | temp_files | temp
_bytes | deadlocks | checksum_failures | checksum_last_failure | blk_read_time |
blk_write_time |         stats_reset
------+-----------+-------------+-------------+---------------+-----------+-------
0 |          |          0 |          0 |            0 |      101 |    85441 |
31483 |       18047 |           4 |           0 |           0 |          0 |
0 |
    0 |       0 |                |                 |          0 |
0 | 2020-11-08 19:38:00.610541-05
```

```
 13468 | postgres  |           2 |        9460 |            7 |      7033 |
446422 |     6550102 |      91795 |      101568 |       2280 |       6123 |
0 |          4 |    2
031616 |       0 |                 |                   |          0 |
0 | 2020-11-08 19:38:00.610528-05
     1 | template1 |           0 |           0 |            0 |         0 |
0 |           0 |           0 |            0 |           0 |           0 |
0 |          0 |
    0 |       0 |                 |                   |          0 |
0 |
 13467 | template0 |           0 |           0 |            0 |         0 |
0 |           0 |           0 |            0 |           0 |           0 |
0 |          0 |
    0 |       0 |                 |                   |          0 |
0 |
(Four rows)

postgres=# select time from dbms_stats.backup_history;
            time
-----------------------------
 2020-11-11 21:55:58.94369-05
(One row)

postgres=# select dbms_stats.restore_database_stats('2020-11-11
21:55:58.94369-05');
                    restore_database_stats
-----------------------------------------------------------------
 pgx_confidential_policies
 pgx_confidential_policies_schema_name_table_name_policy_nam_key
 pgx_confidential_columns
 pgx_confidential_columns_schema_name_table_name_column_name_key
 pgx_confidential_values
 fulldemo
 unsupported_demo
 fulldemo1
 demo
 account
 account_noaudit
 auditlog1
 auditlog2
 pgbench_history
 pgbench_tellers
 pgbench_accounts
 pgbench_branches
 pgbench_branches_pkey
 pgbench_tellers_pkey
 pgbench_accounts_pkey
(20 rows)
postgres=# select * from pg_catalog.pg_stat_database;
 datid |  datname  | numbackends | xact_commit | xact_rollback | blks_read |
blks_hit | tup_returned | tup_fetched | tup_inserted | tup_updated | tup_deleted |
conflicts | temp_files | temp
_bytes | deadlocks | checksum_failures | checksum_last_failure | blk_read_time |
blk_write_time |          stats_reset
```

```
------+-----------+------------+------------+--------------+----------+-------
0 |          |          0 |          0 |            0 |      101 |    85472 |
31497 |      18055 |            4 |            0 |            0 |          0 |
0 |
    0 |        0 |                 |                    |           0 |
0 | 2020-11-08 19:38:00.610541-05
 13468 | postgres  |            2 |        9465 |              7 |       7034 |
451886 |     6579786 |      94918 |      101645 |        2300 |       6200 |
0 |          4 |    2
031616 |        0 |                 |                    |           0 |
0 | 2020-11-08 19:38:00.610528-05
    1 | template1 |           0 |            0 |              0 |          0 |
0 |           0 |          0 |            0 |           0 |           0 |
0 |          0 |
    0 |        0 |                 |                    |           0 |
0 |
 13467 | template0 |           0 |            0 |              0 |          0 |
0 |           0 |          0 |            0 |           0 |           0 |
0 |          0 |
    0 |        0 |                 |                    |           0 |
0 |
(Four rows)
```

Exporting statistics

A sample SQL file with commands to export database statistics is provided in the FUJITSU Enterprise Postgres installation directory. Modify this file based on your own requirements.

When using the sample file, by default, statistics are exported to the `export_stats.dmp` file. You can modify the name of this file by modifying the sample file. Example 9-65 provides an example of the command to export database statistics.

Example 9-65 Command to export database statistics

```
[fsepuser@rdbkpgr4 ~]$ cp
/opt/fsepv12server64/share/doc/extension/export_effective_stats-12.sql.sample
export_effective_stats-12.sql
[fsepuser@rdbkpgr4 ~]$ psql -d postgres -f export_effective_stats-12.sql
BEGIN
COMMIT
[fsepuser@rdbkpgr4 ~]$ ls -l export_stats.dmp
rw-rw-r--. 1 fsepuser fsepuser 30890 Nov 11 22:17 export_stats.dmp
```

Importing statistics

Import statistics read the statistics from an external file that is created by the export feature, and then locks it.

Example 9-66 Command to import statistics from a dump taken earlier

```
postgres=# select
dbms_stats.import_database_stats('/home/fsepuser/export_stats.dmp');
import_database_stats
----------------------
(One row)
```

9.4 Monitoring

In this section, we provide an overview of monitoring and describe the steps to configure and use the monitoring tools **pgBadger** and **pg_statsinfo** on a FUJITSU Enterprise Postgres 12 server with Red Hat Enterprise Linux Server 8 on IBM LinuxONE.

9.4.1 Overview

You should periodically monitor database performance and make sure that it does not degrade over time. FUJITSU Enterprise Postgres integrated the open source modules **pgBadger** and **pg_statsinfo** to check and improve database performance.

pgBadger parses FUJITSU Enterprise Postgres log files and produces statistical reports. It is an easy tool to use to analyze SQL traffic and create HTML5 reports. Using **pgBadger** graphs, you can understand database workloads and queries that affect the performance of a database.

pg_statsinfo periodically gathers information about a database server and stores it as snapshots into a repository database.

Where **pgBadger** reports everything about SQL queries, **pg_statsinfo** captures more information as a snapshot.

9.4.2 How to use pgBadger

pgBadger is integrated with FUJITSU Enterprise Postgres, and it is installed as part of a FUJITSU Enterprise Postgres server installation.

The **pgBadger** command is installed to the following path:

```
<FUJITSU Enterprise Postgres Server Installation path>/
fsepv12server64/OSS/pgbadger/usr/bin
```

Example 9-67 shows our **pgBadger** installation path.

Example 9-67 The pgBadger path

```
[fsepuser@rdbkpgr3 ~]$ cd /opt/fsepv12server64/OSS/pgbadger/usr/bin/
[fsepuser@rdbkpgr3 bin]$ ls
pgbadger
```

FUJITSU Enterprise Postgres configuration settings of pgBadger

Before starting **pgBadger**, a few configuration parameters should be enabled in the `postgresql.conf` configuration file (Table 9-7 on page 361). The configuration parameters are set to ensure that sufficient information is captured in log files for **pgBadger** to analyze.

Table 9-7 FUJITSU Enterprise Postgres configuration settings of pgBadger

Configuration parameter	Default value	Example	Description
`log_min_duration_statement`	`-1`		Enables logging of the completed statements that ran for at least a specified number of milliseconds (0 for all queries).
`log_duration`	`off`	`off`	When this parameter is on, the elapsed time is logged for all the queries.
`log_statement`	`none`	`none`	Defines which SQL statements are logged.
`log_line_prefix`	`'%m [%p] '`		Information to be added at the start of a log message. It uses a `printf-styl` string.
`log_checkpoints`	`off`	`on`	When this parameter is on, logs the checkpoint execution.
`log_connections`	`off`	`on`	When this parameter is on, logs the client connection.
`log_disconnections`	`off`	`on`	When this parameter is on, logs the client disconnection.
`log_lock_waits`	`off`	`on`	When this parameter is on, captures the logs when a session waits longer than a time that is specified by **`deadlock_timeout`**.
`log_temp_files`	`-1`		Captures information about temporary files with a size greater than the specified value (in KB). 0 for all temp files.
`log_autovacuum_min_duration`	`-1`		Logs information about autovaccum activity that takes more than the specified time (in milliseconds). 0 means all.
`log_error_verbosity`	`default`	`default`	
`lc_messages`	`initdb`	`‘C’`	

Example 9-68 provides an example of how to set configuration parameters.

Example 9-68 Configuration parameters

```
log_min_duration_statement = 0
log_duration = off
log_statement = 'none'
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h'
log_checkpoints = on
log_connections = on
log_disconnections = on
log_lock_waits = on
log_temp_files = 0
log_autovacuum_min_duration = 0
lc_messages = 'C'
```

Restart the database instance for the configuration parameter changes to take effect, as shown in Example 9-69.

Example 9-69 Restarting the server

```
[fsepuser@rdbkpgr3 pg_log]$ pg_ctl -D /database/inst3/ restart
waiting for server to shut down.... done
server stopped
waiting for server to start....2020-11-25 00:55:23 EST [396969]:
user=,db=,app=,client=LOG:  starting PostgreSQL 12.1 on s390x-ibm-linux-gnu,
compiled by gcc (GCC) 8.3.1 20190507 (Red Hat 8.3.1-4), 64-bit
2020-11-25 00:55:23 EST [396969]: user=,db=,app=,client=LOG:  listening on IPv4
address "0.0.0.0", port 27502
2020-11-25 00:55:23 EST [396969]: user=,db=,app=,client=LOG:  listening on IPv6
address "::", port 27502
2020-11-25 00:55:23 EST [396969]: user=,db=,app=,client=LOG:  listening on Unix
socket "/tmp/.s.PGSQL.27502"
2020-11-25 00:55:23 EST [396969]: user=,db=,app=,client=LOG:  redirecting log
output to logging collector process
2020-11-25 00:55:23 EST [396969]: user=,db=,app=,client=HINT:  Future log output
will appear in directory "pg_log".
 done
server started
```

Log analysis by using pgBadger

Before analyzing the logs by using **pgBadger**, we collected some sample data by using **pgbench** with 50 clients and 10000 transactions. Example 9-70 provides an example of how we created the tables, generated data, and ran our test to collect sample data.

Example 9-70 Generating sample transactions

```
[fsepuser@rdbkpgr3 pg_log]$ pgbench -i test1 -p 27502
dropping old tables...
creating tables...
generating data...
100000 of 100000 tuples (100%) done (elapsed 0.07 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done.

[fsepuser@rdbkpgr3 pg_log]$ pgbench -c 50 -t 10000 test1 -p 27502
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 50
number of threads: 1
number of transactions per client: 10000
number of transactions actually processed: 500000/500000
latency average = 19.546 ms
tps = 2558.076866 (including connections establishing)
tps = 2558.092567 (excluding connections establishing)
```

By using **pgBadger**, you can analyze a single log file, multiple log files, and a set of different logs files by using shell commands. Example 9-71 analyzes the multiple logs files that are generated after running **pgbench**.

Example 9-71 Running pgBadger to analyze the generated logs

```
[fsepuser@rdbkpgr3 ~]$ pgbadger /database/inst3/pg_log/postgresql-2020-11-25_* -O
/opt/fsepv12server64/OSS/pgbadger/usr/output
[========================>] Parsed 614274863 bytes of 614274863 (100.00%),
queries: 3500031, events: 1
LOG: Ok, generating html report...
```

pgBadger generates a statistical report that is named `out.html` as the output to a log file analysis.

By using **pgBadger**, we can generate incremental reports and cumulative reports. It reports everything about SQL queries and represents these queries through interactive graphs. **pgBadger** automatically detects the log format (`stderr`, `csvlog`, `syslog`, or `jsonlog`). **pgBadger** can also parse connection polling/load balance tools such as Pgpool II, and report more information about sessions, connections, and throughput.

Figure 9-19 shows the statistical reports that are generated for our queries that are run through **pgbench**. The statistical report is categorized under different sections.

- Overview: Provides a high-level view of the overall SQL traffic, read and write queries, traffic, and query duration (Figure 9-19).

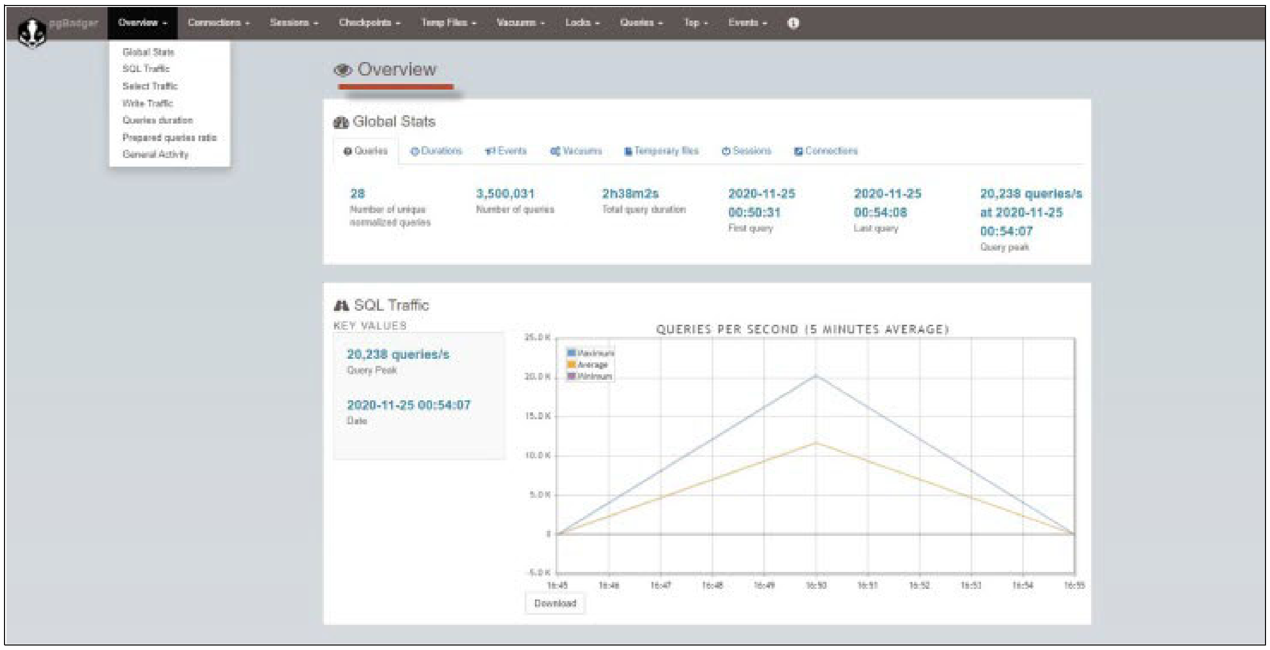


Figure 9-19 Statistical reports generated for the queries

- Connections: Provides connection-related information, total connections, and connections per database, per user, and per host (Figure 9-20).

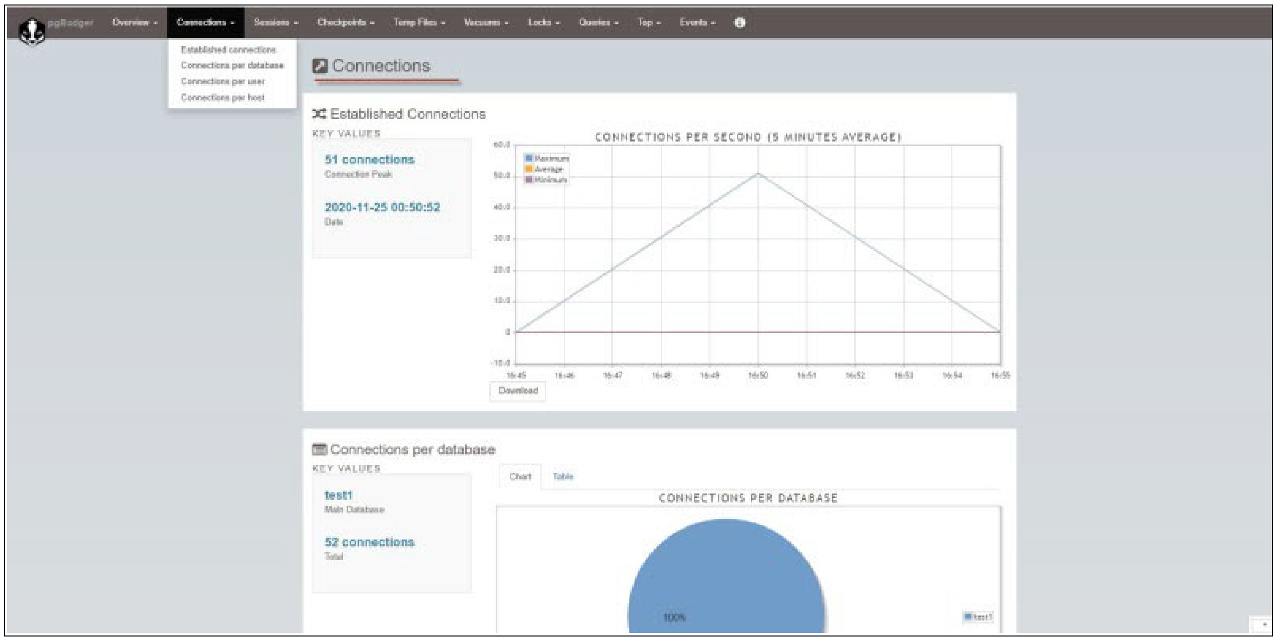


Figure 9-20 Connection-related statistics

- Sessions: Provides information that is related to sessions (Figure 9-21).

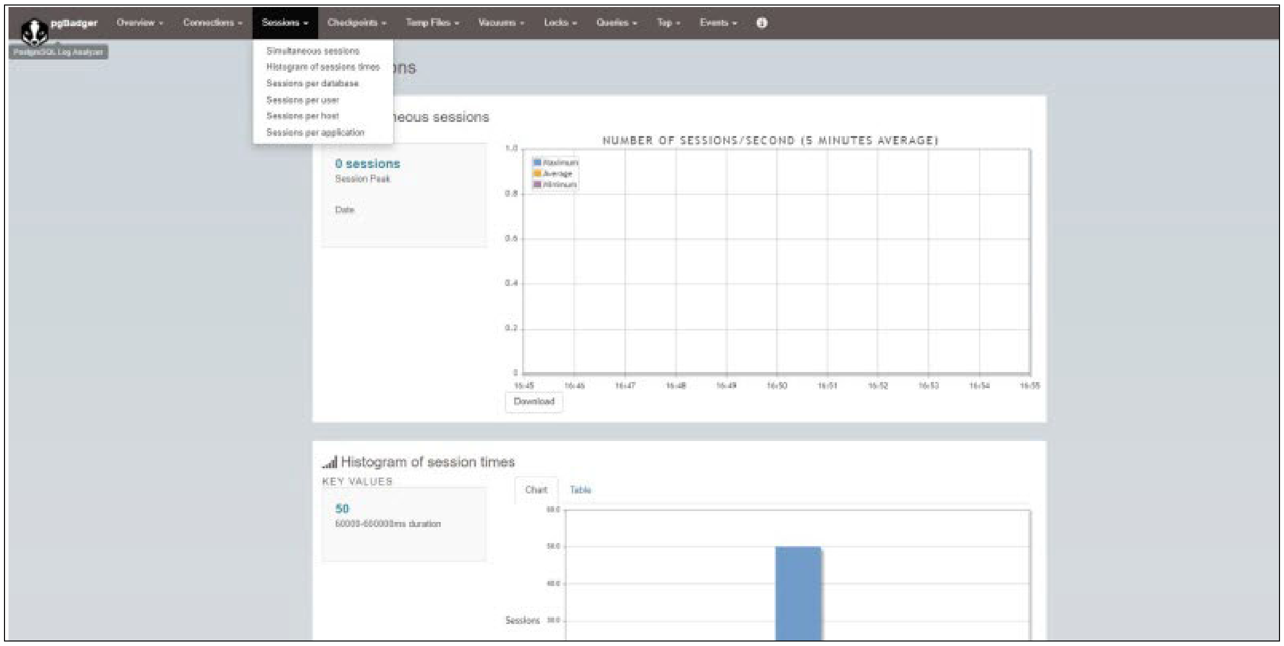
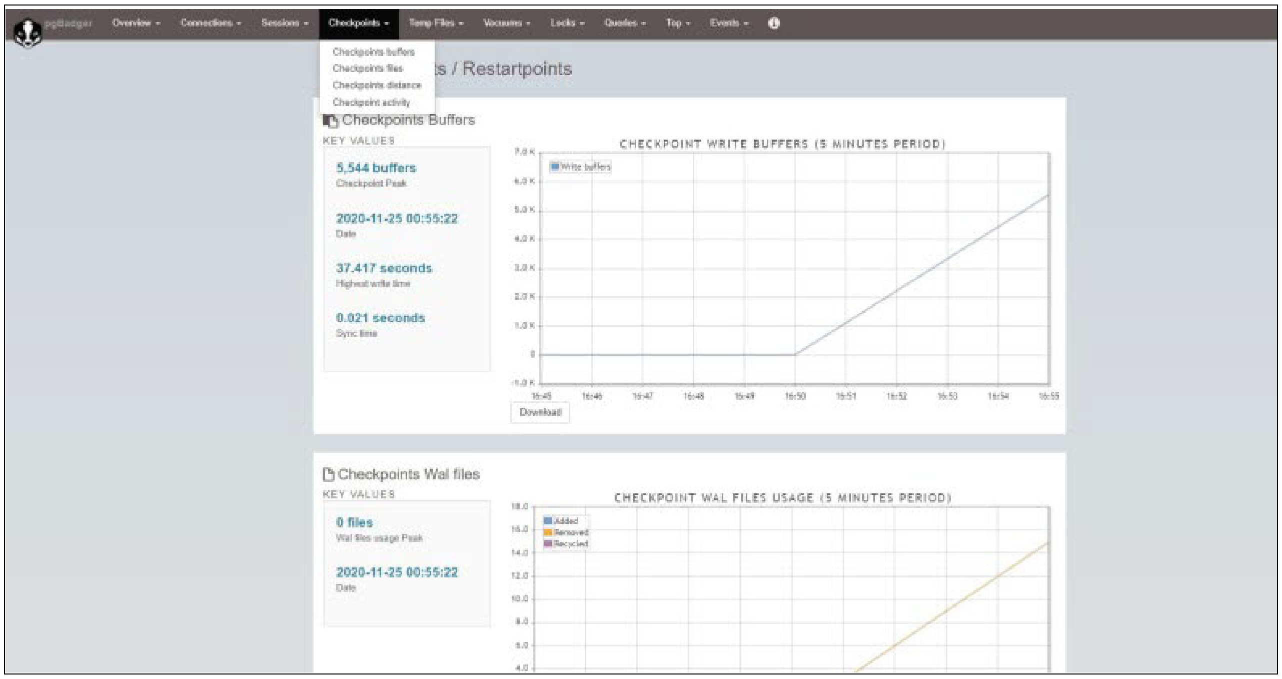


Figure 9-21 Sessions statistics information

- Checkpoints / Restartpoints: Provides information about checkpoint activity. We can further analyze the graphs to identify any bottlenecks (Figure 9-21).



Checkpoints Buffers

KEY VALUES

5,544 buffers
Checkpoint Peak

2020-11-25 00:55:22
Date

37.417 seconds
Highest write time

0.021 seconds
Sync time

Checkpoints Wal files

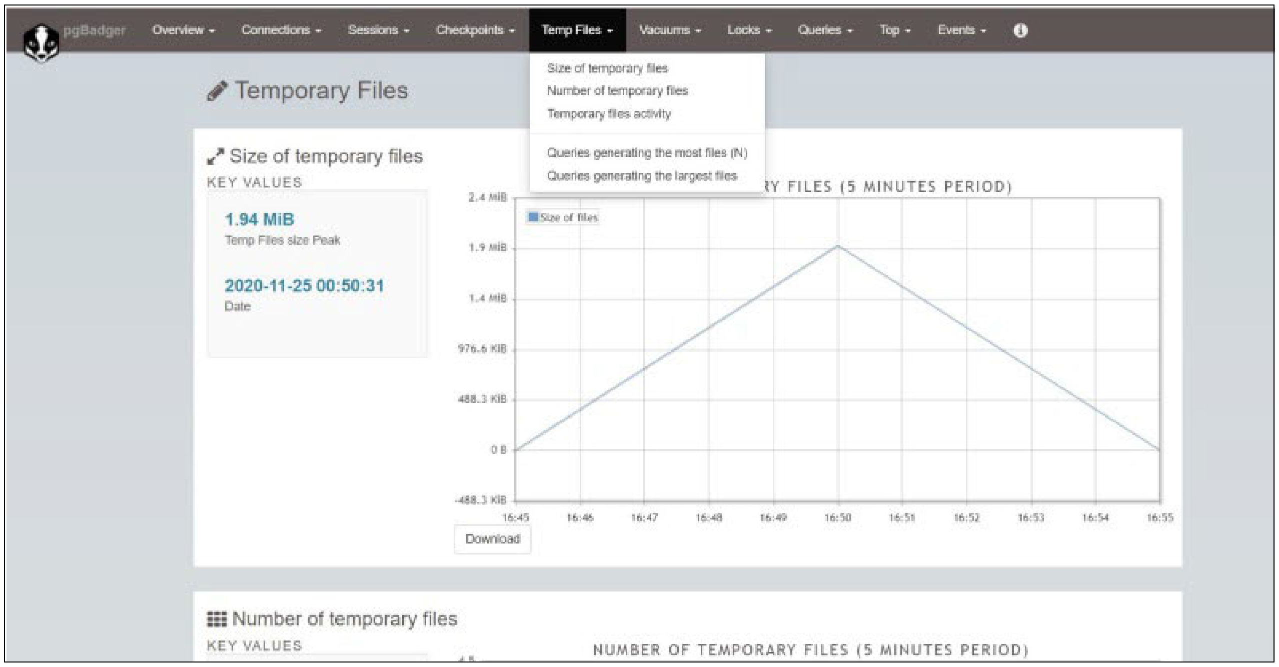
KEY VALUES

0 files
Wal files usage Peak

2020-11-25 00:55:22
Date

Figure 9-22 Checkpoints statistics

- Temporary files: Provides information about temporary files, such as the number and size of temporary files. It also identifies queries that generate the biggest temporary and highest number of temporary files. Using the graphs, we can identify the optimal Postgres configuration and define memory parameters that can improve system performance (Figure 9-23).


pgBadger
Overview
Connections
Sessions
Checkpoints
Temp Files
Vacuums
Locks
Queries
Top
Events
Size of temporary files
Number of temporary files
Temporary files activity
Queries generating the most files (N)
Queries generating the largest files
RY FILES (5 MINUTES PERIOD)
Size of files
Download
NUMBER OF TEMPORARY FILES (5 MINUTES PERIOD)

Temporary Files

Size of temporary files

KEY VALUES

1.94 MiB
Temp Files size Peak

2020-11-25 00:50:31
Date

Number of temporary files

KEY VALUES

Figure 9-23 Temporary files statistics

- Vacuums: Provides information about vacuum activity.
- Locks: Provides information about queries waiting to acquire the lock and lock types.

- Queries: Provides information about ran queries, and classifies queries by type, user, database, and application.

The statistical reports also rank queries by run time and frequency. By reviewing these reports, you can identify bottlenecks and improve database performance (Figure 9-24).

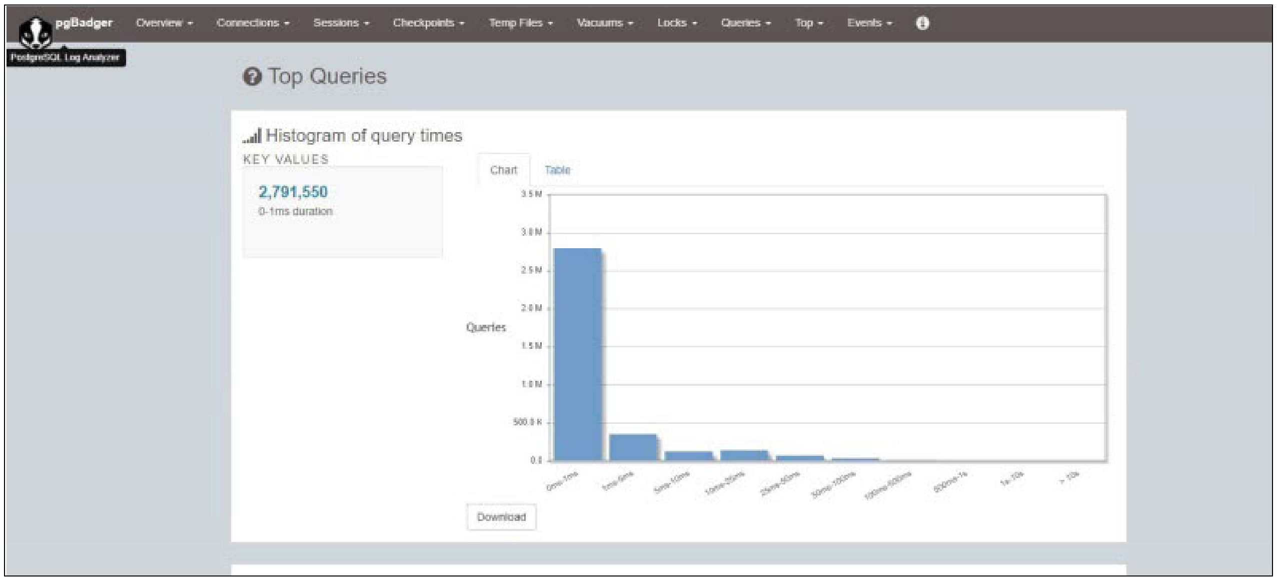
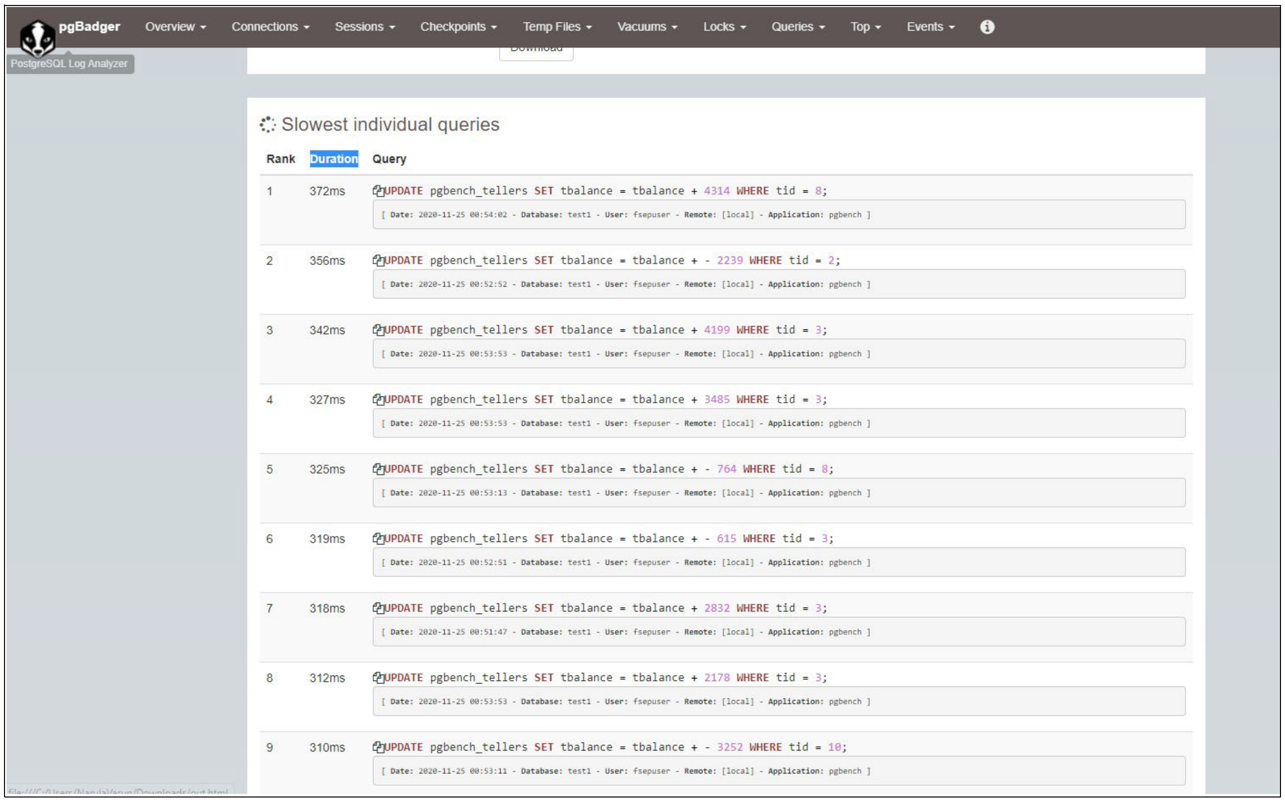


Figure 9-24 Histogram of query times

Along with the summary data, the statistical reports list rank, query run duration (Figure 9-25), and query details.



Rank	Duration	Query
1	372ms	UPDATE pgbench_tellers SET tbalance = tbalance + 4314 WHERE tid = 8; [Date: 2020-11-25 00:54:02 - Database: test1 - User: fsepuser - Remote: [local] - Application: pgbench]
2	356ms	UPDATE pgbench_tellers SET tbalance = tbalance + - 2239 WHERE tid = 2; [Date: 2020-11-25 00:52:52 - Database: test1 - User: fsepuser - Remote: [local] - Application: pgbench]
3	342ms	UPDATE pgbench_tellers SET tbalance = tbalance + 4199 WHERE tid = 3; [Date: 2020-11-25 00:53:53 - Database: test1 - User: fsepuser - Remote: [local] - Application: pgbench]
4	327ms	UPDATE pgbench_tellers SET tbalance = tbalance + 3485 WHERE tid = 3; [Date: 2020-11-25 00:53:53 - Database: test1 - User: fsepuser - Remote: [local] - Application: pgbench]
5	325ms	UPDATE pgbench_tellers SET tbalance = tbalance + - 764 WHERE tid = 8; [Date: 2020-11-25 00:53:13 - Database: test1 - User: fsepuser - Remote: [local] - Application: pgbench]
6	319ms	UPDATE pgbench_tellers SET tbalance = tbalance + - 615 WHERE tid = 3; [Date: 2020-11-25 00:52:51 - Database: test1 - User: fsepuser - Remote: [local] - Application: pgbench]
7	318ms	UPDATE pgbench_tellers SET tbalance = tbalance + 2832 WHERE tid = 3; [Date: 2020-11-25 00:51:47 - Database: test1 - User: fsepuser - Remote: [local] - Application: pgbench]
8	312ms	UPDATE pgbench_tellers SET tbalance = tbalance + 2178 WHERE tid = 3; [Date: 2020-11-25 00:53:53 - Database: test1 - User: fsepuser - Remote: [local] - Application: pgbench]
9	310ms	UPDATE pgbench_tellers SET tbalance = tbalance + - 3252 WHERE tid = 10; [Date: 2020-11-25 00:53:11 - Database: test1 - User: fsepuser - Remote: [local] - Application: pgbench]

Figure 9-25 Query run duration report

All the graphs are downloadable and saved as an image file (`.png`).

9.4.3 How to use pg_statsinfo

`pg_statsinfo` is a monitoring tool that collects statistics about PostgreSQL server activities at defined time intervals. Collected information gets stored as a snapshot into repository database on another or the same PostgreSQL server.

`pg_statsinfo` is integrated with FUJITTSU Enterprise Postgres, and it is installed as part of the FUJITSU Enterprise Postgres server installation.

`pg_statsinfo` is installed in `<FUJITSU Enterprise Postgres Server Installation path>/ fsepv12server64/OSS/pg_statsinfo/bin`. Our path is shown in Example 9-72.

Example 9-72 pg_statsinfo path

```
[fsepuser@rdbkpgr3 ~]$ cd opt/fsepv12server64/OSS/pg_statsinfo/bin
[fsepuser@rdbkpgr3 bin]$ ls
archive_pglog.sh  pg_statsinfo  pg_statsinfod
```

FUJITSU Enterprise Postgres configuration settings of pg_statsinfo

In this publication, we use the same PostgreSQL server as a repository database to store snapshot information. The configuration settings that are shown in Example 9-73 were used to run **`pg_statsinfo`**.

Example 9-73 Configuration parameters

```
shared_preload_libraries = 'pg_statsinfo'       # preinstall pg_statsinfo
libraries
pg_statsinfo.snapshot_interval = 1min          # snapshot interval
pg_statsinfo.enable_maintenance = 'on'          # enable automatic maintenance
('on' or 'off')
pg_statsinfo.maintenance_time = '00:02:00'      # delete old snapshots every day
at this time.
pg_statsinfo.repository_keepday = 7             # keep snapshots for this period.
pg_statsinfo.repolog_keepday = 7               # keep logs in repository for this
period.
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log' # pg_statsinfo requires this
log_filename setting
log_min_messages = 'log'
pg_statsinfo.syslog_min_messages = 'error'
pg_statsinfo.textlog_line_prefix = '%t %p %c-%l %x %q(%u, %d, %r, %a) '
pg_statsinfo.syslog_line_prefix = '%t %p %c-%l %x %q(%u, %d, %r, %a) '
track_functions = 'all'
log_checkpoints = on
log_autovacuum_min_duration = 0
```

Restart the database instance for the configuration parameter changes to take effect, as shown in Example 9-74.

Example 9-74 Restarting the server

```
[fsepuser@rdbkpgr3 inst3]$ pg_ctl -D /database/inst3 restart
waiting for server to shut down.... done
server stopped
waiting for server to start....2020-12-01 18:51:39 EST [499462]:
user=,db=,app=,client=LOG:  starting PostgreSQL 12.1 on s390x-ibm-linux-gnu,
compiled by gcc (GCC) 8.3.1 20190507 (Red Hat 8.3.1-4), 64-bit
2020-12-01 18:51:39 EST [499462]: user=,db=,app=,client=LOG:  listening on IPv4
address "0.0.0.0", port 27502
2020-12-01 18:51:39 EST [499462]: user=,db=,app=,client=LOG:  listening on IPv6
address "::", port 27502
2020-12-01 18:51:39 EST [499462]: user=,db=,app=,client=LOG:  listening on Unix
socket "/tmp/.s.PGSQL.27502"
2020-12-01 18:51:39 EST [499462]: user=,db=,app=,client=LOG:  redirecting log
output to logging collector process
2020-12-01 18:51:39 EST [499462]: user=,db=,app=,client=HINT:  Future log output
will appear in directory "pg_log".
 done
server started
```

Starting and stopping the service

pg_statsinfo automatically starts running on FUJITSU Enterprise Postgres startup. You can individually stop and start **pg_statsinfo**, as shown in Example 9-75.

Example 9-75 Starting and stopping pg_statsinfo

```
[fsepuser@rdbkpgr3 ~]$ pg_statsinfo --stop -d postgres -p 27502
LOG:  waiting for pg_statsinfod to shut down
LOG:  pg_statsinfod stopped
[fsepuser@rdbkpgr3 ~]$ pg_statsinfo --start -d postgres -p 27502
LOG:  waiting for pg_statsinfod to start
LOG:  pg_statsinfod started
```

Capturing and deleting the snapshots

Based on the snapshot interval parameter, **pg_statsinfo** automatically captures the snapshot information. It is also possible to take the snapshot manually, as shown in Example 9-76.

Example 9-76 Manual snapshot

```
[fsepuser@rdbkpgr3 ~]$ psql -d postgres -p 27502 -c "Select
statsinfo.snapshot('Manual Snapshot')";
 snapshot
---------
(One row)
OR
[fsepuser@rdbkpgr3 ~]$ pg_statsinfo –S ‘Manual Snapshot’ –p 27502 –d postgres
```

We can query the snapshot table to list available snapshots (Example 9-77 on page 369).

Example 9-77 Listing available snapshots

```
[fsepuser@rdbkpgr3 pg_log]$ pg_statsinfo -l -p 27502 -d Postgres
OR
[fsepuser@rdbkpgr3 ~]$ psql -d postgres -p 27502 -c "Select * from
statsrepo.snapshot";
 snapid | instid |            time              |     comment     |    exec_time
| snapshot_increase_size
-------+--------+-------------------------------+-----------------+---------------
     47 |      1 | 2020-12-01 21:57:00.103774-05 |                 |
00:00:00.713888 |                155648
     48 |      1 | 2020-12-01 21:58:00.047281-05 |                 |
00:00:01.090892 |                155648
     49 |      1 | 2020-12-01 21:59:00.076021-05 |                 |
00:00:00.574306 |                155648
     50 |      1 | 2020-12-01 21:59:20.580153-05 | Manual Snapshot |
00:00:01.689032 |                172032
     51 |      1 | 2020-12-01 22:00:00.009527-05 |                 |
00:00:02.509178 |                180224
     52 |      1 | 2020-12-01 22:01:00.007746-05 |                 |
00:00:01.519731 |                139264
     53 |      1 | 2020-12-01 22:02:00.15097-05  |                 |
00:00:02.073381 |                172032
     54 |      1 | 2020-12-01 22:03:00.023195-05 |                 |
00:00:02.923532 |                155648
     55 |      1 | 2020-12-01 22:04:00.15846-05  |                 |
00:00:01.58432  |                163840
     56 |      1 | 2020-12-01 22:05:00.066228-05 |                 |
00:00:02.086312 |                147456
     57 |      1 | 2020-12-01 22:06:00.136042-05 |                 |
00:00:02.633935 |                172032
     58 |      1 | 2020-12-01 22:07:00.047069-05 |                 |
00:00:01.741157 |                180224
```

We set the snapshot interval as 1 minute for demonstration purposes, so a new snapshot is created every minute. By default, the snapshot interval is 10 minutes. We also created a manual snapshot, which is listed with comments as “Manual Snapshot”.

Over time, the size of the repository increases if old snapshots are not maintained properly. **pg_statsinfo** deletes the old snapshots when the repository maintenance function is turned on.

Snapshots older than the specific timestamp can be deleted manually by using the following command, and snapshots can also be deleted by specifying the snapshot ID:

```
[fsepuser@rdbkpgr3 pg_log]$ pg_statsinfo -D 58 -p 27502 -d postgres
```

The number 58 in the command is the snapshot ID.

Enabling alerting

pg_statsinfo supports alerting based on the threshold value that is defined in the repository alert table. To turn off the alerts for a specific property, set the threshold value to -1. When alerts are enabled, **pg_statsinfo** writes the alert to a repository and text log file when the property value goes beyond the threshold value that is defined in the alert table. Example 9-78 shows our example.

Example 9-78 Current threshold values in the alert table

```
postgres=# select * from statsrepo.alert;
 instid | rollback_tps | commit_tps | garbage_size | garbage_percent |
garbage_percent_table | response_avg | response_worst | backend_
max | correlation_percent | disk_remain_percent | loadavg_1min | loadavg_5min |
loadavg_15min | swap_size | rep_flush_delay | rep_repla
y_delay | enable_alert
-------+--------------+------------+--------------+-----------------+------------
1 |          100 |       1000 |           -1 |              30 |
30 |          10 |            60 |
100 |              70 |                20 |           7 |           6 |
5 |   1000000 |            100 |
    200 | t
(One row)
```

You can set the commit threshold by setting **commit_tps** by using the command in Example 9-79.

Example 9-79 Updating commit_tps value

```
postgres=# update statsrepo.alert set commit_tps = 4000;
UPDATE 1

postgres=# select * from statsrepo.alert;
 instid | rollback_tps | commit_tps | garbage_size | garbage_percent |
garbage_percent_table | response_avg | response_worst | backend_
max | correlation_percent | disk_remain_percent | loadavg_1min | loadavg_5min |
loadavg_15min | swap_size | rep_flush_delay | rep_repla
y_delay | enable_alert
-------+--------------+------------+--------------+-----------------+------------
1 |          100 |       4000 |           -1 |              30 |
30 |          10 |            60 |
100 |              70 |                20 |           7 |           6 |
5 |   1000000 |            100 |
    200 | t
(One row)
```

Update the **enable_alert** value to false to disable the alerts by using the following command:

```
postgres=# update statsrepo.alert set enable_alert = false;
```

Generating the report

A report can be generated from the statistics that are collected by **pg_statsinfo** and stored in a repository database. The CLI takes **REPORTID** as an input and generates the report in readable text format.

To get the complete list of **REPORTID** options, use the **-help** option with **pg_statsinfo**. An example of this command is shown in Example 9-80 on page 371.

Example 9-80 Viewing the list of all options by using the help option

```
[fsepuser@rdbkpgr3 pg_log]$ pg_statsinfo –help
pg_statsinfo reports a PostgreSQL database.
Usage:
  pg_statsinfo -r REPORTID [-i INSTANCEID] [-b SNAPID] [-e SNAPID] [-B DATE] [-E
DATE]
                           [-o FILENAME] [connection-options]
  pg_statsinfo -l          [-i INSTANCEID] [connection-options]
  pg_statsinfo -s          [connection-options]
  pg_statsinfo -S COMMENT  [connection-options]
  pg_statsinfo -D SNAPID   [connection-options]
  pg_statsinfo --start     [connection-options]
  pg_statsinfo --stop      [connection-options]
General options:
 -r, --report=REPORTID  generate a report that specified by REPORTID
                        ---------------------------
                          * Summary
                          * Alert
                          * DatabaseStatistics
                          * InstanceActivity
                          * OSResourceUsage
                          * DiskUsage
                          * LongTransactions
                          * NotableTables
                          * CheckpointActivity
                          * AutovacuumActivity
                          * QueryActivity
                          * LockConflicts
                          * ReplicationActivity
                          * SettingParameters
                          * SchemaInformation
                          * Profiles
                          * All
                        ---------------------------
                         (can prefix match. For example, "su" means 'Summary')
 -i, --instid           limit to instances of specified instance ID
 -b, --beginid          begin point of report scope (specify by snapshot ID)
 -B, --begindate        begin point of report scope (specify by timestamp)
 -e, --endid            end point of report scope (specify by snapshot ID)
 -E, --enddate          end point of report scope (specify by timestamp)
 -l, --list             show the snapshot list
 -s, --size             show the snapshot size
 -S, --snapshot=COMMENT get a snapshot
 -D, --delete=SNAPID    delete a snapshot
 --start                start the pg_statsinfo agent
 --stop                 stop the pg_statsinfo agent
Output options:
 -o, --output=FILENAME  destination file path for report
Connection options:
 -d, --dbname=DBNAME       database to connect
 -h, --host=HOSTNAME       database server host or socket directory
 -p, --port=PORT           database server port
 -U, --username=USERNAME   username to connect as
 -w, --no-password         never prompt for password
 -W, --password            force password prompt
```

```
Generic options:
 --echo                     echo queries
 --elevel=LEVEL             set output message level
 --help                     show this help, then exit
 --version                  output version information, then exit
```

For more information, see The pg_statsinfo Project.

To create a summary report, run the command that is shown in Example 9-81.

Example 9-81 Generating a summary report

```
[fsepuser@rdbkpgr3 pg_log]$ pg_statsinfo -r Summary -p 27502 -d postgres
--------------------------------------------
STATSINFO Report (host: rdbkpgr3.pbm.ihost.com, port: 27502)
--------------------------------------------
---------------------------------------
/* Summary */
---------------------------------------
Database System ID   : 6896303172262049094
Host                 : rdbkpgr3.pbm.ihost.com
Port                 : 27502
PostgreSQL Version   : 12.1
Snapshot Begin       : 2020-12-01 20:00:00
Snapshot End         : 2020-12-01 23:52:28
Snapshot Duration    : 03:52:27
Total Database Size  : 6771 MiB
Total Commits        : -840855
Total Rollbacks      : -42
```

To generate a report, run the command that is shown in Example 9-82.

Example 9-82 Generating a report of all REPORTID

```
[fsepuser@rdbkpgr3 pg_log]$ pg_statsinfo -r All -p 27502 -d Postgres
# generate a detailed report for all the report ID’s
```

Note: The information that is collected through **`pg_statsinfo`** can be viewed graphically by using an extra tool that is called **`pg_stats_reporter`**.

10

FUJITSU Enterprise Postgres cluster on Red Hat OpenShift Container Platform

This chapter introduces the concepts of container orchestration and management and how they can be applied to set up a container platform for software delivery on IBM infrastructure, including setting up a FUJITSU Enterprise Postgres SQL database cluster.

This chapter covers steps for setup, configuration, and operational management of a FUJITSU Enterprise Postgres cluster within a customer's DevOps by using agile software delivery practices that use Red Hat OpenShift on an IBM LinuxONE platform. Multiple factors come together with FUJITSU Enterprise Postgres cloud-native deployment on the IBM and Red Hat hybrid multi-cloud solution. A reliable and robust relational database that is based on open source PostgreSQL is containerized, which implements the operator pattern and is oriented around micro-services.

FUJITSU Enterprise Postgres deploys on IBM LinuxONE, which is an enterprise server platform in IBM systems that is designed for security, resilience, and scalability. It is server hardware that is combined with the openness of the Linux operating system (OS) and enterprise scale Kubernetes from Red Hat.

This chapter covers the following topics:

- Container-based FUJITSU Enterprise Postgres
- Container orchestration: Red Hat OpenShift on an IBM infrastructure
- FUJITSU Enterprise Postgres on Red Hat OpenShift
- FUJITSU Enterprise Postgres Operator operations
- FUJITSU Enterprise Postgres cluster CR configuration file explanation

10.1 Container-based FUJITSU Enterprise Postgres

As organizations continue to focus on their digital transformation, they must change the way that they manage their existing data assets. Migration to cloud is often seen as a fundamental step in an organization's digital transformation, and as applications migrate to the cloud, often databases naturally follow them into cloud environments such as AWS, Azure, and IBM Cloud. The proliferation of the number of cloud-hosted applications with a tremendous increase in the scale of the data generated, and the data that is consumed by such applications, creates an unprecedented demand for cloud-hosted databases.

Containers and container orchestration have matured to the point where they are now positioned at the core of digital transformation by using cloud-native initiatives. Databases are increasingly popular candidates for containerization, and are becoming an on-demand utility as part of the shift toward microservices and serverless architectures, which are becoming a critical component in the software stack of these new digital applications.

An installation that is composed of an enterprise-grade database such as FUJITSU Enterprise Postgres running in a container environment can provide many advantages over a standard cloud database. With the transition of applications to a microservices architecture, the container deployment of relational database management systems (RDBMSs) is moving to the mainstream, and PostgreSQL is the third most popular technology to run in containers on a Kubernetes platform that manages applications at massive scale.

Containerized databases have emerged as building blocks for shifting large, monolithic applications to workloads based on microservices and serverless architectures. However, containerizing a database is not as straightforward as containerizing an application. Fujitsu has leveraged its Enterprise PostgreSQL knowledge and experience and created a Managed Postgres Cluster offering to organizations to use as part of their cloud-native container journey.

Containerized FUJITSU Enterprise Postgres key benefits

The FUJITSU Enterprise Postgres Database can be easily set up in a container platform to provide an enterprise-level database that is comparable to the cloud-managed Amazon RDS DBaaS without the disadvantages of the latter, such as data breaches, data loss, and inflexibility. The FUJITSU Enterprise Postgres Database in managed containerized environment provides the following benefits:

- On-demand utility: The microservices architecture of smaller container applications can have their own dedicated database that is configured on an as-needed basis.
- Capacity planning and provisioning: Using separate storage nodes, Kubernetes Storage Class, and Persistent Volume features, storage performance and capacity can be scaled independently of compute resources, which provide more flexibility in upfront database capacity and makes changes easier to implement.
- High-velocity DevOps: Software-defined containerized databases provide a crucial missing link in high-velocity DevOps cycles, allowing DevOps teams to collaborate seamlessly.
- High-throughput and low-latency networking: The emergence of container orchestration such as Red Hat OpenShift provides networking and data storage and network resource isolation that is necessary to achieve the required performance.
- Managing numerous tuning parameters: Custom database configuration parameters that use ConfigMaps and Custom Resource Definitions (CRDs) within the Kubernetes cluster can be passed into the container at run time to provide easier version control and dynamic control of DB management.

- High availability (HA) and failover management: Using an automated scripted deployment of containerized databases and an orchestration framework of Red Hat OpenShift provides a built-in HA for failover scenarios without needing to maintain a failover cluster replica. This setup saves resources that are idle much of the time.
- Hybrid cloud / multi-cloud deployments: FUJITSU Enterprise Postgres running on a Red Hat OpenShift based Kubernetes platform provides a cloud-neutral RDBMS solution that can be deployed across any cloud environment-supporting hybrid / multi-cloud strategy without needing to change the application code.
- Software upgrades: By leveraging the Red Hat OpenShift Operator framework, it is easier to plan and run a database software upgrade with or without schema changes. The software upgrade can be easily managed to minimize business disruptions, including the option to skip upgrades to keep compatibility with other applications, which removes a critical dependency with Amazon RDS. This level of control interferes with an automatic environment upgrade process.

10.1.1 Containerized FUJITSU Enterprise Postgres Database on IBM hardware

FUJITSU Enterprise Postgres is available in a cloud-native containerized configuration that can be deployed on Red Hat OpenShift on various IBM infrastructure options. IBM hardware and software that is combined with Red Hat platform middleware can help organizations with establishing a true enterprise-class, hybrid multi-cloud solution that is highly scalable and open. IBM LinuxONE is the enterprise server platform with security and reliability in the hardware and software that is designed to run millions of containers. To make the best decisions to run FUJITSU Enterprise Postgres on Red Hat OpenShift, there are choices to consider.

Note: At the time of writing, a container version of FUJITSU Enterprise Postgres was made available for IBM LinuxONE and tested for deployment on Red Hat OpenShift on the IBM LinuxONE platform.

10.2 Container orchestration: Red Hat OpenShift on an IBM infrastructure

Kubernetes and container technologies emerged when the cloud expanded beyond provider-offered infrastructures to platform-as-a-service (PaaS) or software-as-a-service (SaaS). For over a decade now, cloud computing has rapidly evolved, which left many organizations behind due to the lack of service experiences, such as managing compute and storage resources on demand, and running modular pieces of code in a serverless environment. More often, moving to a cloud-native infrastructure is the only path to maintain a competitive advantage. There are enterprise-grade Kubernetes distributions and container infrastructures that can simplify that journey.

10.2.1 Containers and Kubernetes

Containers run on top of the host OS to abstract away the underlying infrastructure and allow grouping and isolation of the processes of the application. They gained popularity among developers and IT operations in various cloud computing models, whether as private, public, or hybrid. Deploying, maintaining, and scaling containers on different platforms and clouds is easier than virtual servers.

Software that is built with containers encapsulates the entire execution environment, including an application, and its runtime, dependencies, and configuration files, which are packaged into one image (Figure 10-1). A containerized application can install and run in a predictable way because it is less sensitive to the differences in the infrastructure, platform, library versions, OS level, and distribution. Containers can be deployed on many hardware platforms, including Intel x86, IBM Power, IBM LinuxONE, and IBM Z.

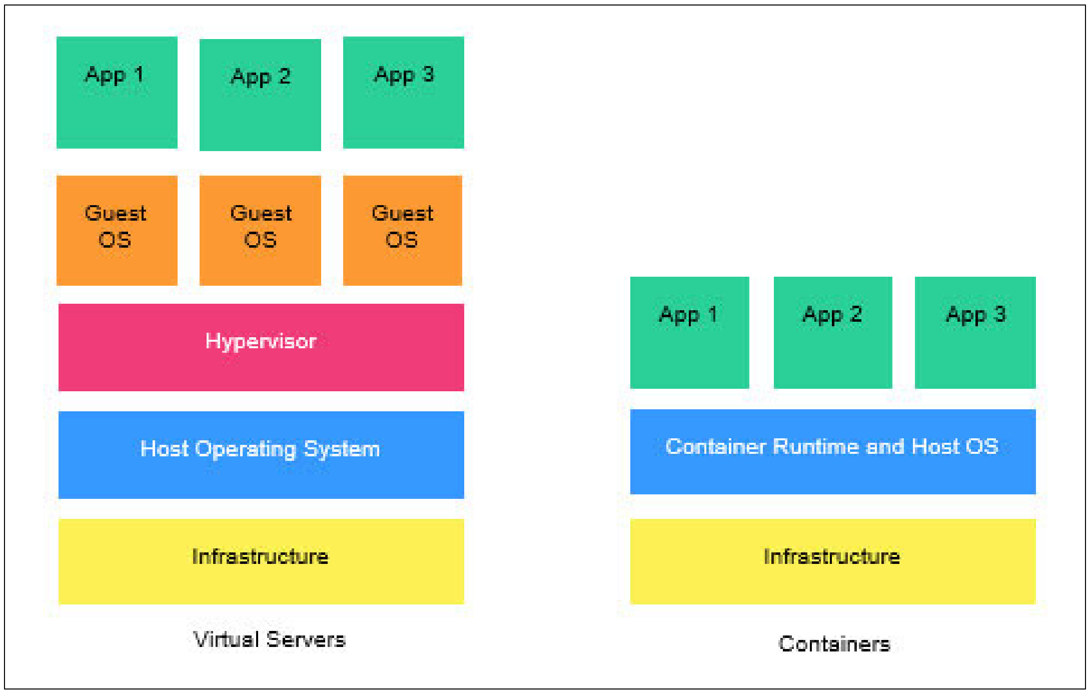


Figure 10-1 Containers encapsulate the entire execution environment

Containers offer great portability across hybrid cloud environments. However, complex applications consist of multiple containers and require mechanisms of availability and scalability to run in production. Kubernetes is an open source framework with a rich set of complex features to orchestrate and manage containers in production, staging, and development environments. As a layer over containers and container engine, it handles the workload scheduling and operation within an application and more (Figure 10-2).

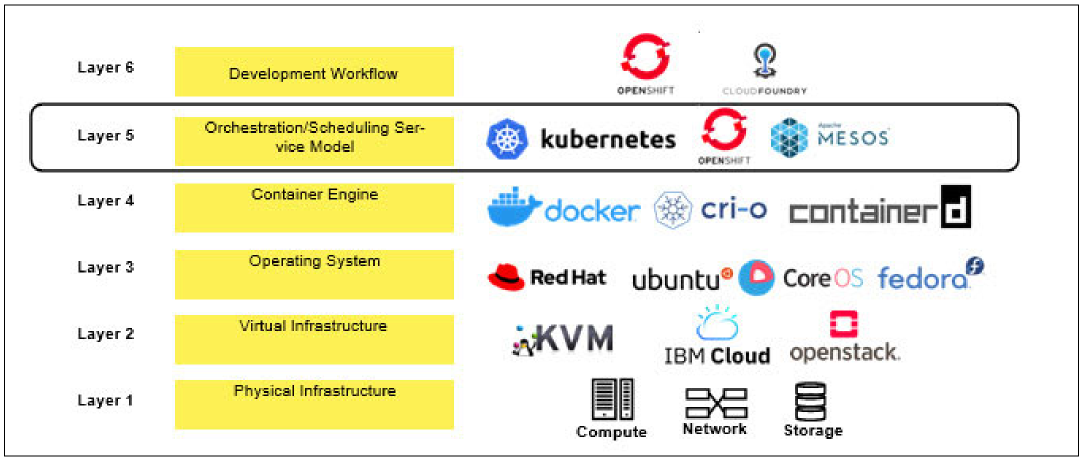


Figure 10-2 Orchestration layer in the software development workflow

As a cluster management system, Kubernetes automates deployment, scaling, and management of containers. It is the environment that ensures availability and scalability of the applications, and it supports DevOps processes for continuous integration and delivery. When containers are deployed, the Kubernetes scheduler automatically places them across the nodes in the cluster based on the resource requirements and constraints. Lifecycle orchestration helps with scaling and updating applications. When containers fail or do not respond, the self-healing mechanism of Kubernetes replaces or stops them to ensure the availability of the application. Service discovery and load-balancing can expose services of an application through a hostname or IP address, and balance the traffic across multiple containers depending on the workload.

10.2.2 Red Hat OpenShift 4

Organizations want flexibility, simplicity, and a better economic equation with their applications in the cloud. IBM hardware and software that is combined with Red Hat platform middleware can help organizations with establishing a true enterprise-class hybrid multi-cloud solution that is highly scalable and open. They want faster development cycle times, lower costs, security and availability, and consistency in managing, building, and operating the cloud.

As one of the on-premises cloud configurations, Red Hat OpenShift provides enterprise-scale Kubernetes on an enterprise-grade IBM LinuxONE server. IBM LinuxONE offers a flexible compute function with horizontal scalability that can grow to thousands of Linux guests and millions of containers, and vertical scalability that can non-disruptively grow databases without requiring any partitioning or sharding. Red Hat OpenShift Container Platform (RHOCP) 4 is a supported Kubernetes installation that uses leading enterprise practices; security, performance, and defect fixes; and includes validated and tested integrations for third-party plug-ins with enterprise lifecycle support.

Unlike Kubernetes that requires an OS to be installed separately, the Red Hat OpenShift 4 package comes with Red Hat Enterprise Linux CoreOS (RHCOS) included. It provides the required interfacing with the hardware, including IBM LinuxONE for the S390x package. RHCOS was designed and optimized to be deployed in the RHOCP platform (Figure 10-3).

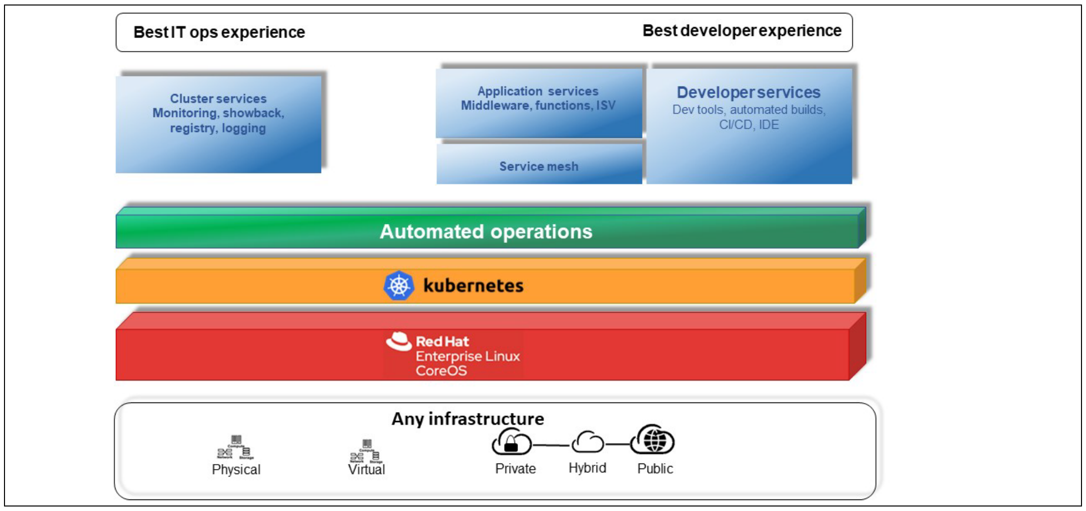


Figure 10-3 Red Hat OpenShift 4: Kubernetes platform

Upon initial deployment, RHCOS is hardened for security to a limited immutability. Some special features such as kernel options, disk encryption, and networking settings can be configured before the installation. The ignition configuration offers the opportunity to add custom `systemd` services and files to the RHCOS nodes. Finally, after the RHOCP cluster is running, DaemonSet and MachineConfig allow hardening of more services or subsets of ignition configurations on the nodes. Machine Config Operator also handles OS upgrades, where the OS is upgraded as an atomic unit.

10.2.3 Red Hat OpenShift 4 for Kubernetes developer experience

Red Hat OpenShift 4 is Kubernetes as fully open source and non-proprietary but way smarter. APIs to the Red Hat OpenShift cluster are 100% Kubernetes. Running a container on any Kubernetes and Red Hat OpenShift does not require any changes in the application. Red Hat OpenShift brings added-value features that make it a turn-key platform from both a PaaS developer perspective and a container-as-a-service (CaaS) operations perspective.

Red Hat OpenShift runs on a trusted OS foundation: RHCOS with a Red Hat Enterprise Linux server. It comes with automated installation, admin, and upgrade operations. Because of the rich infrastructure of operators, many containerized products can be easily installed from the catalog and operated from a unified interface within the Red Hat OpenShift cluster.

The developer experience is achieved by using a developer console and CodeReady Workspaces. Developers can easily deploy applications from different sources, including git repos, external registries, and Dockerfiles; get a visual representation of components; and modify code in a fully containerized web IDE that runs on Red Hat OpenShift itself.

Red Hat OpenShift can run containerized traditional stateful applications alongside microservices and serverless cloud-native applications. Achieving enterprise-grade qualities of the applications is simplified by using cluster services, like monitoring and alerts.

Application and line-of-business teams benefit from Red Hat OpenShift self-service capabilities for the resources that they need, which reduces wait times for productivity. Developers can take advantage of both containerized applications and orchestration without knowing the details, which give them a better focus on the code instead of spending time writing Dockerfiles and running Docker builds. Containers and cloud-native applications increase complexity for IT teams, who can benefit from Red Hat OpenShift improved automation, which means less manual and interrupt-driven work. Red Hat OpenShift gives operations teams the capability for more uniform, policy-driven control across multiple teams.

Here are the Red Hat OpenShift advantages for various teams:

- Developers: Container platform and cloud-native applications:
  - Who: Application architects, application development leads, and individual developers
  - Key benefits:
    - Reduced wait times for resource provisioning
    - Flexible choice of tools and workflows
    - Consistency between environments
- IT operations: Container platform and hybrid cloud infrastructure:
  - Who: Infrastructure, operations, and DevOps leaders and their teams
  - Key benefits:
    - Security controls, for example, do-it-yourself (DIY) solutions
    - Platform installations and updates

    - Less downtime
    - Enabling application development teams
- Technical executives: Business innovation:
  - Who: Chief technology officer (CTO) office, chief information officer (CIO) office, enterprise architect, and line-of-business leaders
  - Key benefits:
    - Cohesion between siloed teams
    - Faster speed to market
    - Higher return on investment (ROI)
    - Lower cost of operations

Red Hat OpenShift has everything that you need for a hybrid cloud, enterprise containers, and Kubernetes development and deployments. The platform has the following features:

- Enables broad application support.
- Drives continuous security.
- Supports on-demand as-a-service marketplaces.
- Is based on standards around interoperability and portability.
- Helps to avoid vendor lock-ins.

It includes an enterprise-grade Linux OS, container runtime, networking, monitoring, container registry, authentication, and authorization solutions that are tested together for unified operations on a complete Kubernetes platform spanning every cloud infrastructure.

10.2.4 Modernizing core business applications by using IBM Cloud Paks

IBM Software along with Red Hat and IBM LinuxONE further accelerate the digital transformation by unleashing open innovations. IBM Cloud Paks are enterprise-ready, containerized software solutions that give companies an open, faster, and more secure way to move core business applications to any cloud. IBM Cloud Paks for Application, Data, Integration, Automation, Multicloud, and Security run on Red Hat OpenShift on various infrastructure options. Refer to the latest release schedule for their availability on IBM LinuxONE.

Table 10-1 compares client-created containers to certified containers.

Table 10-1 Client-created versus IBM Cloud Paks certified containers

Feature	**Containers alone: Client creates containers or receives software as stand-alone containers**	**IBM Cloud Paks: Complete solutions certified for enterprise use cases**
Runs anywhere.	Yes	Yes
Scans for vulnerabilities.	Yes	Yes
Red Hat container certification.	Depends on product.	Yes
Complete solution with a container platform.	No	Yes
Flexible and modular: Pay for what you use.	No	Yes

Feature	Containers alone: Client creates containers or receives software as stand-alone containers	IBM Cloud Paks: Complete solutions certified for enterprise use cases
IBM certified or orchestrated for production (built for Kubernetes by experts, and certified against 250+ criteria).	No	Yes
Multicloud validation.	No	Yes
Integrated deployment experience.	No	Yes
Full stack support by IBM (base OS, software, and container platform).	No	Yes
License metering integration.	No	Yes
Scalable and resilient.	No	Yes
Encrypted secrets / limited privileges.	Do it yourself.	Yes
Management and operations.	Build your own.	Yes
Lifecycle management	Manage it yourself.	Yes

Note: At the time of this writing, IBM Cloud Pak® for Applications, major parts of IBM Cloud Paks for Integration and Multicloud Manager, and a subset of IBM Cloud Pak for Data were made available for IBM LinuxONE.

With all the benefits of cost, speed, user experience, security, and core ownership of the hybrid clouds, there are complexities that cannot be mastered by enterprises alone. The industry expertise and best practices that are packed into IBM Cloud Paks provide significant value to companies because these containers provide pre-packaged and integrated solutions on top of Red Hat OpenShift. They are IBM-certified, enterprise-ready, containerized, and pre-integrated software solutions that provide an open, faster, and more reliable way to build, move, and manage on any cloud. Each IBM Cloud Pak includes RHOCP, IBM certified containerized software and open source components, and common software services for development and management on top of a common integration layer.

10.2.5 Private and public hosting options

With the rise of expectations for a better user experience and connectivity among various devices, application developers strive to deliver intelligence and submillisecond response times. A centralized cloud architecture alone cannot keep up with such requirements, so the flexibility of splitting workloads into on-premises and cloud-based areas in hybrid multicloud environments is necessary. Faster response times and deeper data insights might be achieved by hosting data and processing it closer to where it originates, that is, at the edge. Unlocking new sources of value might be achieved by interconnecting applications across sites and an infrastructure of partners in the public cloud.

Red Hat OpenShift is built on the Kubernetes platform with portability in mind. It is offered on various public and private clouds, which make cloud differences invisible to developers. Many enterprises realize the future of cloud computing is in multicloud solutions, where hybrid environments span on-premises, public, and private clouds from multiple providers. The IBM complete hybrid cloud stack includes a public (IBM Cloud) cloud and systems for on-premises offerings (IBM LinuxONE/IBM Z and IBM Power) (Figure 10-4).

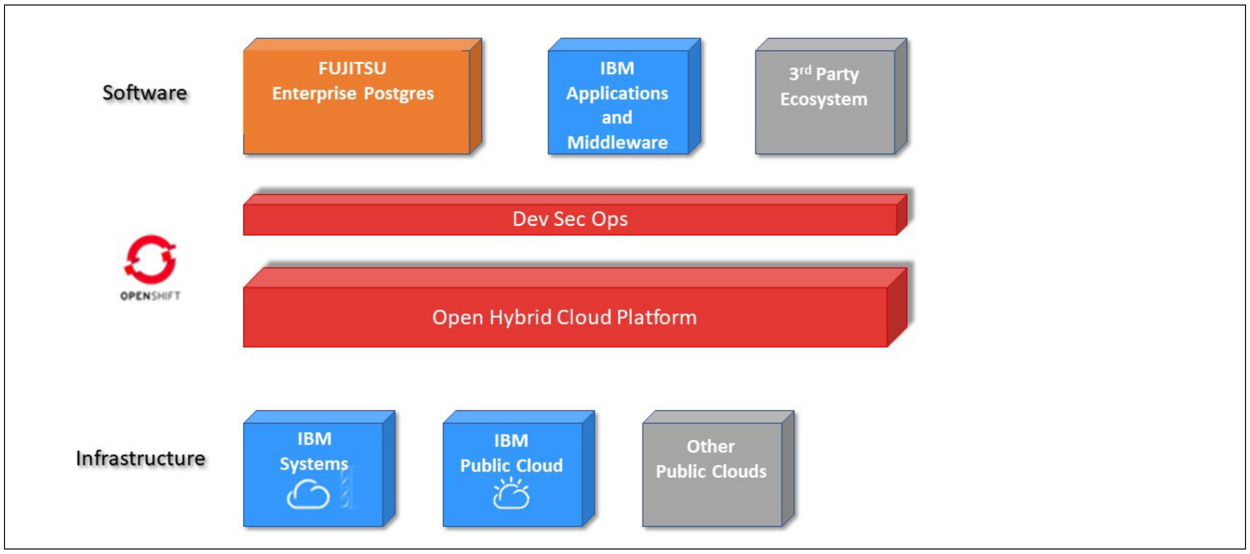


Figure 10-4 FUJITSU Enterprise Postgres on the Red Hat OpenShift hybrid multicloud platform

Red Hat OpenShift on IBM Cloud is an starter solution for adopting the Kubernetes offering as a simplified deployment and configuration of the Red Hat OpenShift Container Platform. As a managed service, IBM takes care of the ongoing maintenance, including OS patches, vulnerability remediation, and any updates in the Red Hat OpenShift stack. Workloads are moved to other worker nodes in the cluster during maintenance, which in combination with the HA application architecture of FUJITSU Enterprise Postgres ensures that there is no disruption to customers.

Red Hat OpenShift on IBM LinuxONE/IBM Z offers flexible computability with horizontal scalability on a single on-premises system that can grow to millions of containers, and vertical scalability that can non-disruptively grow databases without requiring any partitioning or sharding while maintaining response times and throughput.

In addition, IBM LinuxONE provides advanced security for confidential cloud computing, including FIPS 140-2 Level 4 certification and maximum isolation that meets Common Criteria Evaluation Assurance Level (EAL) 5+. IBM LinuxONE provides secure key technology with built-in tamper proof security, including tamper sensing and tamper responding to zeroize keys when attacked, and cryptographic cards and co-processors to perform encryption functions.

10.2.6 Red Hat OpenShift on IBM LinuxONE: System requirements

Red Hat OpenShift Container Platform 4.x is supported on IBM LinuxONE as several deployment options. The minimum resource-required deployment is installing Red Hat OpenShift 4.6 on a single partition (logical partition (LPAR)) with three Integrated Facility for Linux (IFL) cores and one Open System Adapter (OSA) network adapter. In a preferred deployment, you need three LPARs with six IFL processors and multiple OSA or RoCE network adapters. To further expand capacity, performance, and HA, you add worker nodes and storage options, and deploy across multiple systems.

A Red Hat OpenShift cluster is installed on IBM LinuxONE by following a user-provisioned infrastructure (UPI) scenario. General steps include the following actions:

- Partitioning IBM LinuxONE into one or multiple LPARs
- Allocating IFL cores, memory, and network adapters
- Preparing FICON or Fibre Channel Protocol (FCP)-attached disk storage
- Installing z/VM hypervisor and creating guest virtual machines (VMs) with RHCOS images
- Configuring and running the Red Hat OpenShift installer to ignite the instances of the cluster and perform the installation

Note: Red Hat OpenShift Container Platform has been available on IBM LinuxONE since Version 4.2. Version 4.6 is the latest version that is released at the time of writing.

10.2.7 Minimum system requirements

This scenario covers the installation of the smallest Red Hat OpenShift Container Platform cluster. For the recommended production or performance testing, see 10.2.8, “Preferred system requirements” on page 383.

The minimum system requirements are the following ones:

- IBM LinuxONE, or IBM z13/z13s and later
- One LPAR with z/VM Hypervisor 7.1 that uses three IFL processors in SMT2 mode, with 80 GB of memory
- FICON or FCP-attached disk storage
- A direct-attached OSA or RoCE network adapter, or virtual z/VM VSwitch networking

On z/VM, set up six guest VMs with the following components:

- One temporary bootstrap machine
- Three control plane (or master) machines
- At least two compute (or worker) machines

Here are the disk storage options for the z/VM guest VMs:

- FICON attached disk storage (direct-access storage devices (DASDs)). These DASDs can be z/VM minidisks, full-pack minidisks, or dedicated DASDs. To reach the minimum required DASD size for RHCOS installations, you need extended address volumes (EAVs). If available, use HyperPAV to ensure optimal performance.
- FCP-attached disk storage with logical units (LUNS) and SCSI devices.

- Shared file storage for Red Hat OpenShift Persistent Volume Claims (PVCs), which in this scenario is an NFS v4 server with more than 100 GB of disk storage.

Network configuration options include:

- A single vNIC for the z/VM VMs, direct-attached OSA or RoCE to each guest VM
- A single vNIC for the z/VM VMs, z/VM VSwitch with OSA (optionally, using link aggregation)

Each cluster machine must meet the minimum requirements that are shown in Table 10-2.

Table 10-2 Red Hat OpenShift minimum infrastructure requirements

Machine	OS	vCPU	Virtual RAM	Storage
Bootstrap	RHCOS		16 GB	120 GB
Control plane	RHCOS		16 GB	120 GB
Compute	RHCOS		8 GB	120 GB

Figure 10-5 shows the minimal resource deployment with NFS file storage for a Red Hat OpenShift dynamic storage provisioner.

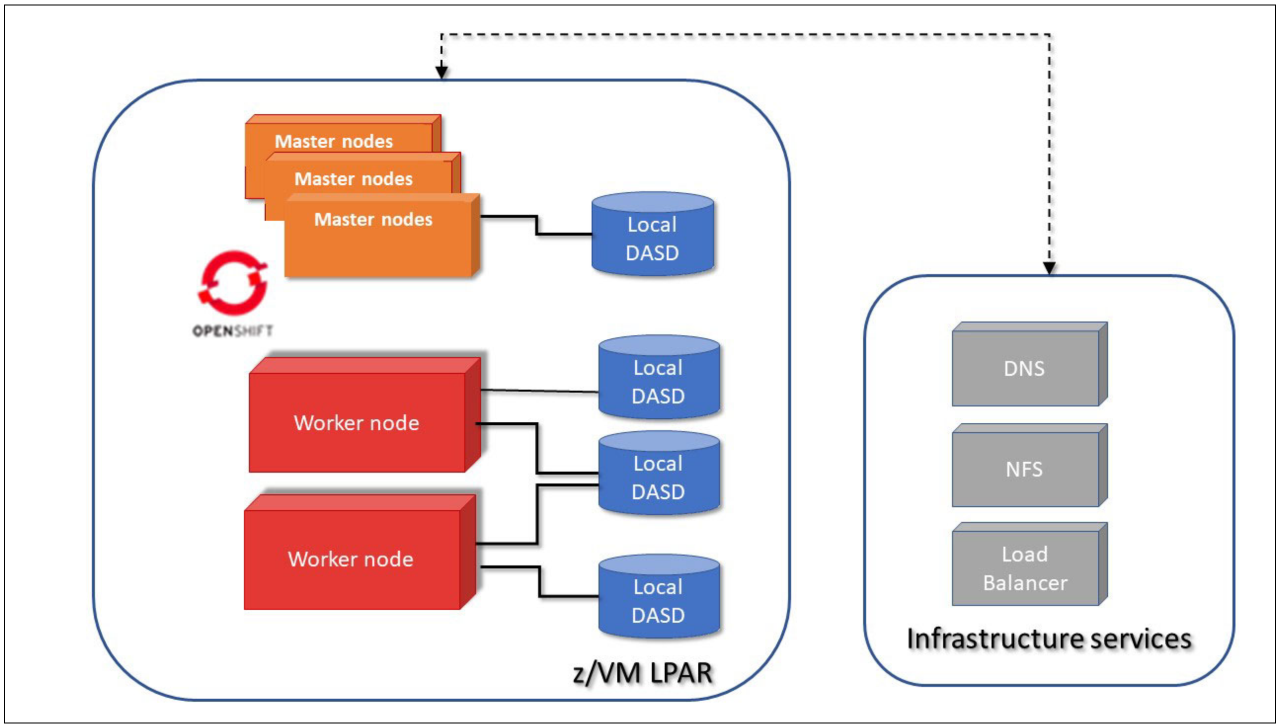


Figure 10-5 Red Hat OpenShift minimal deployment with NFS storage

10.2.8 Preferred system requirements

This scenario covers the installation of the recommended Red Hat OpenShift Container Platform cluster for production or performance testing. For experimentation or non-essential deployments, see 10.2.7, “Minimum system requirements” on page 382.

Here are the recommended system requirements:

- IBM LinuxONE, or IBM z13/z13s and later.
- Three LPARs with z/VM Hypervisor 7.1 that use six IFL processors, each in SMT2 mode (total 18 IFL processors) with 80 GB of memory.
- FICON or FCP-attached disk storage.
- One or two direct-attached OSA or RoCE network adapters, or HiperSockets, which are attached to a node either directly as a device or by bridging with one z/VM VSWITCH to be transparent to the z/VM guest. To directly connect HiperSockets to a node, you must set up a gateway to the external network through a Red Hat Enterprise Linux 8 guest to bridge to the HiperSockets network.

Set up two or three instances of z/VM for HA. On z/VM instances, set up six guest VMs for the following purposes:

- One temporary bootstrap machine
- Three control plane (or master) machines, one per z/VM instance
- At least six compute (or worker) machines that are distributed across the z/VM instances

Disk storage options for the z/VM guest VMs:

- FICON attached disk storage (DASDs). These DASDs can be z/VM minidisks, full-pack minidisks, or dedicated DASDs. To reach the minimum required DASD size for RHCOS installations, you need EAV. If available, use HyperPAV and High-Performance FICON (zHPF) to ensure optimal performance.
- FCP-attached disk storage with LUNs and SCSI devices.
- Shared block storage for Red Hat OpenShift PVCs uses an IBM Spectrum Virtualize storage back end with more than 100 GB of disk storage.

Network configuration options include the following ones:

- A single vNIC for the z/VM VMs, direct-attached OSA or RoCE, to each guest VM (multiple adapters can be bonded for higher throughput).
- A single vNIC for the z/VM VMs, with a z/VM VSwitch with OSA (using link aggregation).

Each cluster machine must meet at least the requirements that are shown in Table 10-3.

Table 10-3 Red Hat OpenShift recommended infrastructure requirements

Machine	OS	vCPU	Virtual RAM	Storage
Bootstrap	RHCOS		16 GB	120 GB
Control plane	RHCOS		16 GB	120 GB
Compute	RHCOS		8 GB	120 GB

Figure 10-6 on page 385 depicts the preferred resources deployment with IBM Spectrum block storage for a Red Hat OpenShift dynamic storage provisioner.

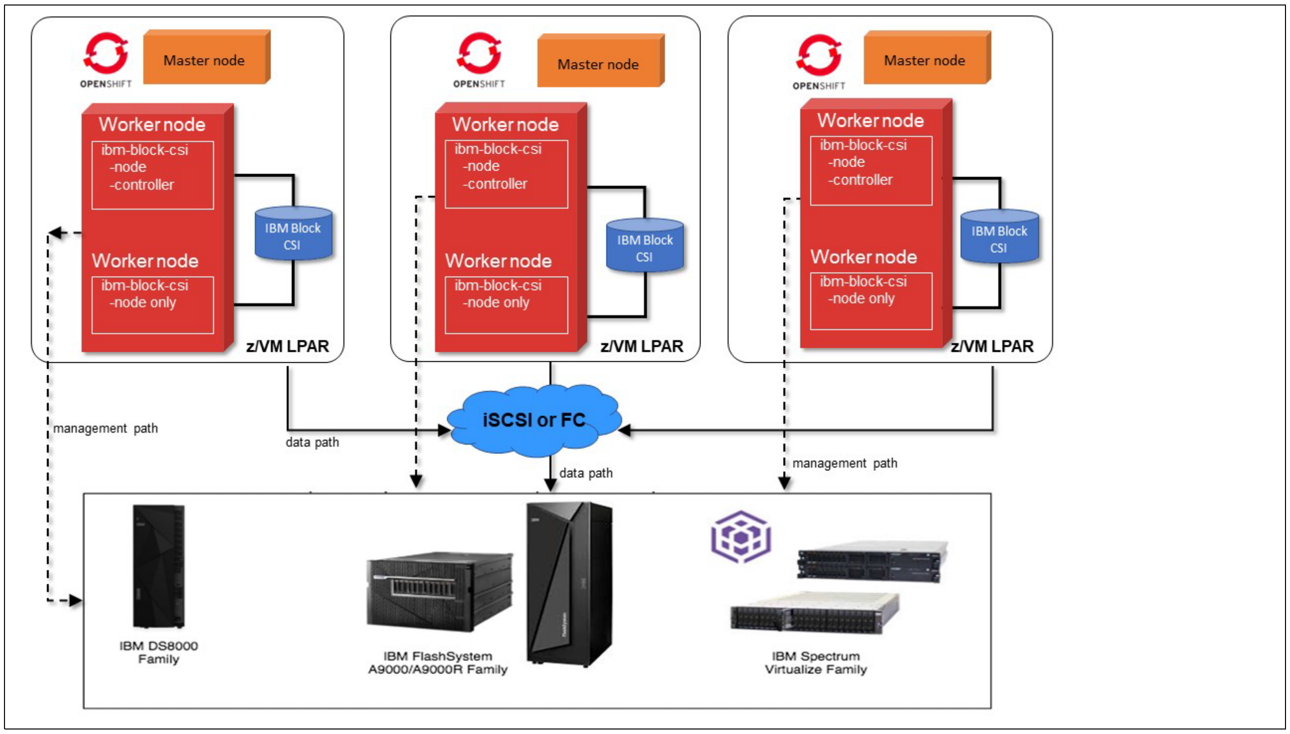


Figure 10-6 Red Hat OpenShift preferred deployment with integration of IBM block storage systems and Container Storage Interface driver

10.2.9 Red Hat OpenShift Deployment procedure on IBM LinuxONE

This section describes the installation sequence and configuration procedure flow for RHOCP on IBM LinuxONE. The requirements for either a minimum or preferred resources deployment must be met. The installation procedure takes about 90 minutes to complete.

Installation begins with setting up the UPI, including the helper (bastion) server that runs the DNS service, load-balancing, and the local HTTP Server that hosts the CoreOS files. The full list of tested integrations is available online at OpenShift Container Platform 4.x Tested Integrations.

Creating a user-provisioned infrastructure

To create the underlying infrastructure, complete the following steps.

1. Partition the machine into the required number of LPARs. Allocate IFL cores, memory, and network adapters. Prepare FICON or FCP-attached disk storage.
2. Install z/VM instances and provision guest VMs:
   a. Set up networking and static IP addresses.
   b. Set up an FTP server or use VMUR z/VM spool file queues to punch RHCOS bootable files and boot (IPL) the guest VMs.
3. Set up the helper/bastion node:
   a. Set up the DNS service for the cluster nodes (`named`).
   b. Set up the load balancer for the cluster rules (`HAproxy`).
   c. Set up the http repository for the ignition files (`HTTPd`).
   d. Configure the ports to be accessible in the local firewall.

The network connectivity between machines must be configured to allow cluster components to communicate. Each machine must be able to resolve the hostnames of all other machines in the cluster.

All machines require the ports that are shown in Table 10-4 to be open.

Table 10-4 Allowed ports from all machines to all machines

Protocol	Port	Description
ICMP	N/A	Network reachability tests
TCP	9000 - 9999	Host level services, including the node exporter on ports 9100 - 9101 and the Cluster Version Operator on port 9099
	10250 - 10259	The default ports that Kubernetes reserves
	10256	`openshift-sdn`
UDP	4789	VXLAN and Geneve
	6081	VXLAN and Geneve
	9000 - 9999	Host level services, including the node exporter on ports 9100 - 9101
TCP/UDP	30000 - 32767	Kubernetes NodePort

All machines to the control panel require that the ports shown in Table 10-5 to be open.

Table 10-5 Allowed ports from all machines to the control panel

Protocol	Port	Description
TCP	2379 - 2380	etcd server, peer, and metrics ports
	6443	Kubernetes API

Setting up the DNS server for use with a Red Hat OpenShift cluster

You must install and configure the DNS server. The domain name server that is available with the Red Hat Enterprise Linux Server distribution can be used to create the cluster DNS server. Complete the following steps:

1. Install the bind OS package by using the following command:

   ```
   yum install bind-chroot
   ```

2. Update the configuration file (`/etc/named.conf`) to implement the records from Table 10-6 on page 387, which lists the DNS records that are required for an Red Hat OpenShift Container Platform cluster that uses user-provisioned infrastructure (UPI).

Table 10-6 DNS records for Red Hat OpenShift Container Platform cluster.

Component	Record	Description
Kubernetes API	`api.<cluster_name>.<base_domain>.`	This DNS A/AAAA or CNAME record must point to the load balancer for the control plane machines. This record must be resolvable by both clients that are external to the cluster and from all the nodes within the cluster.
	`api-int.<cluster_name>.<base_domain>.`	This DNS A/AAAA or CNAME record must point to the load balancer for the control plane machines. This record must be resolvable from all the nodes within the cluster. Note: The apiserver must be able to resolve the worker nodes by the hostnames that are recorded in Kubernetes. If it cannot resolve the node names, proxied API calls can fail, and you cannot retrieve logs from pods.
Routes	`*.apps.<cluster_name>.<base_domain>.`	A wildcard DNS A/AAAA or CNAME record that points to the load balancer that targets the machines that run the ingress router pods, which are the worker nodes by default. This record must be resolvable by both clients that are external to the cluster and from all the nodes within the cluster.

3. Start or restart the server and enable the named service by using the following commands:

```
systemctl restart named-chroot
systemctl enable named-chroot
```

Setting up a web server for hosting the ignition files for Red Hat OpenShift

You must install and configure a web server (`HTTPd`) for hosting the ignition files. The web server is available in the Red Hat Enterprise Linux distribution and can be installed by completing the following steps:

1. Install the `HTTPd` OS package by using the following **`yum`** command:

```
yum install HTTPd
```

2. Update the configuration file `httpd.conf` to point to the location of the RHCOS images and ignition files that are generated and copied during the installation. Because ports 80 and 443 are already reserved for Red Hat OpenShift, `HTTPd` must be configured to listen on another port, for example, 8080.

3. Start or restart the server and enable the `HTTPd` service by using the following commands:

   ```
   systemctl restart HTTPd
   systemctl enable HTTPd
   ```

A required service in the UPI is a load balancer. The load balancer (`haproxy`) that is available with the Red Hat Enterprise Linux Servers distribution can be used to create the haproxy server. Complete the following steps:

1. Install the `haproxy` OS package by using the following command:

   ```
   yum install haproxy
   ```

2. Update the configuration files (`/etc/haproxy/haproxy.cfg`) to implement the rules from Table 10-7 and Table 10-8. For more examples, see “HAProxy” in Deploying a UPI environment for Red Hat OpenShift 4.1 on VMs and Bare Metal.
3. Start or restart the `haproxy` service by using the following commands:

   ```
   systemctl restart haproxy
   systemctl enable haproxy
   ```

   The `haproxy` rules that are shown in Table 10-7 must be configured for balancing the traffic between multiple master and worker nodes.

   The API load balancer requires the ports that are shown in Table 10-7 to be open.

Table 10-7 Internal and external ports for API load balancer

Port	Back-end machines (pool members)	Internal	External	Description
6443	Bootstrap and control plane. You remove the bootstrap machine from the load balancer after the bootstrap machine initializes the cluster control plane. You must configure the `/readyz` endpoint for the apiserver health check probe.	X	X	Kubernetes apiserver
22623	Bootstrap and control plane. You remove the bootstrap machine from the load balancer after the bootstrap machine initializes the cluster control plane.	X		Machine config server

The application, Ingress, load balancer requires the ports that are shown in Table 10-8 to be open.

Table 10-8 Internal and external ports for application Ingress load balancer

Port	Back-end machines (pool members)	Internal	External	Description
443	The machines that run the Ingress router pods, compute nodes, or worker nodes by default.	X	X	HTTPS traffic
80	The machines that run the Ingress router pods, compute nodes, or worker nodes by default.	X	X	HTTP traffic

Installation Red Hat images and ignition configurations

The underlying infrastructure is complete. Before moving on with the installation, the RHCOS images must be installed on each z/VM guest of the Red Hat OpenShift Cluster Platform. Download the latest RHCOS images whose minor release versions are less than or equal to the version of the Red Hat OpenShift installer that you plan to use. Create kernel parameter (PARAM) files for each node of the cluster with the URLs `coreos.inst.image_url` and `coreos.inst.ignition_url` pointing to the RHCOS images and ignition files on the `HTTPd` server.

10.2.10 Creating the Red Hat OpenShift cluster

To create the cluster, perform the following steps.

1. Download the Red Hat OpenShift installer.
2. Download the Red Hat OpenShift client.
3. Download the Red Hat OpenShift Cluster Platform 4 `pull_secret` file by using your account at https://cloud.redhat.com.
4. Create the working directory.
5. Generate `install-config.yaml` and prepare the ignition files.
6. Prepare the SSH keys.
7. Prepare the `kubeconfig` files.
8. Wait for the bootstrap and nodes to the installation are complete.

The version of the `openshift-installer` that matches the minor version of the RHCOS images (for example, 4.5) must be downloaded, extracted, and copied to the directory in the PATH of the root user on the client machine. Example 10-1 shows the commands that we used in our environment.

Example 10-1 Downloading, extracting, and copying openshift-installer

```
wget
https://mirror.openshift.com/pub/openshift-v4/s390x/clients/ocp/latest-4.5/openshi
ft-install-linux.tar.gz
tar zxvf openshift-install-linux.tar.gz
cp openshift-install /usr/local/sbin/
chmod 755 /usr/local/sbin/openshift-install
```

10.2.11 Downloading the Red Hat OpenShift client

The latest version of the `openshift-client` must be downloaded, extracted, and copied to the directory in the PATH of the root user on the client machine, as shown in Example 10-2.

Example 10-2 Downloading, extracting, and copying openshift-client

```
wget
https://mirror.openshift.com/pub/openshift-v4/s390x/clients/ocp/latest/openshift-c
lient-linux.tar.gz
tar zxvf openshift-client-linux.tar.gz
cp oc /usr/local/sbin/
chmod 755 /usr/local/sbin/oc
```

10.2.12 Preparing the ssh keys

Generate the ssh keys by using the following command. The key generation parameters may be changed as needed.

```
ssh-keygen -t rsa -b 2048 -N '' -C 'OCP-4-Admin' -f /root/.ssh/id_rsa
```

10.2.13 Creating the installation configuration file

Unlike in an installer-provisioned infrastructure, for installations of Red Hat OpenShift Container Platform that use a UPI, you must manually create your installation configuration file. To do this task, perform the following steps:

1. Create an installation directory to store your required installation assets in by using the following command:

   ```
   mkdir <installation_directory>
   ```

2. Customize `install-config.yaml` and save it in the `<installation_directory>`. Make a backup copy so that you can reinstall the cluster. A sample of the `install-config yaml` file is shown in Example 10-3.

Example 10-3 Sample install-config.yaml file for IBM Z

```
apiVersion: v1
baseDomain: example.com
compute:
 hyperthreading: Enabled
  name: worker
  replicas: 0
  architecture : s390x
controlPlane:
  hyperthreading: Enabled
  name: master
  replicas: 3
  architecture : s390x
metadata:
  name: test
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  networkType: OpenShiftSDN
  serviceNetwork: - 172.30.0.0/16
platform:
  none: {}
fips: false
pullSecret: '{"auths": ...}'
sshKey: 'ssh-ed25519 AAAA...'
```

10.2.14 Creating the Kubernetes manifest and ignition config files

Because you must modify some cluster definition files and manually start the cluster machines, you must generate the Kubernetes manifest and Ignition config files that the cluster needs to make its machines.

Note: As a best practice, perform a backup of the file before running the **`openshift-install`** command.

Complete the following steps:

1. Generate the Kubernetes manifests for the cluster by using the following command:

   ```
   openshift-install create manifests --dir=<installation_directory>
   ```

2. Modify the `<installation_directory>/manifests/cluster-scheduler-02-config.yml` Kubernetes manifest file to prevent pods from being scheduled on the control plane machines by completing the following steps:

   a. Open the `<installation_directory>/manifests/cluster-scheduler-02-config.yml` file and complete the following steps:

      i. Locate the **`mastersSchedulable`** parameter and set its value to `False`.

      ii. Save and exit the file.

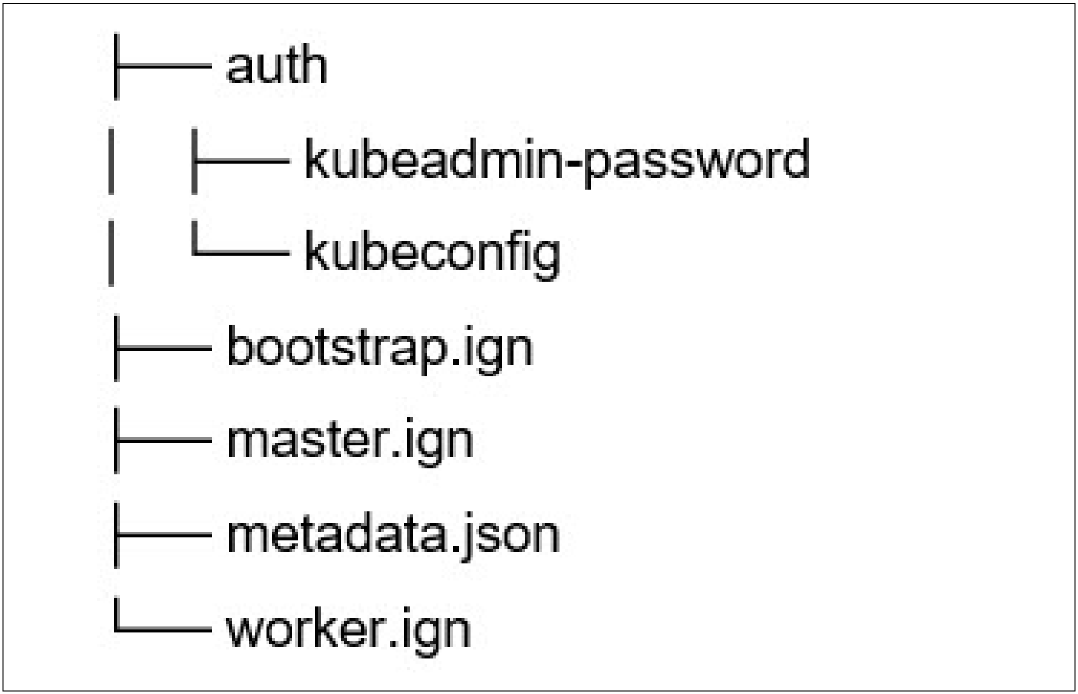
   b. Obtain the Ignition configuration files by completing the following steps:

      i. Run the following command:

      ```
      openshift-install create ignition-configs --dir=<installation_directory>
      ```

      i. For `<installation_directory>`, specify the same installation directory.

The files that are shown in Figure 10-7 are generated in the directory.



```
├── auth
│   ├── kubeadmin-password
│   └── kubeconfig
├── bootstrap.ign
├── master.ign
├── metadata.json
└── worker.ign
```

Figure 10-7 Ignition configuration files in the installation directory

10.2.15 Creating the cluster

To create the Red Hat OpenShift Container Platform cluster, wait for the bootstrap process to complete on the machines that you provisioned by using the ignition config files that you generated with the installation program.

Monitor the bootstrap process by using the following command:

```
openshift-install --dir=<installation_directory> wait-for bootstrap-complete
--log-level=debug
```

After the bootstrap process completes, remove the bootstrap machine from the load balancer.

10.2.16 Installing the NFS server and setting up a storage class in Red Hat OpenShift Container Platform

To dynamically provision persistent file volumes that are attached to the Red Hat OpenShift pods, you must have an NFS server on the network and an NFS dynamic provisioner in Red Hat OpenShift cluster. The provisioner allows volumes creation on demand and eliminates the need for cluster administrators to pre-provision storage. Stateful containers, such as FUJITSU Enterprise Postgres, require many volumes to be created. The dynamic provisioner and storage class that are present in the cluster create the volumes and bind them automatically to the pods during the FUJITSU Enterprise Postgres deployment.

For other dynamic file storage provisioners that are supported on the platform, see Introduction to IBM Spectrum Scale Container Storage Interface driver.

Complete the following steps:

1. Install the NFS server and create a `nfs` directory. Volumes are dynamically created in that location. The basic installation of an NFS server can be done on the bastion node. Run the following commands:

   ```
   sudo yum install -y nfs-utils rpcbind
   sudo mkdir /home/nfsshare -p
   ```

2. Configure the NFS server by specifying the domain and hostname permissions by running the following commands:

   ```
   sudo echo "Domain = ${cluster_domain}" >> /etc/idmapd.conf
   sudo echo "/home/nfsshare ${hostname}(rw,no_root_squash)" >> /etc/exports
   ```

3. Enable the NFS service and update the local firewall rules by using the following commands:

   ```
   sudo systemctl enable --now rpcbind nfs-server
   sudo firewall-cmd --add-service=nfs --permanent
   sudo firewall-cmd --add-service={nfs3,mountd,rpc-bind} --permanent
   sudo firewall-cmd --reload
   ```

The NFS server installation is complete. Proceed with deploying an NFS dynamic provisioner to the Red Hat OpenShift cluster by using Helm charts. The Helm chart is parameterized in the `yaml` file, and the following commands make a directory for the `yaml` file and create the file:

```
mkdir ~/nfs-helm-provisioner
vi ~/nfs-helm-provisioner/nfs_helm_values.yaml
```

The values that the `yaml` file specifies for a S390x image of the `nfs-client-provisioner`, `nfs` server IP address, and path are shown in Example 10-4.

Example 10-4 YAML file parameters

```
# This is a YAML-formatted file.
replicaCount: 1
strategyType: Recreate
image:
  repository: docker.io/ibmcom/nfs-client-provisioner-s390x
  tag: latest
  pullPolicy: IfNotPresent
nfs:
  server: ${nfs_ip}
```

```
  path: /home/nfsshare
  mountOptions:
# For creating the StorageClass automatically:
storageClass:
  create: true
  defaultClass: true
  name: nfs-client
  # Allow volume to be expanded dynamically
  allowVolumeExpansion: true
  # Method used to reclaim an obsoleted volume
  reclaimPolicy: Delete
  # When set to false your PVs will not be archived by the provisioner upon 
deletion of the PVC.
  archiveOnDelete: true
  # Set access mode - ReadWriteOnce, ReadOnlyMany or ReadWriteMany
  accessModes: ReadWriteOnce
## For RBAC support:
rbac:
  # Specifies whether RBAC resources should be created
  create: true
# If true, create and use Pod Security Policy resources
# https://kubernetes.io/docs/concepts/policy/pod-security-policy/
podSecurityPolicy:
  enabled: true
serviceAccount:
  # Specifies whether a ServiceAccount should be created
  create: true
  # The name of the ServiceAccount to use.
  # If not set and create is true, a name is generated using the fullname template
  name:
resources: {}
nodeSelector: {}
tolerations: []
affinity: {}
```

Run the **helm install** command while using a user-defined or a system namespace. Specify the `yaml` file with the following values:

```
helm install nfs-client-provisioner --values 
~/nfs-helm-provisioner/nfs_helm_values.yaml --namespace kube-system 
stable/nfs-client-provisioner
```

10.2.17 Installing IBM Spectrum Virtualize and setting up a storage class

Red Hat OpenShift supports dynamic provisioning of block storage persistent volumes. IBM provides a block storage CSI driver that enables Red Hat OpenShift to dynamically provision block storage that is used with stateful containers. Volumes creation on demand eliminates the need for cluster administrators to pre-provision storage in IBM storage systems. Stateful containers, such as FUJITSU Enterprise Postgres, require several volumes to be created. A dynamic provisioner and storage class that are present in the cluster create the volumes and bind them to the pods automatically during the FUJITSU Enterprise Postgres deployment.

The basic installation of the IBM block storage CSI driver begins with setting up the storage systems and access to them, whether over iSCSI or FCP storage area network (SAN) protocols. Supported IBM storage systems for Version 1.3 are:

- The IBM Spectrum Virtualize Family (including IBM Flash family members built with IBM Spectrum Virtualize (IBM FlashSystem® 5010, IBM FlashSystem 5030, IBM FlashSystem 5100, IBM FlashSystem 7200, IBM FlashSystem 9100, IBM FlashSystem 9200, and IBM FlashSystem 9200R) and IBM SAN Volume Controller (SVC) models SV2, SA2)
- IBM FlashSystem A9000/R
- IBM DS8880
- IBM DS8900

To install IBM Spectrum Virtualize, complete the following steps:

1. Download the IBM block storage operator manifest by using the following command:

   ```
   curl -L
   https://github.com/IBM/ibm-block-csi-operator/releases/download/v1.1.0/ibm-bloc
   k-csi-operator-z.yaml > ibm-block-csi-operator.yaml
   ```

2. Modify the version of the operator in the `yaml` file as necessary, for example, from `1.1.0` to `1.3.0`:

   ```
   sed -i "s/1.1.0/1.3.0/g" ibm-block-csi-operator.yaml
   ```

3. Install the operator while connected to a user-defined or a system namespace:

   ```
   oc apply -f ibm-block-csi-operator.yaml
   ```

4. Verify that the operator is running:

   ```
   oc get pod -l app.kubernetes.io/name=ibm-block-csi-operator
   ```

5. Install the IBM block storage CSI driver by creating an `IBMBlockCSI` custom resource (CR). Download the manifest of the CSI driver from GitHub:

   ```
   curl -L
   https://github.com/IBM/ibm-block-csi-operator/releases/download/v1.1.0/csi.ibm.
   com_v1.1_ibmblockcsi_cr_ocp.yaml > csi.ibm.com_v1_ibmblockcsi_cr.yaml
   ```

6. Modify the version of the operator in the yaml file as needed, for example, from `1.1.0` to `1.3.0`:

   ```
   sed -i "s/1.1.0/1.3.0/g" csi.ibm.com_v1_ibmblockcsi_cr.yaml
   ```

7. Install the CSI driver to your Red Hat OpenShift cluster:

   ```
   oc apply -f csi.ibm.com_v1_ibmblockcsi_cr.yaml
   ```

After the driver is installed and running, configure the array secrets and storage classes by completing the following steps.

1. Create an array secret:

   ```
   oc create secret generic <NAME> --from-literal=username=<USER>
   --from-literal=password=<PASSWORD>--from-literal=management_address=<ARRAY_MGMT
   >
   ```

2. Create a storage class `yaml` file that is called `storageclass-gold.yaml`, as shown in Example 10-5 on page 395.

Example 10-5 Storage class YAML file

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: gold
provisioner: block.csi.ibm.com
parameters:
  SpaceEfficiency: <VALUE>
  pool: <VALUE_POOL_NAME>
  csi.storage.k8s.io/provisioner-secret-name: <VALUE_ARRAY_SECRET>
  csi.storage.k8s.io/provisioner-secret-namespace: <VALUE_ARRAY_SECRET_NAMESPACE>
  csi.storage.k8s.io/controller-publish-secret-name: <VALUE_ARRAY_SECRET>
  csi.storage.k8s.io/controller-publish-secret-namespace:
<VALUE_ARRAY_SECRET_NAMESPACE>
  # csi.storage.k8s.io/fstype: <xfs>
  # volume_name_prefix: <prefix_name>
```

3. Apply the storage class by using the following command:

```
oc apply -f storageclass-gold.yaml
```

Now, create a Persistent Volume Claims (PVC) object. Example 10-6 provides a sample `yaml` file that specifies a PVC with 1 GB of storage with a raw block volume that is called `ibm-pvc-raw-block.yaml`.

Example 10-6 Sample YAML file

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: demo-pvc-raw-block
spec:
  volumeMode: Block
  accessModes:
 - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: gold
```

Apply the PVC configuration by using the following command:

```
oc apply -f ibm-pvc-raw-block.yaml
```

10.3 FUJITSU Enterprise Postgres on Red Hat OpenShift

To facilitate the installation and configuration of FUJITSU Enterprise Postgres on Red Hat OpenShift, two container images are available: *FUJITSU Enterprise Postgres Server* and *FUJITSU Enterprise Postgres Operator*.

The FUJITSU Enterprise Server container contains the FUJITSU version of Postgres software that runs the database engine. The FUJITSU Enterprise Operator container contains the logic to deploy and maintain a HA Postgres cluster. With these two container images, the users can set up a FUJITSU Enterprise Postgres cluster for data serving needs on Red Hat OpenShift Platform running on an IBM LinuxONE platform.

The FUJITSU Enterprise Postgres Operator allows you to take advantage of the unique features of FUJITSU Enterprise Postgres and operational management of database clusters, such as:

- Stand-up single node and multi-node cluster
- Operational management of a FUJITSU Enterprise Postgres server by using FUJITSU Enterprise Postgres Operator

10.3.1 Multi-architectural container images

Container technology is usually branded as cloud-neutral. The same container image can run on different cloud technology. Container images are still platform- or architecture-specific. An image that is built for the Intel/AMD platform (x86_64/amd64 architecture) cannot run on an ARM platform (arm/arm64 architecture) or IBM LinuxONE (s390x architecture) platform.

In 2016, a new specification, Image Manifest V2, Schema 2, also known as a *manifest list*, was introduced. The specification defines a manifest that combines multiple container image information, including architecture and O/S support, as an image digest into a single bundle. With this manifest, you can pull an image from a container registry by using one reference, regardless of the underlying platform/architecture. Using a busy box as an example, it is now possible to pull the image by referencing busybox1.0 on all platforms (architectures) that are supported by the bundle.

Both FUJITSU Enterprise Postgres Server and FUJITSU Enterprise Postgres Operator are built by following the Version 2 specification and providing multi-architectural container images. The container images can be deployed to a platform based on the amd64 architecture and a platform that is based on the S390x architecture. Hence, Red Hat OpenShift clusters running on an Intel/AMD system and on IBM LinuxONE can use the same deployment definitions for installing FUJITSU Enterprise Postgres.

10.3.2 FUJITSU Enterprise Postgres server

Setting up an HA Postgres cluster is a challenging task for the database administrator. It requires in-depth knowledge of how Postgres replication works and how to coordinate the failover process if the master node fails. The FUJITSU Enterprise Postgres server container image provides enterprise-grade Postgres server engine functions. You can set up an HA cluster with proper configuration on a software level by using automated installation. The Postgres server uses a leader election process to ensure only one pod in the cluster runs as the master (see Figure 10-8 on page 397). All other pods in the cluster run as replicas.

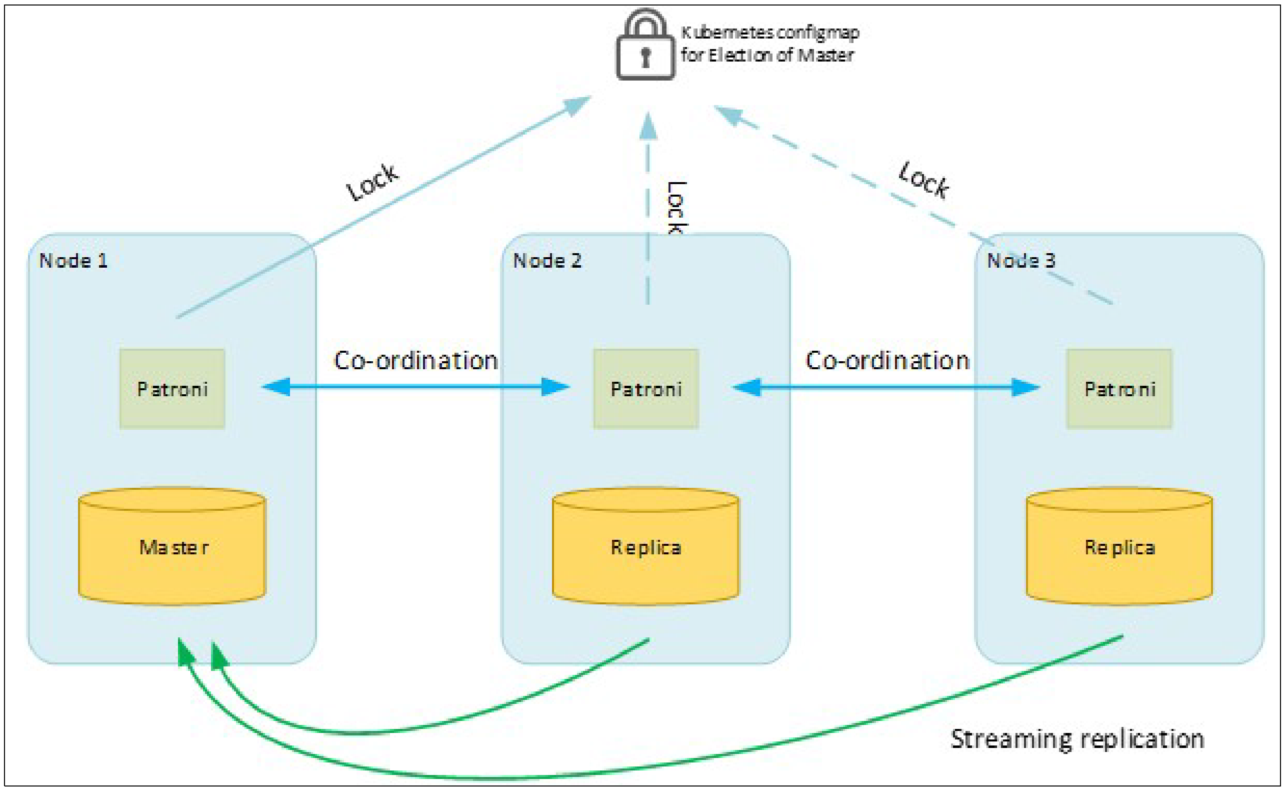


Figure 10-8 High availability FUJITSU Enterprise Postgres cluster with leader election

10.3.3 FUJITSU Enterprise Postgres Operator

The FUJITSU Enterprise Postgres Operator automates the deployment and management of FUJITSU Enterprise Postgres Server clusters on Red Hat OpenShift platform. By using FUJITSU Enterprise Postgres Operator, you can do the following tasks:

- Deploy a single pod Postgres cluster.
- Deploy a multiple pod HA Postgres cluster by using streaming replication, both synchronous and asynchronous.
- Configure all Postgres parameters.
- Install a certificate and enforce secure TLS communication between applications and the Postgres cluster.
- Enable (by default) FUJITSU Enterprise Postgres unique features:
  - Transparent Data Encryption (TDE) and create an encrypted table space.
  - Data Masking.
  - In-Memory Columnar Index (Vertical Clustered Index (VCI)).
  - Dedicated audit logs.
  - Global Meta Cache (GMC).
- Perform an orchestrated minor version upgrade with minimal downtime on the cluster.

10.4 FUJITSU Enterprise Postgres Operator operations

In this section, we describe an example installation and deployment of FUJITSU Enterprise Postgres Cluster by using FUJITSU Enterprise Operator V2.2.0. When working on other FUJITSU Enterprise Postgres Operator versions or a configuration that is specific to your needs, see FUJITSU Enterprise Postgres manuals.

10.4.1 FUJITSU Enterprise Postgres Operator installation

The FUJITSU Enterprise Postgres Operator is available in the Red Hat Ecosystem Catalog. The FUJITSU Enterprise Postgres Operator can be installed by using the Red Hat OpenShift Console (GUI) or command-line interface (CLI). For more information about the installation of the Red Hat OpenShift Console CLI command, see 10.2.11, “Downloading the Red Hat OpenShift client” on page 389.

Prerequisites

Here are the prerequisites to install the FUJITSU Enterprise Postgres Operator:

- Red Hat OpenShift V4.5+.
- Minimum of three worker nodes are required for an HA deployment with synchronous replication.
- Four or more worker nodes are required for an HA deployment with synchronous replication and anti-affinity enabled.
- Cluster administrator privilege for the initial deployment.
- The `oc` CLI for the installation through a CLI.
- NFS as the default storage class, as described in 10.2.16, “Installing the NFS server and setting up a storage class in Red Hat OpenShift Container Platform” on page 392.

Note: To obtain a copy of the Red Hat OpenShift CLI tool, see Where can I download the Red Hat OpenShift command-line tool?

Ports

The Postgres server exposes the ports that are shown in Table 10-9 by default.

Table 10-9 Default TCP ports that are exposed by the container

Service	Port
Postgres	27500
Patroni API	25001

Preparation

To prepare for the installation, complete the following steps:

1. Log in to Red Hat OpenShift by using an account with Cluster Admin privileges.
2. Go to **Projects,** as shown in Figure 10-9 on page 399.
3. Click **Create Project** to create a project that is called znprj, as shown in Figure 10-9 on page 399.

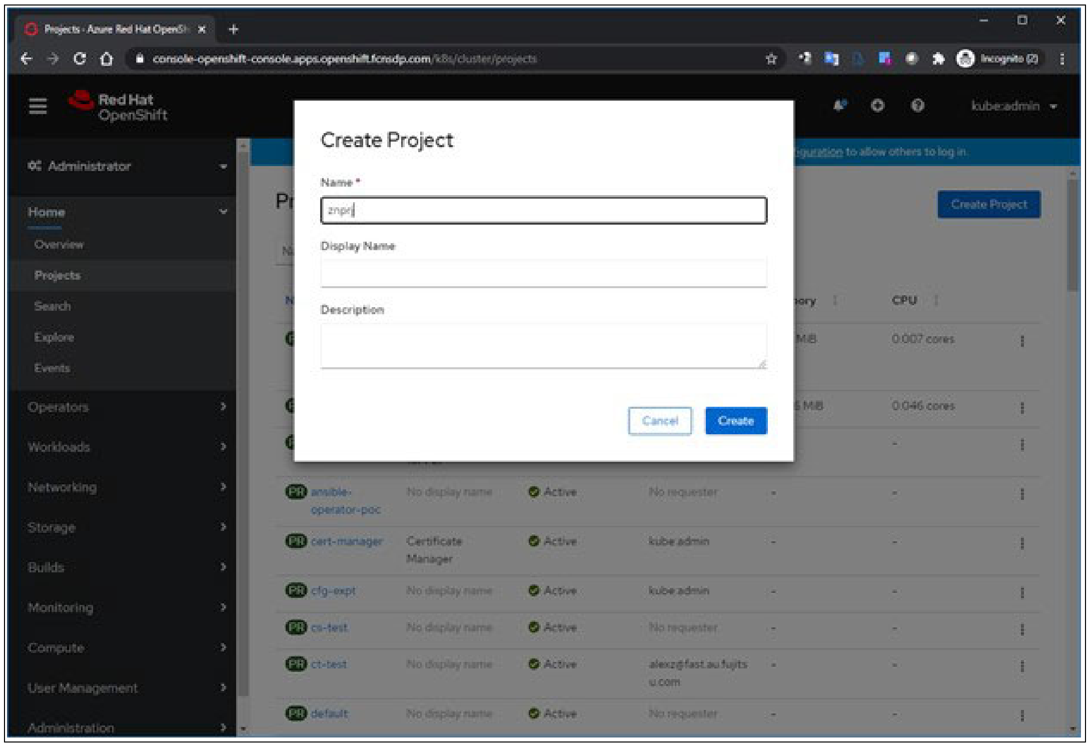


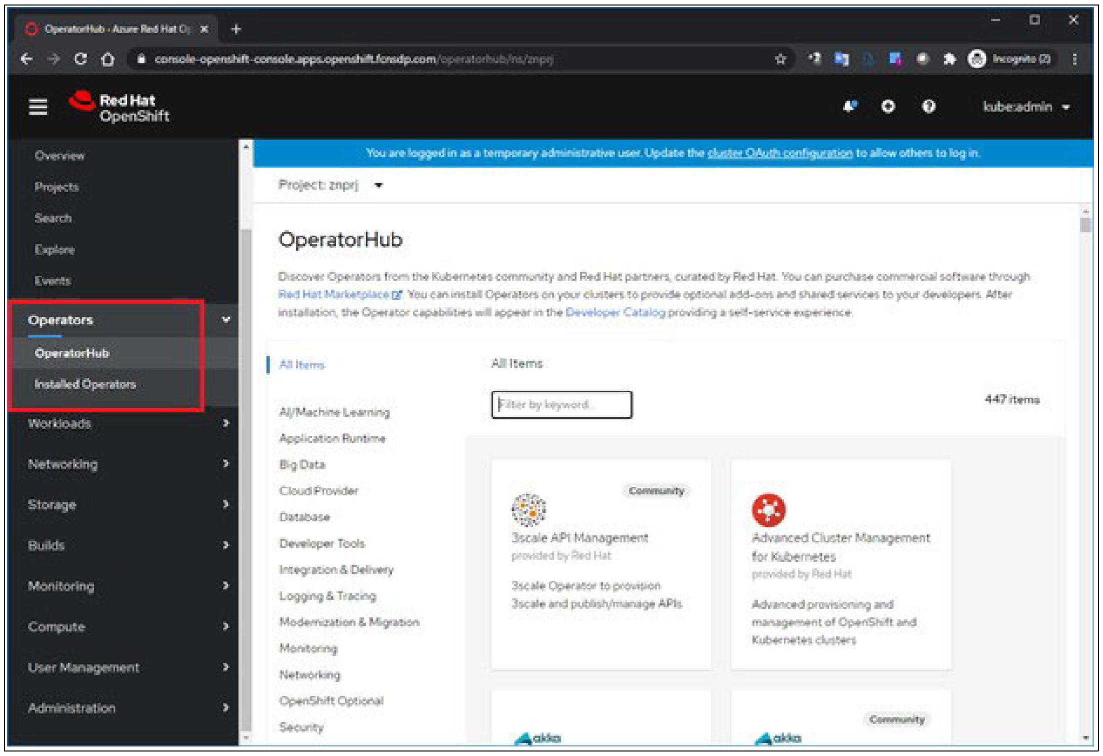
Figure 10-9 Creating a project

Note: For illustration purposes, all the actions are performed under the znprj project.

Operator installation and Postgres cluster deployment

Complete the following steps:

1. Select **Operator** → **Operator Hub**, as shown in Figure 10-10.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

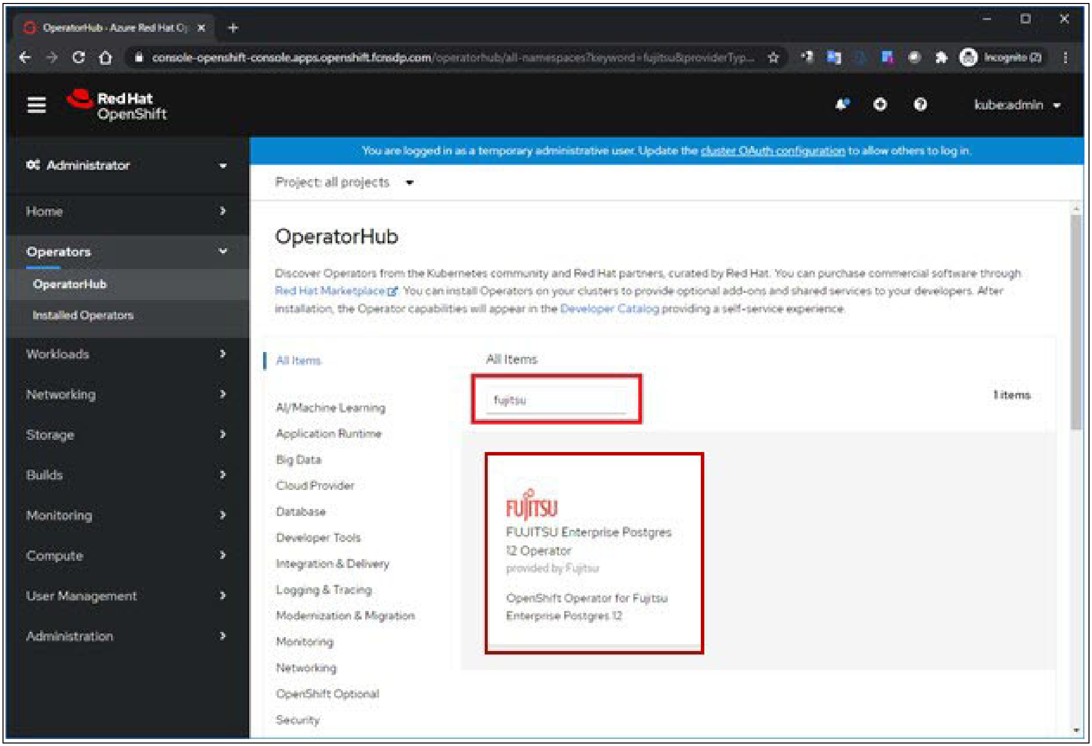
Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through Red Hat Marketplace. You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the Developer Catalog providing a self-service experience.

3scale Operator to provision 3scale and publish/manage APIs

Advanced provisioning and management of OpenShift and Kubernetes clusters

Figure 10-10 Red Hat OpenShift OperatorHub

2. In the search box, enter `fujitsu`. When the result comes back, click the **FUJITSU Enterprise Postgres 12 Operator** icon, as shown in Figure 10-11.



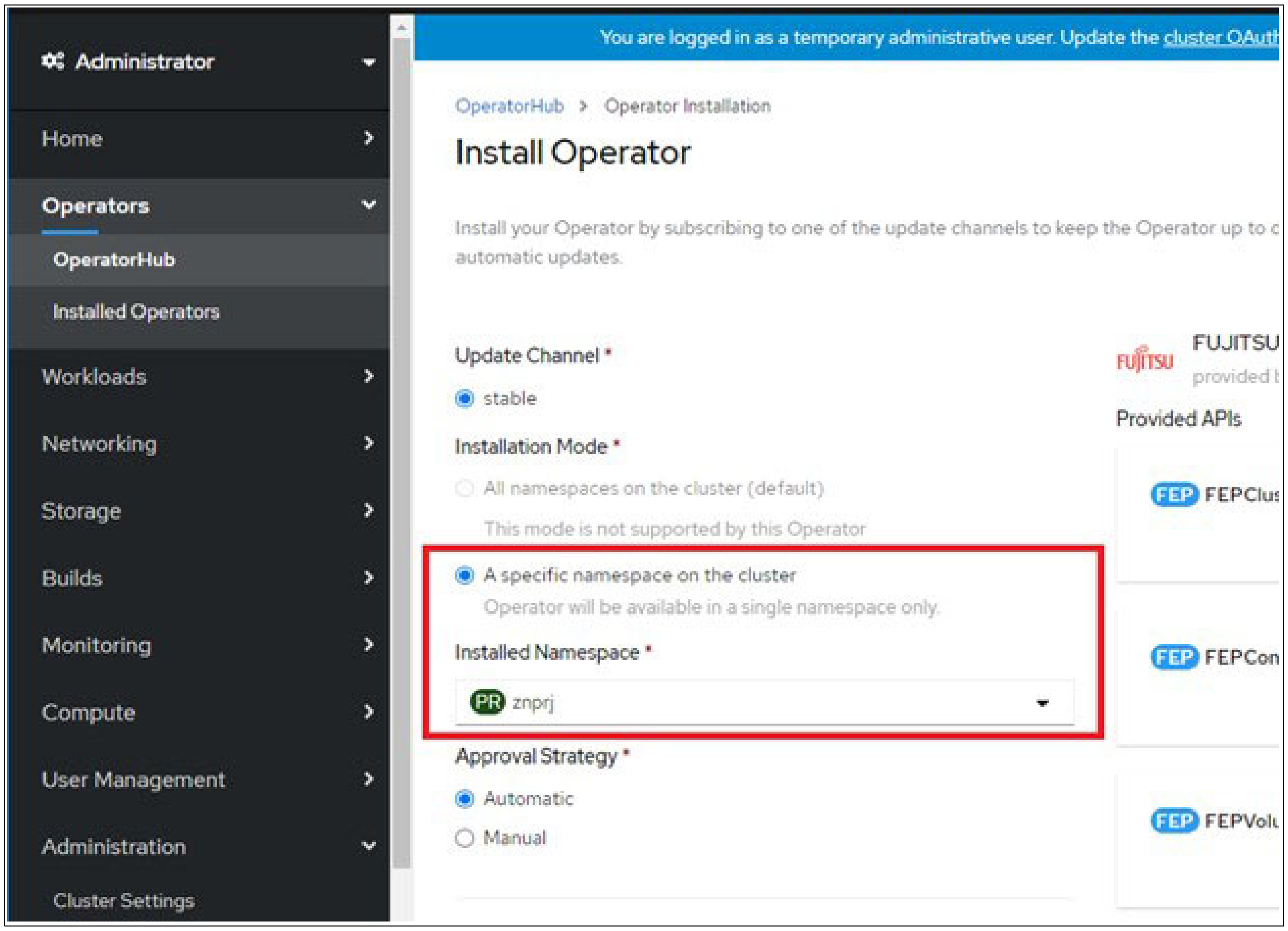
You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

OperatorHub

Discover Operators from the Kubernetes community and Red Hat partners, curated by Red Hat. You can purchase commercial software through Red Hat Marketplace. You can install Operators on your clusters to provide optional add-ons and shared services to your developers. After installation, the Operator capabilities will appear in the Developer Catalog providing a self-service experience.

Figure 10-11 Red Hat OpenShift OperatorHub: Search

3. On the Install Operator window, select **Update Channel** and **Installation Mode**. We selected **stable** and **A specific namespace on the cluster**, and selected **znprj** from the drop-down menu (Figure 10-12).



You are logged in as a temporary administrative user. Update the cluster OAuth

Install Operator

Install your Operator by subscribing to one of the update channels to keep the Operator up to d
automatic updates.

Update Channel *
● stable

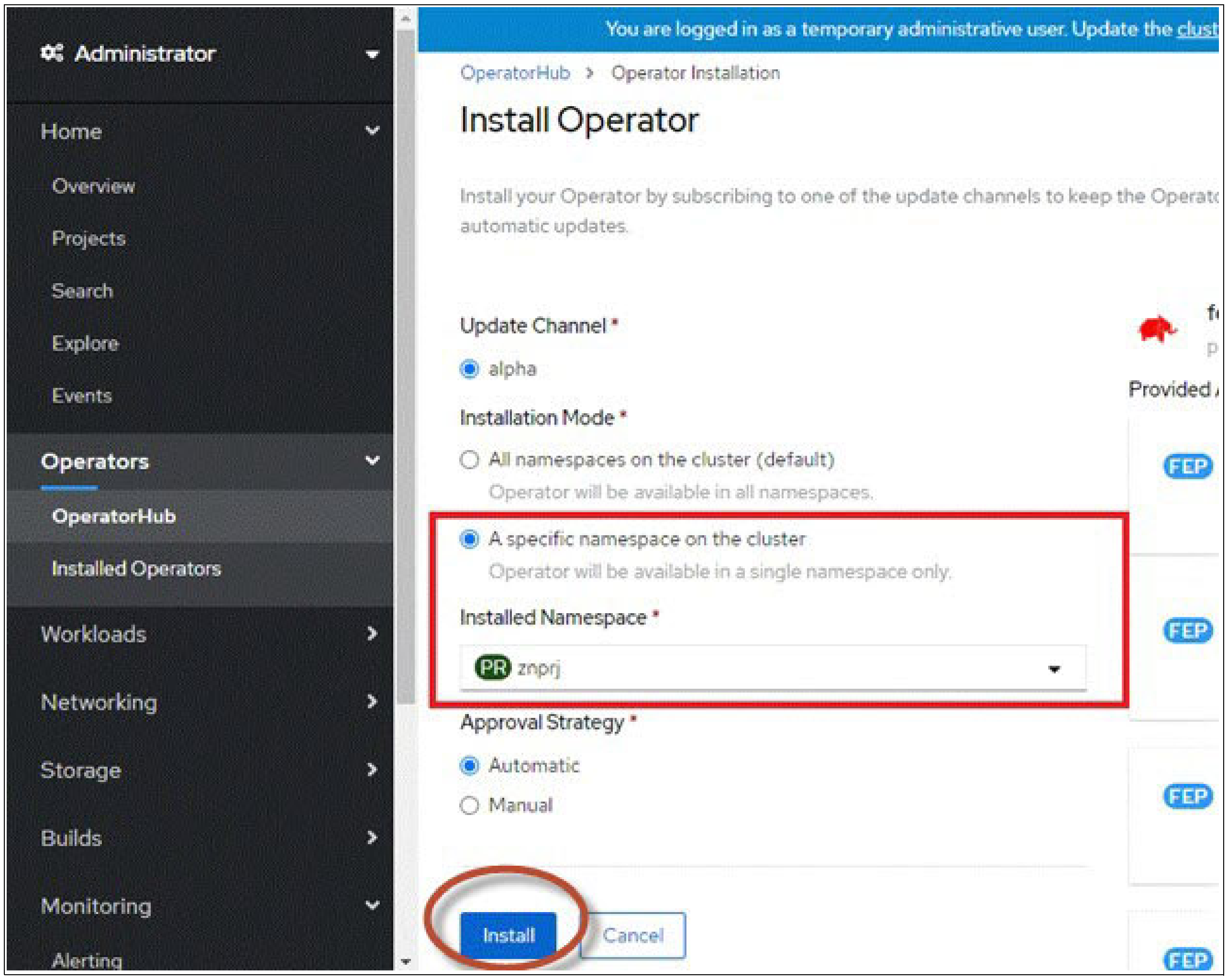
Installation Mode *
○ All namespaces on the cluster (default)
This mode is not supported by this Operator
● A specific namespace on the cluster
Operator will be available in a single namespace only.

Installed Namespace *
PR znprj

Approval Strategy *
● Automatic
○ Manual

Figure 10-12 Red Hat OpenShift OperatorHub: FUJITSU Enterprise Postgres 12 Operator Install menu

4. Click **Install** (Figure 10-13 on page 403).


You are logged in as a temporary administrative user. Update the clust

OperatorHub > Operator Installation

Install Operator

Install your Operator by subscribing to one of the update channels to keep the Operato automatic updates.

Update Channel *

(●) alpha

Installation Mode *

( ) All namespaces on the cluster (default)

Operator will be available in all namespaces.

(●) A specific namespace on the cluster

Operator will be available in a single namespace only.

Installed Namespace *

PR znprj

Approval Strategy *

(●) Automatic

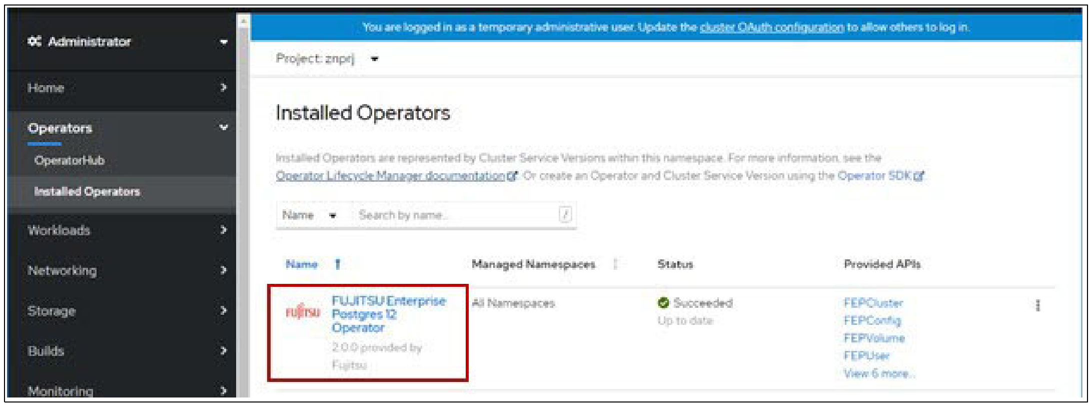
( ) Manual

Figure 10-13 Operator installation

Note: The Red Hat OpenShift Operator can be installed to all namespaces or to a specific namespace. When the Red Hat OpenShift Operator is installed to all namespaces, it is available to every namespace in the Red Hat OpenShift Cluster, including namespaces/projects that are created after the Red Hat OpenShift Operator is installed. When an updated Red Hat OpenShift Operator is available, it will be updated globally on the whole Red Hat OpenShift cluster.

When the Red Hat OpenShift Operator is installed to a specific namespace, it is restricted to that namespace/project. Each namespace/project can have a different version of Red Hat OpenShift Operator. Depending on the Approval Strategy for the Red Hat OpenShift Operator in a specific namespace, the Red Hat OpenShift Operator version in one namespace/project can be different from another namespace/project.

5. The FUJITSU Enterprise Postgres 12 Operator appears under Installed Operators after the installation completes (Figure 10-14).



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Project: znprj

Installed Operators

Installed Operators are represented by Cluster Service Versions within this namespace. For more information, see the Operator Lifecycle Manager documentation. Or create an Operator and Cluster Service Version using the Operator SDK.

Name	Managed Namespaces	Status	Provided APIs
FUJITSU Enterprise Postgres 12 Operator 2.0.0 provided by Fujitsu	All Namespaces	Succeeded Up to date	FEPCluster FEPConfig FEPVolume FEPUser View 6 more...

Figure 10-14 Operator installed

10.4.2 Deploying a single-node FUJITSU Enterprise Postgres cluster by using the command-line interface

FUJITSU Enterprise Postgres Operator leverages the Kubernetes Custom Resource Definitions (CRD) to deploy the FUJITSU Enterprise Postgres server cluster. Using the Custom Resource (CR), the user can specify the characteristics of FUJITSU Enterprise Postgres cluster to be deployed. The FUJITSU Enterprise Postgres 12 Operator uses the CR configuration file to define the required Kubernetes objects, such as config maps, secrets, storage volumes, services, and stateful sets (HA) to deploy a working FUJITSU Enterprise Postgres cluster within a few minutes.

Different aspects of cluster can be managed by updating corresponding CRs as needed, such as changing the parameters of `postgresql.conf`, that is, **`pg_hba.conf`** or **`pg_audit.conf`**. The changes in CRs, if allowed, are reflected in the required Kubernetes resources and passed to running FUJITSU Enterprise Postgres containers in a cluster to take effect.

Note: For more information about the various parameters that can be specified by using a CR configuration file to deploy the cluster and how to change the deployed parameter as required, see the FUJITSU Enterprise Postgres 12 Operation Guide.

The following examples show how to deploy a FUJITSU Enterprise Postgres cluster with a single pod (stand-alone mode) and multiple pods (HA) by using the FUJITSU Enterprise Postgres 12 Operator. The FUJITSU Enterprise Postgres 12 Operator can be used by using the **`oc`** CLI and the Red Hat OpenShift Console.

Stand-alone FUJITSU Enterprise Postgres server deployment by using a CLI

To create a stand-alone FUJITSU Enterprise Postgres server deployment by using a CLI, complete the following steps:

1. Create a CR configuration file that is used as deployment file to define a FUJITSU Enterprise Postgres cluster. In this example, a CR file that is called `standalone.yaml` is created, as shown in Example 10-7 on page 405.

Example 10-7 Stand-alone FUJITSU Enterprise Postgres Cluster deployment file

```
ngz@W10T-FAS200307:~/deploy$ cat standalone.yaml
apiVersion: fep.fujitsu.io/v2
kind: FEPCluster
metadata:
  name: standalone-fep
spec:
  fep:
    forceSsl: true
    image:
      pullPolicy: IfNotPresent
      image: "quay.io/fujitsu/fujitsu-enterprise-postgres-server:ubi8-12-1.0"
    mcSpec:
      limits:
        cpu: 500m
        memory: 700Mi
      requests:
        cpu: 200m
        memory: 512Mi
    podAntiAffinity: false
    podDisruptionBudget: false
    instances: 1
    servicePort: 27500
    syncMode: "off"
    sysExtraLogging: false
  fepChildCrVal:
    # The values that are specified below are used ONLY at start-up to create
    #child Custom Resources - FEPVolume, FEPConfig, FEPUser
    #customPgAudit: |
      # define pg audit custom params here to override defaults.
```

The CR configuration file includes a reference to the FUJITSU Enterprise Postgres server specification, that is, name, SSL termination, CPU, memory, FUJITSU Enterprise Postgres server Image, and other items. Table 10-10 describes some of the fields that can be modified by using the CR configuration file.

Table 10-10 CR configuration parameter details

Field	Value	Details

`metadata:`
`name:`
	`standalone-fep`	Name of the FUJITSU Enterprise Postgres cluster. Must be unique within a namespace.

`spec:`
`  fep:`
`    forceSsl:`
	`true`	The FUJITSU Enterprise Postgres cluster accepts only a TLS encrypted connection from a client.

`spec:`
`  fep:`
`    image:`
	`quay.io/fujitsu/xxx`	FUJITSU Enterprise Postgres server image.

Field	Value	Details

`spec:`
`  fep:`
`  mcSpec:`
	`limits:`
	`  cpu: 500m`
	`  memory: 700Mi`
	`requests:`
	`  cpu: 200m`
	`  memory: 512Mi`
		Resource allocation to this container.

`spec:`
`  fep:`
`    syncMode:`
	`off`	Replication mode.

`spec:`
`  fepChildCrVal:`
`    customPgAudit:`
	`[output]`
	`logger = 'auditlog'`
	`log_directory =`
	`'/database/log/audit'`
	`log_truncate_on_rotation =`
	`on`
	`log_filename =`
	`'pgaudit-%a.log'`
	`log_rotation_age = 1d`
	`log_rotation_size = 0`
	`[rule]`
		pgAudit configuration.

`spec:`
`  fep:`
`  instances:`
		Number of FUJITSU Enterprise Postgres pods in the cluster. For a stand-alone deployment, change it to 1.

`spec:`
`  fepChildCrVal:`
`  customPgParams:`
	`shared_preload_libraries='p`
	`gx_datamasking,pg_prewarm'`
	`session_preload_libraries='`
	`pg_prewarm'`
		Custom values to go into `postgresql.conf`. The two lines listed here enable the FUJITSU Enterprise Postgres unique features.

`spec:`
`  fepChildCrVal:`
`    sysUsers`
		Password for Postgres system users.

For this example, the CR configuration file, which is named `standalone.yaml`, is used to stand up a single FUJITSU Enterprise Postgres server with the configurable parameters that are defined in Table 10-10 on page 405. The CR configuration parameter details are shown in Table 10-10 on page 405 and highlighted in red in Example 10-8.

Example 10-8 CR configuration parameters that are used in our example

```
apiVersion: fep.fujitsu.io/v2
kind: FEPCluster
metadata:
  name: standalone-fep
  namespace: znprj
spec:
  fep:
    forceSsl: true
    image:
      pullPolicy: IfNotPresent
      image: "quay.io/fujitsu/fujitsu-enterprise-postgres-server:ubi8-12-1.0"
    mcSpec:
```

```
    limits:
      cpu: 500m
      memory: 700Mi
    requests:
      cpu: 200m
      memory: 512Mi
  podAntiAffinity: false
  podDisruptionBudget: false
  instances: 1
  servicePort: 27500
  syncMode: "off"
  sysExtraLogging: false
fepChildCrVal:
  # The values specified below are used ONLY at start-up to create child
  # Custom Resources - FEPVolume, FEPConfig, FEPUser
  customPgAudit: |
    # define pg audit custom params here to override defaults.
    [output]
    logger = 'auditlog'
    log_directory = '/database/log/audit'
    log_truncate_on_rotation = on
    log_filename = 'pgaudit-%a.log'
    log_rotation_age = 1d
    log_rotation_size = 0
    [rule]
  customPgHba: |
    # define pg_hba custom rules here to be merged with default rules.
    # TYPE DATABASE USER ADDRESS METHOD
  customPgParams: |
    # define custom postgresql.conf parameters below to override defaults.
    # Current values are as default FUJITSU Enterprise Postgres deployment
    shared_preload_libraries='pgx_datamasking,pg_prewarm'
    session_preload_libraries='pg_prewarm'
    max_prepared_transactions = 100
    max_worker_processes = 20
    max_connections = 100
    work_mem = 1MB
    maintenance_work_mem = 12MB
    shared_buffers = 128MB
    effective_cache_size = 384MB
    checkpoint_completion_target = 0.8
    # tcp parameters
    tcp_keepalives_idle = 30
    tcp_keepalives_interval = 10
    tcp_keepalives_count = 3
    # logging parameters in default fep installation
    # if log volume is not defined, log_directory should be
    # changed to '/database/userdata/data/log'
    log_filename = 'logfile-%a.log'
    log_file_mode = 0600
    log_truncate_on_rotation = on
    log_rotation_age = 1d
    log_rotation_size = 0
    log_checkpoints = on
    log_line_prefix = '%e %t [%p]: [%l-1] user=%u,db=%d,app=%a,client=%h'
    log_lock_waits = on
    log_autovacuum_min_duration = 60s
    logging_collector = on
    log_directory = '/database/userdata/data/log'
    pgaudit.config_file='/opt/app-root/src/pgaudit-cfg/pgaudit.conf'
    log_replication_commands = on
    log_min_messages = WARNING
    log_destination = stderr
    # wal_archive parameters in default fep installation
    archive_mode = on
    archive_command = '/bin/true'
    wal_level = replica
    max_wal_senders = 12
    wal_keep_segments = 64
  storage:
    dataVol:
      size: "5Gi"
    walVol:
      size: "1200Mi"
  sysUsers:
    pgAdminPassword: admin-password
    pgdb: mydb
    pgpassword: mydbpassword
    pguser: mydbuser
    pgrepluser: repluser
    pgreplpassword: repluserpwd
    tdepassphrase: tde-passphrase
  systemCertificates:
    key: |-
      -----BEGIN RSA PRIVATE KEY-----
      MIIEowIBAAKCAQEA4AI33yvHZws+jta6qpV6wzJqF8odIfTIpCfbrVcUUtLFKJ1I
      2e4SceTKi6O3C/I1XuvWlpng5IO65+fQQLO06z1/AuQT78YUn/Wlm9x1aHVsv4AN
      B5JWWqDOjrRT3o7nRPGXfilabPOrGE2mJJcVR9nExJ3IeaktgT3sb8YlXvtchyYp
      mjdbfxabTzO7igO+6/cwKoRRxOK8Uf7f5euEOcI/490J6r5Rs4lgD8sIQNCUFlTF
      YvmAH7gcdssSFBt8NPlUATHEsoFmlWODKCJWNhTLOht+s6L/1zwTHLjPG2pdkG6W
      dgmu5H2pDml8CDNLDv98Aj7i+I5SRKKcVPlnuQIDAQABAoIBAFPQYKlOzw/+BAOb
      yMIUpdctIMb/54CR/xROmVw1DbSjigNVPjHUQvB8Y1B2FAITQObgJO06bAvOQdWN
      RbO/v/yYiNJDFjaLjaIAHlO/2+oWrXbFaZqgpVDJhB+e1xaZr2x7XGxm+p925k30
      l6pvIRY+I8JRKvZiV1VZHwL/R3JOtPr++xMZtLVjVOI+f+ySqJ+TZHuAjm49EKxj
      cEmmJ28b7QcziXsvKyO0f+zbqLIBKXQdZAFU5eEr1BsDRXdRW+KfOXIvftuy4BJZ
      voKT+VGhEvF/qysswL4+6IAO6tpuYnnMOY2d3sOGoWPkTcQKOMekYKzL/WmtCjNs
      9hodJtECgYEA5EWyhEOf4uOKe5TDp697UCUvXLoOR58FDe/S8XNvScn29jjOkqIg
      OMoqo9xAkJTNTzqn5UUdt1x/pgM2NxlPLFijrcOzQlX3SoOO2ryDd9WNi7YKtN16
      KJqa536WeZu2OEbuAZ+S3GALVy1RPeTNPnUOmKnFO6DjDUGzLNCZy10CgYEA+zfw
      952DWuz1UOZ4wvAEqqcgUKXPKrkTXV/iUnjkDkrLYVrOZofDNTXrdHl+UedFmaOC
      cieZn6DNhcdz5tKtyysGMH3g/qs9PfoGUngvcXsyOEgkO4l3x1jc8TTCLqXZXYaQ
      HMsx51n+R58oncPtzYSUOr9qQ6PbC2CstTbFJAOCgYEAjGEsUliAB/jknfEzjXjG
      PdhQUxb8VyE864Az2lah9t/kJzFyIAziAeqZ5GE7t247AGFTBRTHHI8e1Qoemi3P
      Wbc9GVIbFs1lIYbcIDpUIyrKPEP8O5QEXtoNLxXTFgAjRGKiVY87spjCAJ+W2ZhO
      e/1it5GYXfgQCYQA2yuBmOUCgYANRkR2YR1axaCk+NlSu6oTdmdPu6M5x7PNQE7O
      OtMaKjua9lppvIzFGAdMDUtueoEEAE7ZR1xnwfB6PDLUpJdIYAqgr1YfPt8qkjaZ
      Tv56yZ7CwLOpbF8m6nwqRrZoDp1wwraEvvvxFKFKGY/k3kCHlpTakdjEoDjn3gDi
      RnWeVQKBgCEneMSzucei5LRppRtRaJw/Btll8qlPMlX3W7dxQ3cLwpmLOnOm51Fp
      PIZ44zYK8R6fu4+/sSrlfaIg86Ugeufp6YNxyNROKxUGza5vDIu5OftwWtBeg+UK
      Z8lLWNdX6pp7WMujmF3H1DrkBbauYMUKZ4UxUYtelgHERMePIxwb
      -----END RSA PRIVATE KEY-----
    crt: |-
      -----BEGIN CERTIFICATE-----
      MIIDUTCCAjmgAwIBAgIRAMocW3qMoHrD6qRvMPppMkMwDQYJKoZIhvcNAQELBQAw
      NzEQMA4GA1UECgwHRnVqaXRzdTEjMCEGA1UEAwwaRkVQIFJvb3QgQ0EgZm9yIEt1
      YmVybmV0ZXMwHhcNMjEwMjA2MDQzMjM2WhcNMjYwMjA1MDQzMjM2WjA/MRAwDgYD
      VQQKEwdGdWppdHN1MSswKQYDVQQDEyJGVUpJVFNVIEVudGVycHJpc2UgUG9zdGdy
      ZXMgU2VydmVyMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4AI33yvH
      Zws+jta6qpV6wzJqF8odIfTIpCfbrVcUUtLFKJ1I2e4SceTKi6O3C/I1XuvWlpng
      5IO65+fQQLOO6z1/AuQT78YUn/Wlm9x1aHVsv4ANB5JWWqDOjrRT3o7nRPGXfila
      bPOrGE2mJJcVR9nExJ3IeaktgT3sb8YlXvtchyYpmjdbfxabTz07ig0+6/cwKoRR
      xOK8Uf7f5euEOcI/49OJ6r5Rs4lgD8sIQNCUFlTFYvmAH7gcdssSFBt8NPlUATHE
      soFmlWODKCJWNhTLOht+s6L/1zwTHLjPG2pdkG6Wdgmu5H2pDml8CDNLDv98Aj7i
      +I5SRKKcVPlnuQIDAQABo1AwTjAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUH
      AwIwDAYDVR0TAQH/BAIwADAfBgNVHSMEGDAWgBQcwrrUOOu+FhIUuVdrDRCQRsi6
      ZjANBgkqhkiG9w0BAQsFAAOCAQEAm5dxBoI9pScOCvRAchg4CprdRDSJb9K6yB3O
      nCAxnM47iHeXnY3WlnI388kHu8DU7O4ba1tJbGs3KY9KzioPk43pU12jWkO1onoF
      +mTDjx/Ef1cYWA9r5q/LtgTa6Q2sxV4O2x67QW82aAnaxO34dV5zWCPIvAoovZBV
      HRT+BgCg3r2vD1RGKK2nl1aYJtWhO1SZubam+VttdZ/vbM9oOJctxmImsEtBXjkY
      KteePdQtLL5oO3JhyXWyRshCq+HMmKf2KgyY8gvydGcP4eLQdBWcW4OLcnVq6UjT
      OkJycJEKngMVademq1ZWHGaiYB7hyT6GhgIcHUJ2cKrPgbEh1Q==
      -----END CERTIFICATE-----
    cacrt: |-
      -----BEGIN CERTIFICATE-----
      MIIDTzCCAjegAwIBAgIUYssQ8I74US5g+1+Z7CHuaDgkZnEwDQYJKoZIhvcNAQEL
      BQAwNzEQMA4GA1UECgwHRnVqaXRzdTEjMCEGA1UEAwwaRkVQIFJvb3QgQ0EgZm9y
      IEt1YmVybmV0ZXMwHhcNMjEwMjA2MDM1MjI4WhcNMzEwMjA0MDM1MjI4WjA3MRAw
      DgYDVQQKDAdGdWppdHN1MSMwIQYDVQQDDBpGRVAgUm9vdCBDQSBmb3IgS3ViZXJu
      ZXRlczCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMs97gUF0xkUzCgL
      7MiiDju9ySr/ziwjvcYU7jA9ML+SLmftMs3HtcYbAmSntqI+MDBSR/FAJTOoytuT
      pV+mCFcGj2YAjDpliHPeNcUpbryy4YMChF3+MovkIwGCksxo5rhiWhGmoBYpA48P
      4Xe8SPlzqMzhFvNeKzyiUhvjutS2Y1Ss38lsTaurFPx64vQ2PaC54XzdwMptXtpb
      tYmWSzCpJWwxZ6lF3vitdA2w0tnBWNyctAd0+RIM/fvArxiIqseAux9tOuogm5to
      lRIhvekuxOpXBPEqtIYQ4j9XUW2JH8vUDnzPkPvjrq+A3Ug8OyyfGVrW7+VYXozu
      c4aP7POCAwEAAaNTMFEwHQYDVR0OBBYEFBzCutQ7S74WEhS5V2sNEJBGyLpmMB8G
      A1UdIwQYMBaAFBzCutQ7S74WEhS5V2sNEJBGyLpmMA8GA1UdEwEB/wQFMAMBAf8w
      DQYJKoZIhvcNAQELBQADggEBAMDwD85RAaWEBptFgLzKw+9xEUy1vcZaonAuA1qc
      T342XTueyAugxkC11HwdCGgGS34VyctfMGqj4AW6pA2ez4tLrbOps4DmV4sw8uBL
      8pgRDgfly3ob9FEg2waOhmrwX9jH5Bt4vySUE2785uPAqaspT2UNtTBxS85BUi1T
      sKId2Rtil6an281Z81wyWVI6Jm2D4MGOmbsiGcTPlCtdg/UljvDYymXlAvd4vNhl
      k9hDa13TgDqJKgKdTIcmZoNQdpEVgFcOOh9AEUy5AuLqxHq6OdLfZ6ESGPlMI7Lm
      i4PzYbCnBmOe+7TnHcPSyrnehs66Ik+oifRd82eYS7vKjFw=
      -----END CERTIFICATE-----
```

2. Apply the CR configuration file `standalone.yaml` by using the command that is shown in Example 10-9.

Example 10-9 Deploying the stand-alone cluster

```
ngz@W10T-FAS200307:~/deploy$ oc get fepcluster standalone-fep
NAME             AGE
standalone-fep   4m1s
ngz@W10T-FAS200307:~/deploy$ oc get pod -l app=standalone-fep-sts -L feprole
NAME                   READY   STATUS    RESTARTS   AGE     FEPROLE
standalone-fep-sts-0   1/1     Running   0          4m37s   master
```

3. The single node cluster is deployed, and the deployment status can be checked by using the command that is shown in Example 10-10 (`standalone-fep-sts-0` is the master stateful set for the cluster name `standalone-fep`, which is specified in the CR configuration file). The status shows `Running` after the cluster is ready.

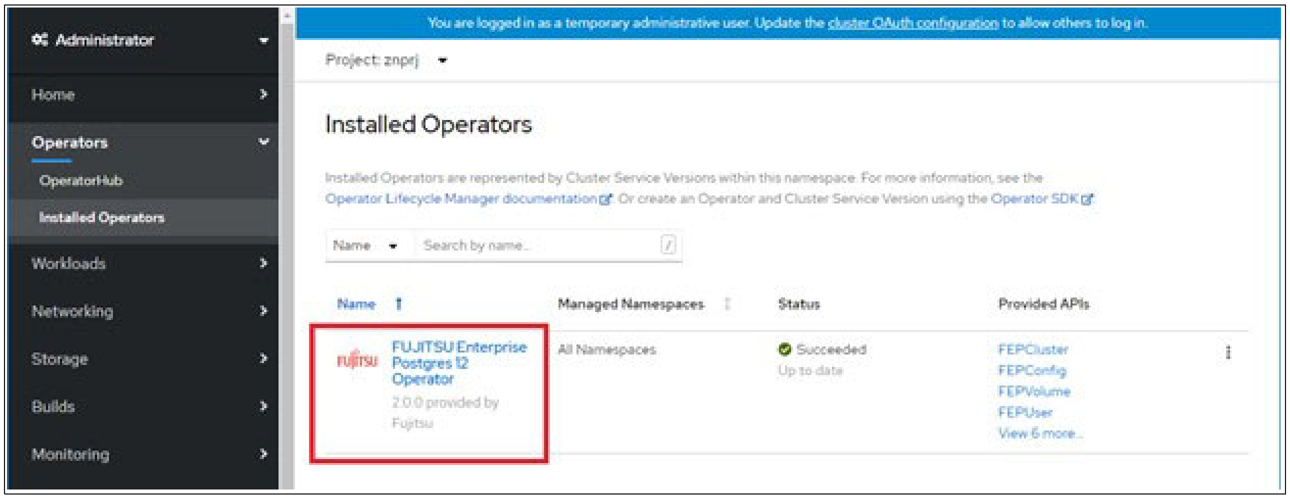
Example 10-10 Checking the stand-alone cluster deployment status

```
ngz@W10T-FAS200307:~/deploy$ oc get fepcluster standalone-fep
NAME                              AGE
STANDALONE-FEP                    4m1s

ngz@W10T-FAS200307:~/deploy$ oc get pod -1 app-standalone-fep-sts -L feprole
NAME                   READY   STATUS    RESTARTS   AGE    FEPROLE
standalone-fep-sts-0   1/1     Running   0          2,
ngz@W10T-FAS200307:~/deploy$
```

10.4.3 Deploying a single node FUJITSU Enterprise Postgres cluster by using the Red Hat OpenShift console

1. In the Red Hat OpenShift Console, select **Operators** → **Installed Operators**.
2. Select the FUJITSU Enterprise Postgres 12 Operator, as shown in Figure 10-15.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Installed Operators

Installed Operators are represented by Cluster Service Versions within this namespace. For more information, see the Operator Lifecycle Manager documentation. Or create an Operator and Cluster Service Version using the Operator SDK.

Name	Managed Namespaces	Status	Provided APIs
FUJITSU Enterprise Postgres 12 Operator 2.0.0 provided by Fujitsu	All Namespaces	Succeeded Up to date	FEPCluster FEPConfig FEPVolume FEPUser View 6 more...

Figure 10-15 Installed Operators

3. Select **Create Instance**, as shown in Figure 10-16.

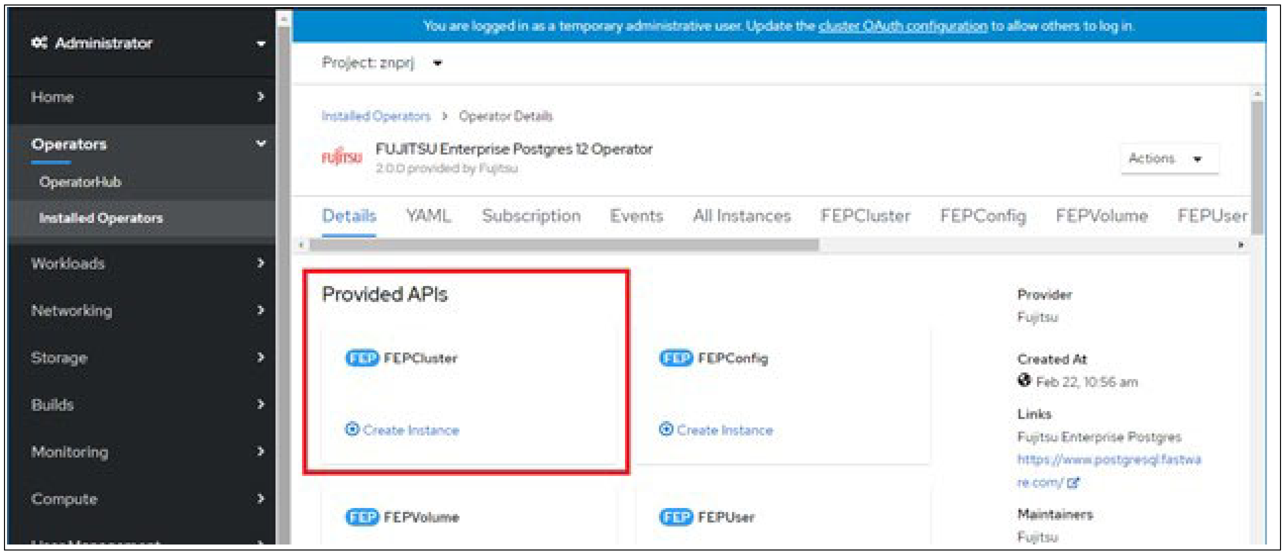
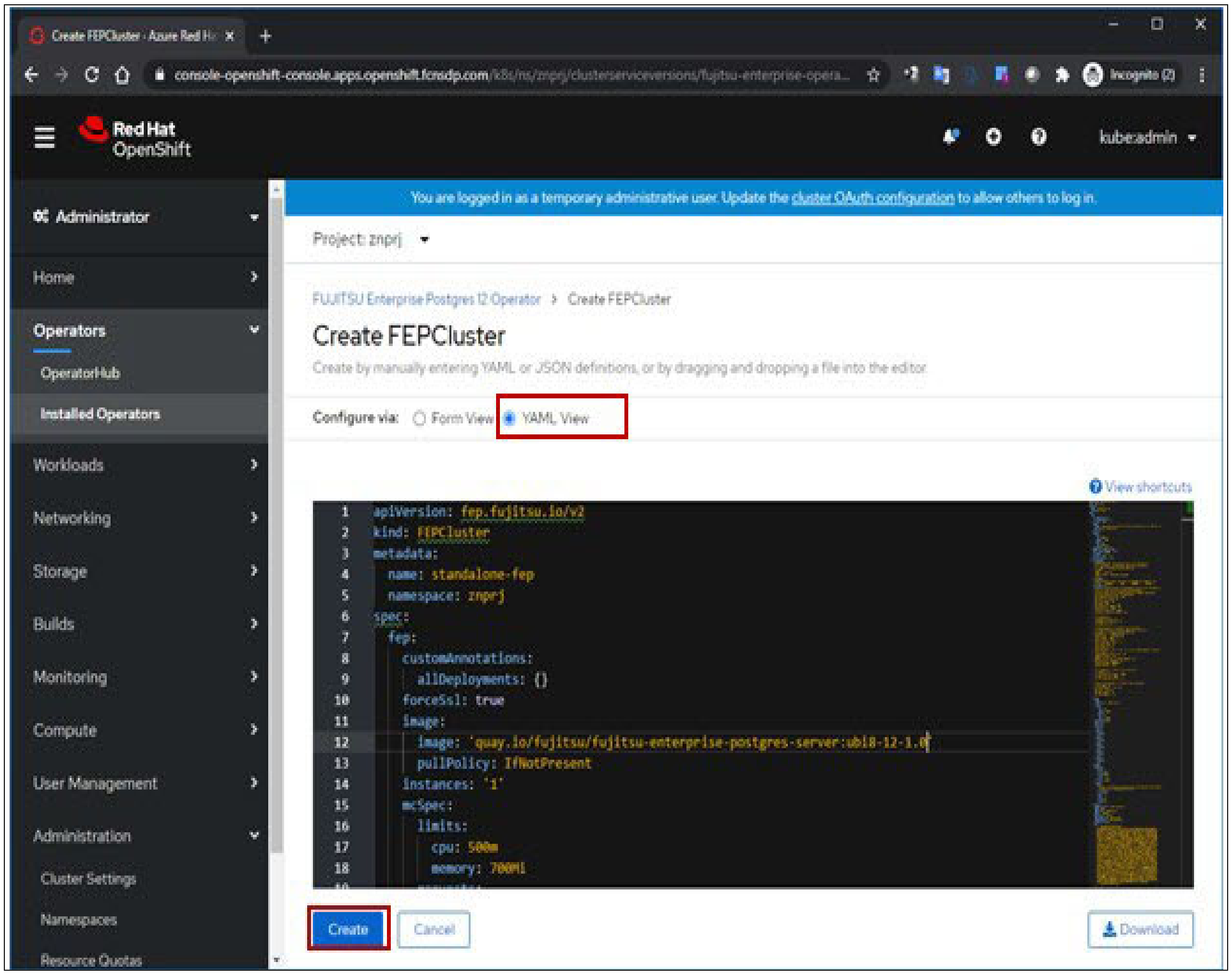


Figure 10-16 Creating a stand-alone FUJITSU Enterprise Postgres cluster

4. In the Create FEPCluster window, change the display to the YAML view and update the values, as shown in Table 10-10 on page 405. Click **Create** to create a cluster, as shown in Figure 10-17.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Create FEPCluster

Create by manually entering YAML or JSON definitions, or by dragging and dropping a file into the editor.

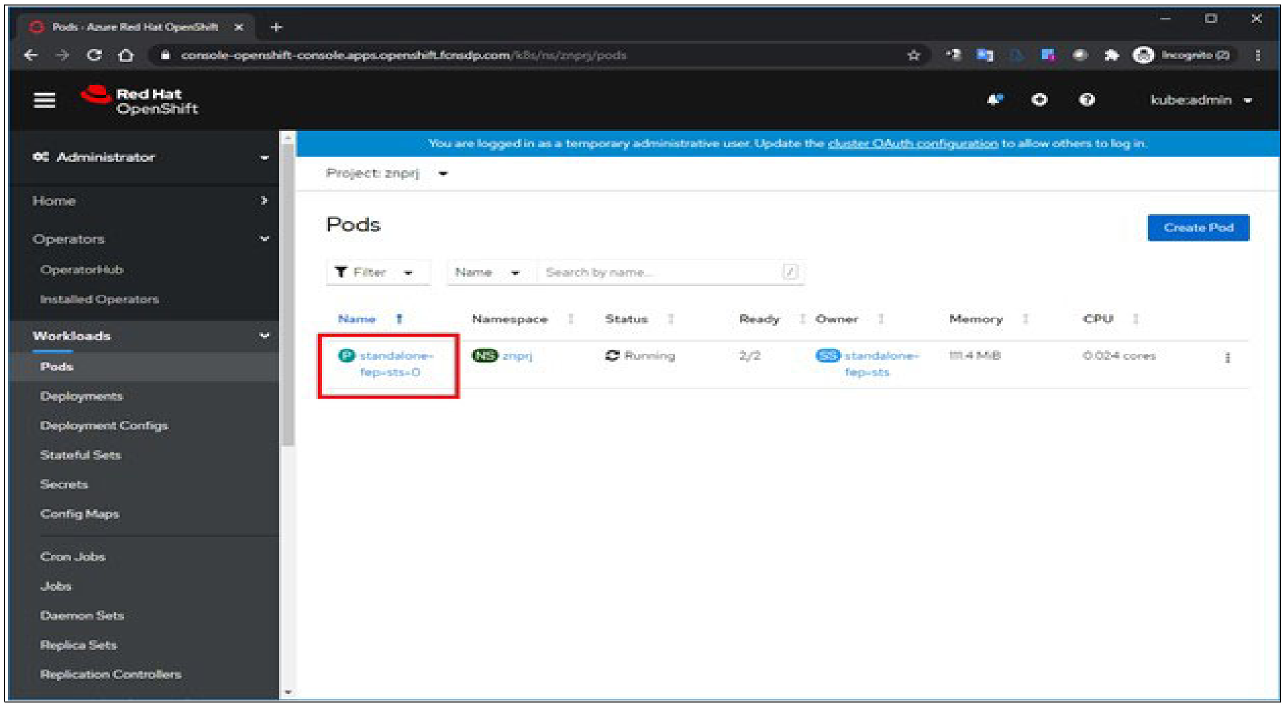
Configure via: Form View [✓ YAML View]

```
apiVersion: fep.fujitsu.io/v2
kind: FEPCluster
metadata:
  name: standalone-fep
  namespace: znprj
spec:
  fep:
    customAnnotations:
      allDeployments: {}
    forceSsl: true
    image:
      image: 'quay.io/fujitsu/fujitsu-enterprise-postgres-server:ubi8-12-1.0'
      pullPolicy: IfNotPresent
    instances: '1'
    mcSpec:
      limits:
        cpu: 500m
        memory: 700Mi
```

Figure 10-17 Updating the deployment parameters

After a while, check the deployment status by selecting **Workloads** → **Pods**.

The single node cluster is deployed, and the deployment status can be checked by selecting **Workloads** → **Pods** (`standalone-fep-sts-0` is the master stateful set for the cluster name `standalone-fep` that is specified in the CR configuration file). The status shows `Running` after the cluster is ready (Figure 10-18 on page 413).



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Pods

Name	Namespace	Status	Ready	Owner	Memory	CPU
P standalone-fep-sts-0	NS znprj	Running	2/2	SS standalone-fep-sts	111.4 MiB	0.024 cores

Figure 10-18 Checking the status of the deployment

Deploying a three-node HA FUJITSU Enterprise Postgres cluster by using a CLI

In the following example, we deploy a FUJITSU Enterprise Postgres cluster with three pods by using the **oc** CLI with a CR configuration file.

The cluster has one Postgres master and two Postgres replicas running in HA mode. If the master node fails, the built-in mechanism fails over that role to a different node with a minimal outage. Complete the following steps:

1. Create a CR Configuration file that is used as deployment file to define a FUJITSU Enterprise Postgres cluster. In this example, a CR file that is called `ha.yaml` is created, as shown in Example 10-11.

Example 10-11 HA cluster deployment YAML file

```
ngz@W10T-FAS200307:~/deploy$ cat standalone.yaml
apiVersion: fep.fujitsu.io/v2
kind: FEPCluster
metadata:
  name: ha-fep
spec:
  fep:
    forceSsl: true
    image:
      pullPolicy: IfNotPresent
      image: "quay.io/fujitsu/fujitsu-enterprise-postgres-server:ubi8-12-1.0"
    mcSpec:
      limits:
        cpu: 500m
        memory: 700Mi
      requests:
        cpu: 200m
        memory: 512Mi
    podAntiAffinity: false
    podDisruptionBudget: false
    instances: 3
    servicePort: 27500
    syncMode: "on"
    sysExtraLogging: false
  fepChildCrVal:
    # The values specified below are used ONLY at start-up to create child
    # Custom Resources - FEPVolume, FEPConfig, FEPUser
    customPgAudit: |
      # define pg audit custom params here to override defaults.
```

The CR configuration file includes references to the FUJITSU Enterprise Postgres server specification: name, SSL termination, CPU, memory, FUJITSU Enterprise Postgres server, and other items. Table 10-11 describes some of the fields that can be modified by using the CR configuration file.

Table 10-11 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`	The FUJITSU Enterprise Postgres cluster accepts only a TLS encrypted connection from the client.

`spec:`
`  fep:`
`    image:`
	`quay.io/fujitsu/xxx`	FUJITSU Enterprise Postgres server image.

`spec:`
`  fep:`
`  mcSpec:`
	`limits:`
	`  cpu: 500m`
	`  memory: 700Mi`
	`requests:`
	`  cpu: 200m`
	`  memory: 512Mi`
		Resource allocation to this container.

`spec:`
`  fep:`
`  instances:`
		Number of FUJITSU Enterprise Postgres pods in the cluster. For a stand-alone deployment, change this value to 1.

Field	Value	Details

`spec:`
`  fep:`
`    syncMode:`
	`on`	Replication mode (on for sync and off for async and stand-alone).

`spec:`
`  fepChildCrVal:`
`    customPgAudit:`
	`[output]`
	`logger = 'auditlog'`
	`log_directory = '/database/log/audit'`
	`log_truncate_on_rotation = on`
	`log_filename = 'pgaudit-%a.log'`
	`log_rotation_age = 1d`
	`log_rotation_size = 0[rule]`
		pgAudit configuration.

`spec:`
`  fepChildCrVal:`
 	
`customPgParams:`
	`shared_preload_libraries='pgx_datamasking,pg_prewarm'`
	`session_preload_libraries='pg_prewarm'`
		Custom values to go into `postgresql.conf`. The two lines that are listed here enable FUJITSU Enterprise Postgres unique features.

`spec:`
`  fepChildCrVal:`
`    sysUsers:`
		Password for Postgres system users.

The CR configuration parameter details are shown in Table 10-11 on page 414 and highlighted in red in Example 10-12.

Example 10-12 Configuration parameter details

```
apiVersion: fep.fujitsu.io/v2
kind: FEPCluster
metadata:
  name: ha-fep
spec:
  fep:
    forceSsl: true
    image:
      pullPolicy: IfNotPresent
      image: "quay.io/fujitsu/fujitsu-enterprise-postgres-server:ubi8-12-1.0"
    mcSpec:
      limits:
        cpu: 500m
        memory: 700Mi
      requests:
        cpu: 200m
        memory: 512Mi
    podAntiAffinity: false
    podDisruptionBudget: false
    instances: 3
    servicePort: 27500
    syncMode: "on"
    sysExtraLogging: false
  fepChildCrVal:
```

```
# The values specified below are used ONLY at start-up to create child
# Custom Resources - FEPVolume, FEPConfig, FEPUser
customPgAudit: |
  # define pg audit custom params here to override defaults.
  [output]
  logger = 'auditlog'
  log_directory = '/database/log/audit'
  log_truncate_on_rotation = on
  log_filename = 'pgaudit-%a.log'
  log_rotation_age = 1d
  log_rotation_size = 0
  [rule]
customPgHba: |
  # define pg_hba custom rules here to be merged with default rules.
  # TYPE DATABASE USER ADDRESS METHOD
customPgParams: |
  # define custom postgresql.conf parameters below to override defaults.
  # Current values are as per default FUJITSU Enterprise Postgres deployment
  shared_preload_libraries='pgx_datamasking,pg_prewarm'
  session_preload_libraries='pg_prewarm'
  max_prepared_transactions = 100
  max_worker_processes = 20
  max_connections = 100
  work_mem = 1MB
  maintenance_work_mem = 12MB
  shared_buffers = 128MB
  effective_cache_size = 384MB
  checkpoint_completion_target = 0.8
  # tcp parameters
  tcp_keepalives_idle = 30
  tcp_keepalives_interval = 10
  tcp_keepalives_count = 3
  # logging parameters in default fep installation
  # if log volume is not defined, log_directory should be
  # changed to '/database/userdata/data/log'
  log_filename = 'logfile-%a.log'
  log_file_mode = 0600
  log_truncate_on_rotation = on
  log_rotation_age = 1d
  log_rotation_size = 0
  log_checkpoints = on
  log_line_prefix = '%e %t [%p]: [%l-1] user=%u,db=%d,app=%a,client=%h'
  log_lock_waits = on
  log_autovacuum_min_duration = 60s
  logging_collector = on
  log_directory = '/database/userdata/data/log'
  pgaudit.config_file='/opt/app-root/src/pgaudit-cfg/pgaudit.conf'
  log_replication_commands = on
  log_min_messages = WARNING
  log_destination = stderr
  # wal_archive parameters in default fep installation
  archive_mode = on
  archive_command = '/bin/true'
  wal_level = replica
  max_wal_senders = 12
```

```
    wal_keep_segments = 64
  storage:
    dataVol:
      size: "5Gi"
    walVol:
      size: "1200Mi"
  sysUsers:
    pgAdminPassword: admin-password
    pgdb: mydb
    pgpassword: mydbpassword
    pguser: mydbuser
    pgrepluser: repluser
    pgreplpassword: repluserpwd
    tdepassphrase: tde-passphrase
  systemCertificates:
    key: |-
      -----BEGIN RSA PRIVATE KEY-----
      MIIEowIBAAKCAQEA4AI33yvHZws+jta6qpV6wzJqF8odIfTIpCfbrVcUUtLFKJ1I
      2e4SceTKi6O3C/I1XuvWlpng5IO65+fQQLOO6z1/AuQT78YUn/Wlm9x1aHVsv4AN
      B5JWWqDOjrRT3o7nRPGXfilabPOrGE2mJJcVR9nExJ3IeaktgT3sb8YlXvtchyYp
      mjdbfxabTzO7igO+6/cwKoRRxOK8Uf7f5euEOcI/49OJ6r5Rs4lgD8sIQNCUFlTF
      YvmAH7gcdssSFBt8NPlUATHEsoFmlWODKCJWNhTLOht+s6L/1zwTHLjPG2pdkG6W
      dgmu5H2pDml8CDNLDv98Aj7i+I5SRKKcVPlnuQIDAQABAoIBAFPQYKlOzw/+BAOb
      yMIUpdctIMb/54CR/xROmVw1DbSjigNVPjHUQvB8Y1B2FAITQObgJOO6bAvOQdWN
      RbO/v/yYiNJDFjaLjaIAHlO/2+oWrXbFaZqgpVDJhB+e1xaZr2x7XGxm+p925k3O
      l6pvIRY+I8JRKvZiV1VZHwL/R3JOtPr++xMZtLVjVOI+f+ySqJ+TZHuAjm49EKxj
      cEmmJ28b7QcziXsvKyOOf+zbqLIBKXQdZAFU5eEr1BsDRXdRW+KfOXIvftuy4BJZ
      voKT+VGhEvF/qysswL4+6IAO6tpuYnnMOY2d3sOGoWPkTcQKOMekYKzL/WmtCjNs
      9hodJtECgYEA5EWyhEOf4uOKe5TDp697UCUvXLoOR58FDe/S8XNvScn29jjOkqIg
      OMoqo9xAkJTNTzqn5UUdt1x/pgM2NxlPLFijrcOzQlX3SoOO2ryDd9WNi7YKtN16
      KJqa536WeZu2OEbuAZ+S3GALVy1RPeTNPnUOmKnFO6DjDUGzLNCZy1OCgYEA+zfw
      952DWuz1UOZ4wvAEqqcgUKXPKrkTXV/iUnjkDkrLYVrOZofDNTXrdHl+UedFmaOC
      cieZn6DNhcdz5tKtyysGMH3g/qs9PfoGUngvcXsyOEgkO4l3x1jc8TTCLqXZXYaQ
      HMsx51n+R58oncPtzYSUOr9qQ6PbC2CstTbFJAOCgYEAjGEsUliAB/jknfEzjXjG
      PdhQUxb8VyE864Az2lah9t/kJzFyIAziAeqZ5GE7t247AGFTBRTHHI8e1Qoemi3P
      Wbc9GVIbFs1lIYbcIDpUIyrKPEP8O5QEXtoNLxXTFgAjRGKiVY87spjCAJ+W2ZhO
      e/1it5GYXfgQCYQA2yuBmOUCgYANRkR2YR1axaCk+NlSu6oTdmdPu6M5x7PNQE7O
      OtMaKjua9lppvIzFGAdMDUtueoEEAE7ZR1xnwfB6PDLUpJdIYAqgr1YfPt8qkjaZ
      Tv56yZ7CwLOpbF8m6nwqRrZoDp1wwraEvvvxFKFKGY/k3kCHlpTakdjEoDjn3gDi
      RnWeVQKBgCEneMSzucei5LRppRtRaJw/Btll8qlPMlX3W7dxQ3cLwpmLOnOm51Fp
      PIZ44zYK8R6fu4+/sSrlfaIg86Ugeufp6YNxyNROKxUGza5vDIu5OftwWtBeg+UK
      Z8lLWNdX6pp7WMujmF3H1DrkBbauYMUKZ4UxUYtelgHERMePIxwb
      -----END RSA PRIVATE KEY-----
    crt: |-
      -----BEGIN CERTIFICATE-----
```

2. Apply the CR configuration file `ha.yaml` by using the following command, as shown in Example 10-13:

```
oc apply -f ha.yaml
```

Example 10-13 Deploying an HA cluster

```
ngz@W10T-FAS200307:~/deploy$ oc apply -f ha.yaml
fepcluster.fep.fujitsu.io/ha-fep created
ngz@W10T-FAS200307:~/deploy
```

3. The HA cluster is deployed, and the deployment status can be checked by using the commands that are shown in Example 10-14 (`ha-fep-sts-0` is the master stateful set and `ha-fep-sts-1` and `ha-fep-sts-2` are replica master sets for the cluster name `ha-fep` that is specified in the CR configuration file). The status shows `Running` after the cluster is ready.

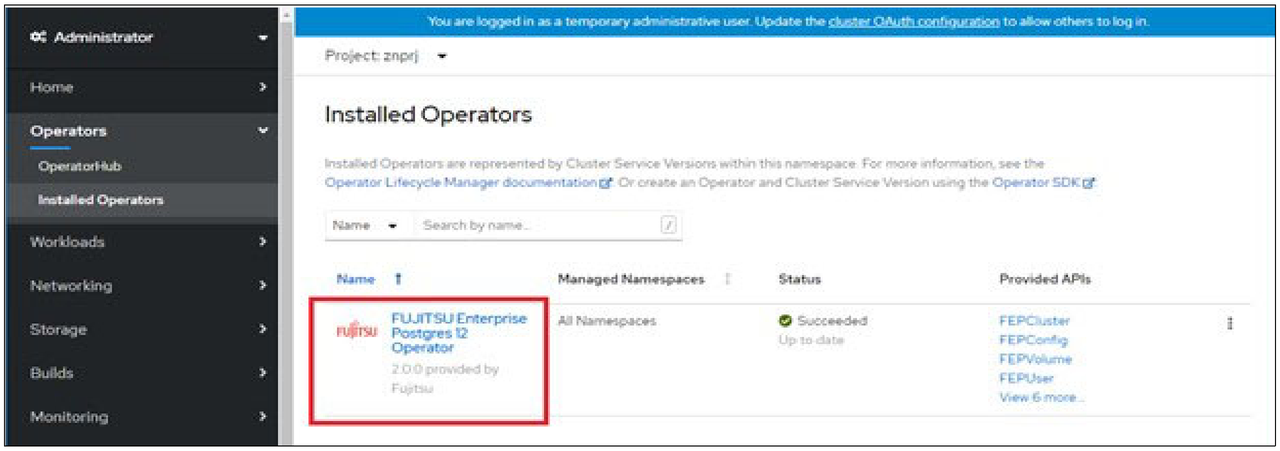
Example 10-14 Checking the HA cluster deployment status

```
ngz@W10T-FAS200307:~/deploy$ oc get fepcluster ha-fep
NAME             AGE
ha-fep           2m12s
ngz@W10T-FAS200307:~/deploy$ oc get pod -l app=ha-fep-sts -L feprole
NAME                   READY   STATUS    RESTARTS   AGE     FEPROLE
ha-fep-sts-0           1/1     Running   0          4m37s   master
ha-fep-sts-1           1/1     Running   0          4m37s   replica
ha-fep-sts-2           1/1     Running   0          4m37s   replica
```

10.4.4 Deploying a three-node HA FUJITSU Enterprise Postgres cluster by using a Red Hat OpenShift console

To deploy a three-node HA FUJITSU Enterprise Postgres cluster by using a Red Hat OpenShift console, complete the following steps:

1. In the Red Hat OpenShift Console, select **Operators** → **Installed Operators**.
2. Select the FUJITSU Enterprise Postgres 12 Operator, as shown in Figure 10-19.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

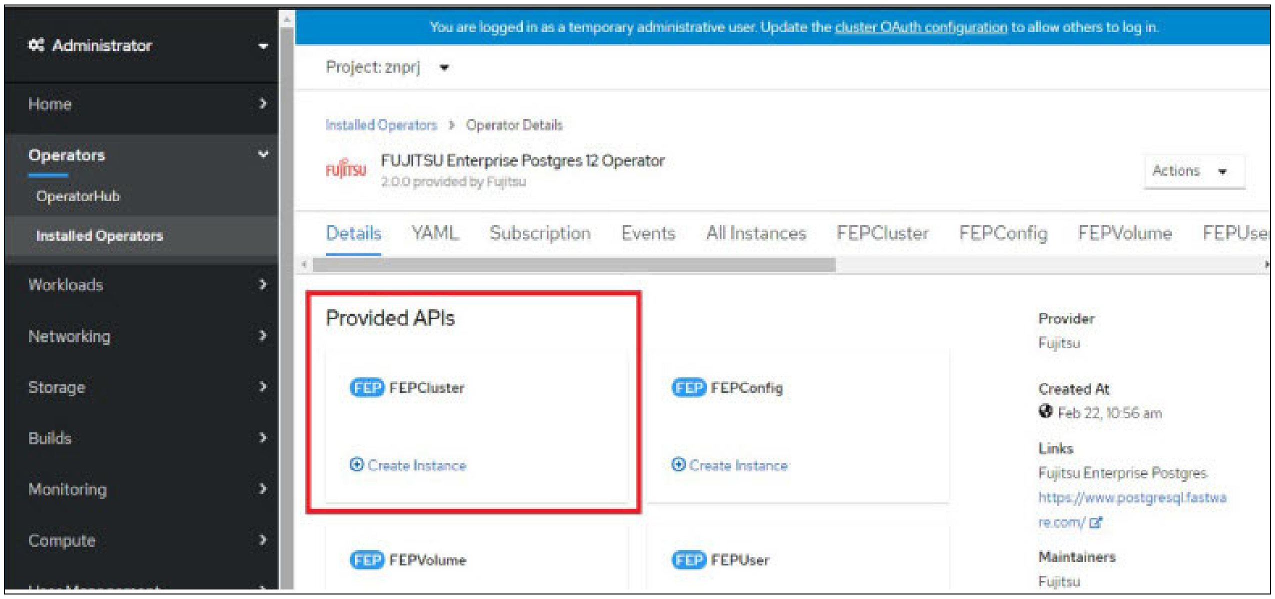
Installed Operators

Installed Operators are represented by Cluster Service Versions within this namespace. For more information, see the Operator Lifecycle Manager documentation. Or create an Operator and Cluster Service Version using the Operator SDK.

Name	Managed Namespaces	Status	Provided APIs
FUJITSU Enterprise Postgres 12 Operator 2.0.0 provided by Fujitsu	All Namespaces	Succeeded Up to date	FEPCluster FEPConfig FEPVolume FEPUser View 6 more...

Figure 10-19 Installed Operators

3. In the Operator window, select **Create Instance**, as shown in Figure 10-20 on page 419.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

FUJITSU Enterprise Postgres 12 Operator
2.0.0 provided by Fujitsu

Provider
Fujitsu

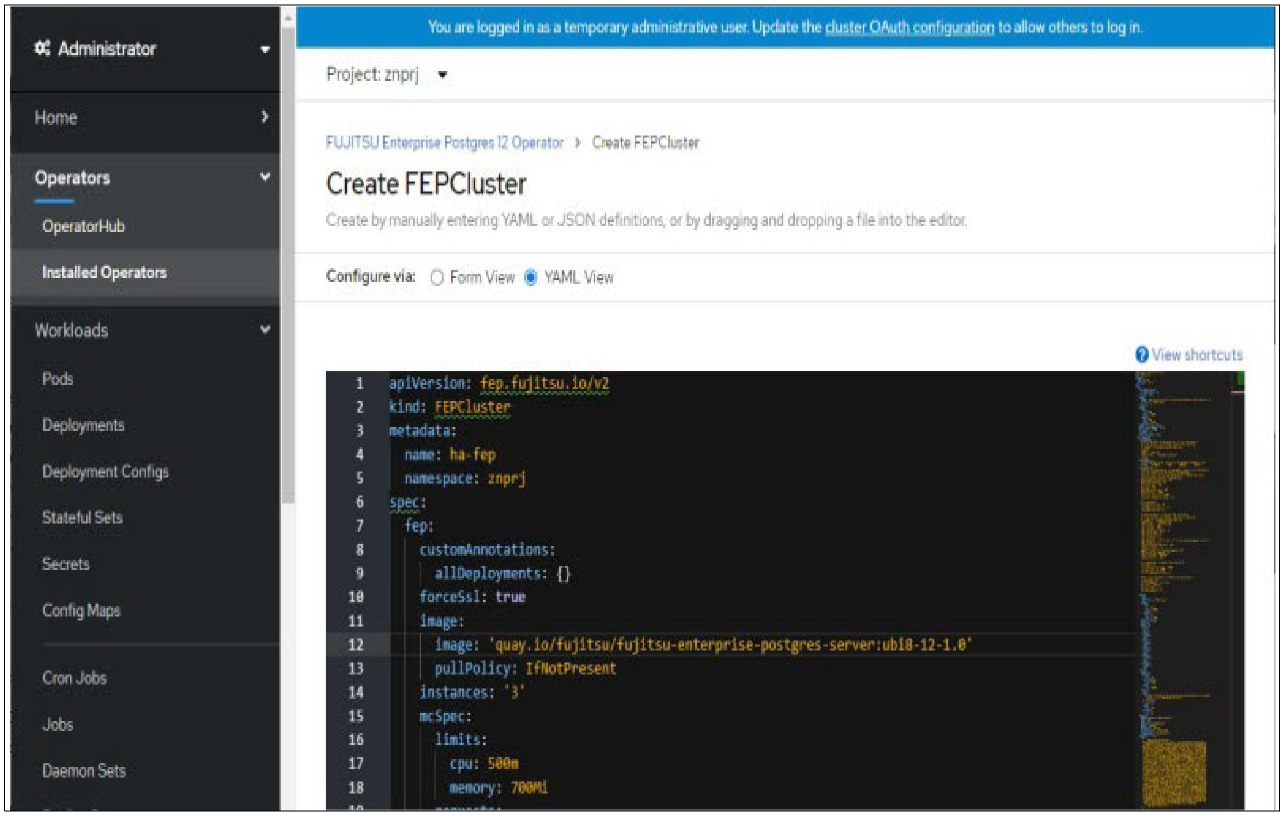
Created At
Feb 22, 10:56 am

Links
Fujitsu Enterprise Postgres
https://www.postgresql.fastware.com/

Maintainers
Fujitsu

Figure 10-20 Creating a cluster

4. In the Create FEPCluster window, change the display to **YAML View**, update the values as shown in Table 10-11 on page 414 and the CR configuration parameters as shown in Figure 10-21, and click **Create** to create a cluster.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Create FEPCluster

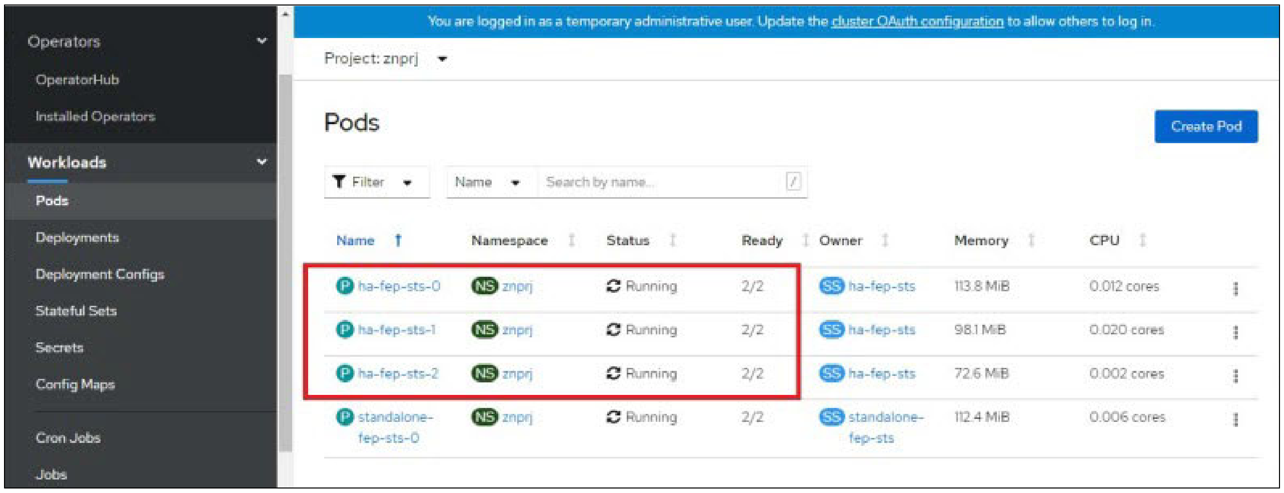
Create by manually entering YAML or JSON definitions, or by dragging and dropping a file into the editor.

Configure via: Form View [✓ YAML View]

```
apiVersion: fep.fujitsu.io/v2
kind: FEPCluster
metadata:
  name: ha-fep
  namespace: znprj
spec:
  fep:
    customAnnotations:
      allDeployments: {}
    forceSsl: true
    image:
      image: 'quay.io/fujitsu/fujitsu-enterprise-postgres-server:ubi8-12-1.0'
      pullPolicy: IfNotPresent
    instances: '3'
    mcSpec:
      limits:
        cpu: 500m
        memory: 700Mi
```

Figure 10-21 HA cluster deployment

5. The single node cluster is deployed, and the deployment status can be checked by selecting **Workloads** → **Pods**, as shown in Figure 10-22 (`ha-fep-sts-0` is the master stateful set, and `ha-fep-sts-1` and `ha-fep-sts-2` are the replica master sets for the cluster name `ha-fep` that is specified in the CR configuration file). The status shows `Running` after the cluster is ready.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Name	Namespace	Status	Ready	Owner	Memory	CPU
P ha-fep-sts-0	NS znprj	Running	2/2	SS ha-fep-sts	113.8 MiB	0.012 cores
P ha-fep-sts-1	NS znprj	Running	2/2	SS ha-fep-sts	98.1 MiB	0.020 cores
P ha-fep-sts-2	NS znprj	Running	2/2	SS ha-fep-sts	72.6 MiB	0.002 cores
P standalone-fep-sts-0	NS znprj	Running	2/2	SS standalone-fep-sts	112.4 MiB	0.006 cores

Figure 10-22 HA Cluster Deployment result

All FUJITSU Enterprise Postgres pods must show the status as `Running`.

10.4.5 Changing FUJITSU Enterprise Postgres cluster configurations

It is possible to change the FUJITSU Enterprise Postgres cluster configurations after deployment. There are several types of changes, and some of them involve redeploying the pod, some trigger a Postgres restart, and others reload Postgres without restarting the pod (Table 10-12).

Table 10-12 Changes to cluster configurations

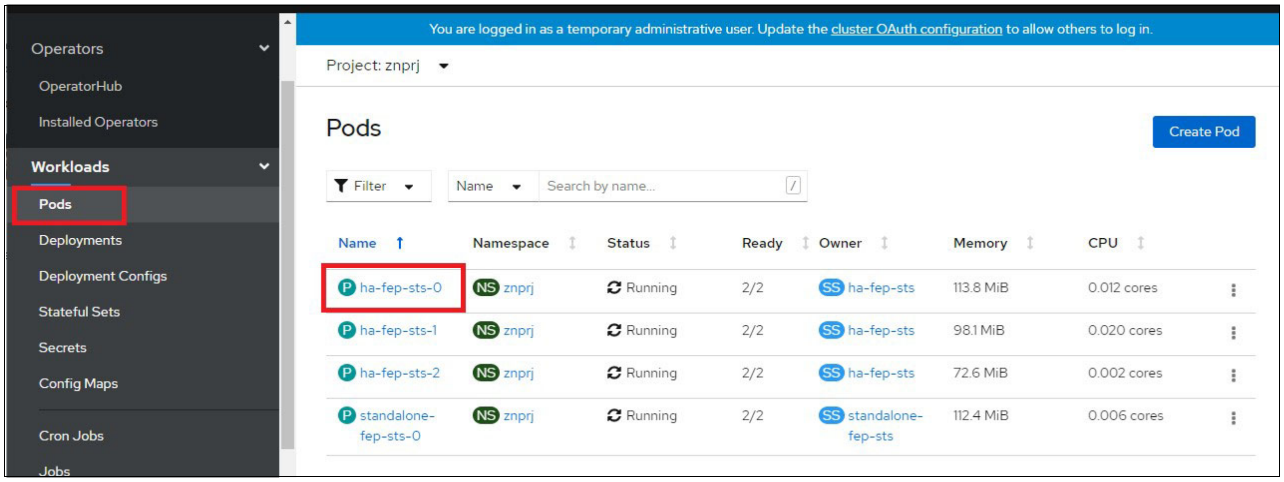
Type of change	Behavior	Follow-up action
Changing the pod definition, such as CPU/Memory allocation, except for the FUJITSU Enterprise Postgres container image.	New settings are applied to a stateful set. To minimize outage, the pods are not redeployed automatically.	The user should schedule a time to delete the pods and let them redeploy with new settings.
Changing the FUJITSU Enterprise Postgres container image.	The new image name is applied to a stateful set. The Pods are redeployed in a controlled manner, starting with all the replica pods. The FUJITSU Enterprise Postgres Operator performs a switchover and updates the master pod last. There is a small outage on the cluster.	No additional action is required by the user.

Type of change	Behavior	Follow-up action
Changing Postgres parameters that require restarting the FUJITSU Enterprise Postgres server.	New settings are applied to `postgresql.conf` on all pods. To minimize outages, the change is in a pending stage until a restart is performed by a user.	The user should schedule a time to restart the FUJ ITSU Enterprise Postgres server.
Changing Postgres parameters that can take effect by reloading FUJITSU Enterprise Postgres.	New settings are applied to `postgresql.conf` on all pods. FUJITSU Enterprise Postgres is reloaded, and new parameters take effect immediately.	No additional action is required by the user.

10.4.6 Changing the FUJITSU Enterprise Postgres cluster configuration by using the Red Hat OpenShift console

In this example, we demonstrate how to change FUJITSU Enterprise Postgres parameters that are reflected immediately. In this example, a three-node HA FUJITSU Enterprise Postgres Cluster is used. Complete the following steps:

1. In the Red Hat OpenShift Console, select **Workloads** → **Pods**.
2. Select one of the pods, as shown in Figure 10-23.



You are logged in as a temporary administrative user. Update the cluster OAuth configuration to allow others to log in.

Pods

Name	Namespace	Status	Ready	Owner	Memory	CPU
P ha-fep-sts-0	NS znprj	Running	2/2	SS ha-fep-sts	113.8 MiB	0.012 cores
P ha-fep-sts-1	NS znprj	Running	2/2	SS ha-fep-sts	98.1 MiB	0.020 cores
P ha-fep-sts-2	NS znprj	Running	2/2	SS ha-fep-sts	72.6 MiB	0.002 cores
P standalone-fep-sts-0	NS znprj	Running	2/2	SS standalone-fep-sts	112.4 MiB	0.006 cores

Figure 10-23 Inspecting a pod

3. In the Pod Details window, select the **Terminal** tab.
4. In the terminal shell, connect to the database and issue the command that is shown in Example 10-15 to check the current value of the configured parameter **`checkpoint_completion_target`**.

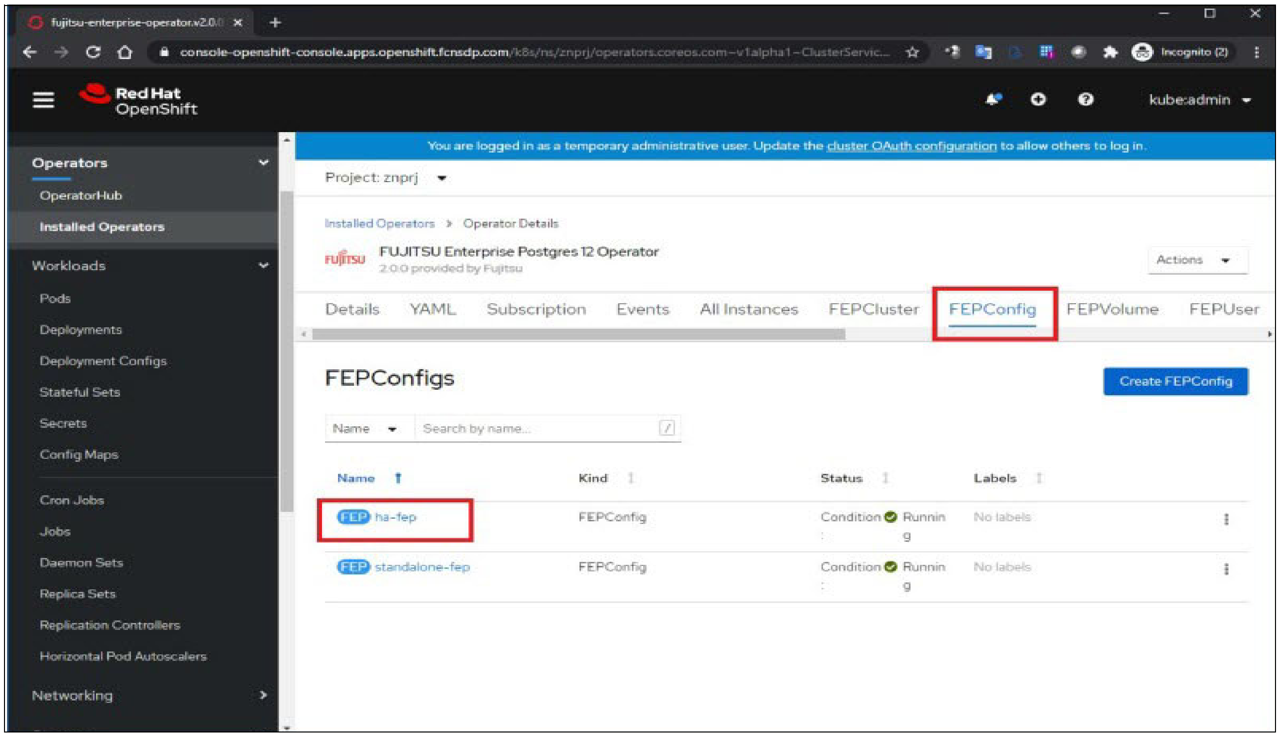
Example 10-15 Checking the current value of the configured parameter

```
sh-4.4$ psql -U postgres postgres
Password for user postgres:
psql (12.1)
Type "help" for help.
postgres=# show checkpoint_completion_target ;
 checkpoint_completion_target
------------------------------
```

```
 0.8
(One row)
postgres=#
```

Note: The value is `0.8` for the parameter **`checkpoint_completion_target`**.

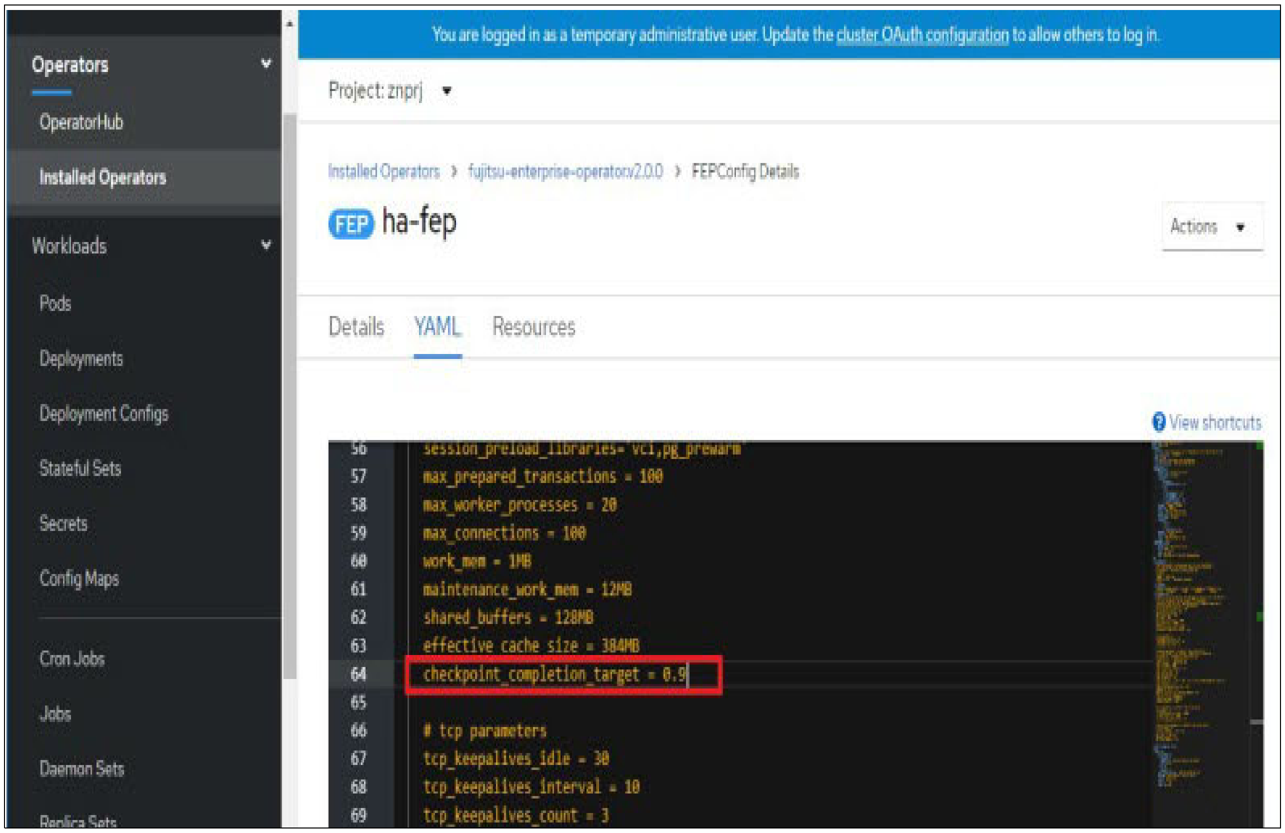
5. Select **Operators** → **Installed Operators**.
6. Select the FUJITSU Enterprise Postgres 12 Operator.
7. Select the **FEPConfig** tab, as shown in Figure 10-24.
8. Select the **ha-fep** cluster that we deployed.



Name	Kind	Status	Labels
FEP ha-fep	FEPConfig	Condition: Running	No labels
FEP standalone-fep	FEPConfig	Condition: Running	No labels

Figure 10-24 FEPConfig

9. In the FEPConfig Details window, select the **YAML** tab.
10. Locate the line `checkpoint_completion_target` and update the value from `0.8` to `0.9`, as shown in Figure 10-25 on page 423.



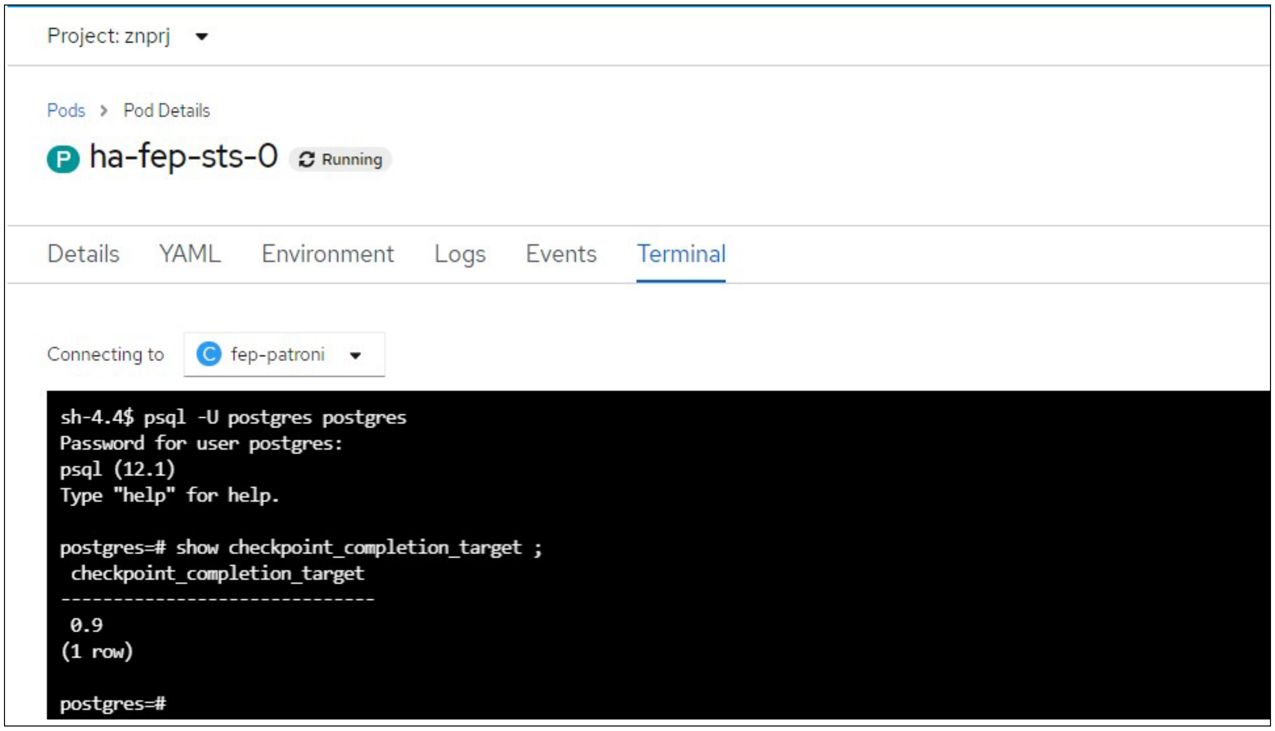
```
session_preload_libraries='vci,pg_prewarm'
max_prepared_transactions = 100
max_worker_processes = 20
max_connections = 100
work_mem = 1MB
maintenance_work_mem = 12MB
shared_buffers = 128MB
effective_cache_size = 384MB
checkpoint_completion_target = 0.9

# tcp parameters
tcp_keepalives_idle = 30
tcp_keepalives_interval = 10
tcp_keepalives_count = 3
```

Figure 10-25 Updating the FEPConfig CR

11. Click **Save** to apply the change.
12. Wait for a minute for FUJITSU Enterprise Postgres Operator to apply the change.

13. Repeat steps 1 on page 421 - 4 on page 421 to check the value again. The results are shown in Figure 10-26.


Project: znprj
Pods > Pod Details
P ha-fep-sts-0 Running
Details YAML Environment Logs Events Terminal
Connecting to C fep-patroni

```
sh-4.4$ psql -U postgres postgres
Password for user postgres:
psql (12.1)
Type "help" for help.

postgres=# show checkpoint_completion_target ;
 checkpoint_completion_target
------------------------------
 0.9
(1 row)

postgres=#
```

Figure 10-26 Checking the updated results

Note: The value for parameter **`checkpoint_completion_target`** is updated to `0.9` without a server restart.

10.5 FUJITSU Enterprise Postgres cluster CR configuration file explanation

FUJITSU Enterprise Postgres 12 Operator creates the FUJITSU Enterprise Postgres cluster and associated child CRs with the specified information in a CR YAML file. The descriptions and constraints of each field are provided in Table 10-13.

Table 10-13 Cluster parameters

FEPCluster CR parameters (with default values)	Remarks
`apiVersion`: `fep.fujitsu.io/v2`	Mandatory as is.
`kind`: `FEPCluster`	Mandatory as is.
`metadata`:	
`name`: `new-fep`	The name of the cluster, which must be unique in the namespace.
`namespace`: `<demo>`	The namespace, which OCP populates as current.
`spec`:	

FEPCluster CR parameters (with default values)	Remarks
fep:	
forceSsl: `true`	► True: Forces all connections to FUJITSU Enterprise Postgres to use SSL. ► False: Connections can be with or without SSL.
image:	
pullPolicy: `IfNotPresent`	Image-pull policy from Kubernetes.
image: `"quay.io/fujitsu/fep-server-test:ubi8-patroni_79"`	FUJITSU Enterprise Postgres image URL. Update this URL to a new image URL to upgrade all FUJITSU Enterprise Postgres containers to run a new image. When you save a new image, the upgrade process starts, and pods restart one-by-one. Before restarting the original master pod, switch over the master role to one of the upgraded pods. Now, the original master pod starts as a replica to get the new image.
mcSpec:	
limits:	
cpu: `500m`	Maximum CPU that is allocated to a FUJITSU Enterprise Postgres container.
memory: `700Mi`	Maximum memory that is allocated to a FUJITSU Enterprise Postgres container.
requests:	
cpu: `200m`	Initial CPU allocation for a FUJITSU Enterprise Postgres container.
memory: `512Mi`	Initial memory allocation for a FUJITSU Enterprise Postgres container.
podAntiAffinity: `true`	► True: Multiple FUJITSU Enterprise Postgres pods should not run a node. ► False: Many FUJITSU Enterprise Postgres pods can run on a node.
podDisruptionBudget: `true`	True: A maximum of one node can be unavailable.
instances: `3`	Number of FUJITSU Enterprise Postgres nodes in a cluster.
servicePort: `27500`	FUJITSU Enterprise Postgres service port.
syncMode: `"on"`	► On: FUJITSU Enterprise Postgres cluster with sync replicas. ► Off: FUJITSU Enterprise Postgres cluster with async replicas.
sysExtraLogging: `false`	To turn on extra debugging, set the value to `true`. It can be turned on or off at any time.

FEPCluster CR parameters (with default values)	Remarks
changeSecurityContext: `true` (Not defined on OCP.)	Defined only on a cloud-native platform. This key must *not* be defined for the Red Hat OpenShift platform.
fepChildCrVal: The values that are specified below are used *only* at start-up to #create a child. The CRs are FEPVolume, FEPConfig, FEPUser, and EPCert.	All values under this section are used *only* when creating child CRs FEPConfig, FEPVolume, FEPUser, and FEPCert. Afterward, all changes that are needed for values under this section must be done by using child CRs.

customPgAudit: Define the **pgaudit** custom parameters here to override the defaults. If the log volume is not defined, **log_directory** should be changed to `'/database/userdata/data/log'`.
`[output]`
`logger = 'auditlog'`
`log_directory = '/database/log/audit'`
`[rule]`
	The initial values for `FEPConfig` for **pgAudit** configuration. The `log_directory` location depends on **logVol**:
	► If **logVol** is defined, then `log_directory = '/database/log/audit'`.
	► If **logVol** is not defined, then `log_directory = '/database/userdata/data/log'`.

customPgHba: Define the pg_hba custom rules here so that they merge with the default rules. `# TYPE  DATABASE  USER   ADDRESS   METHOD`	The initial values for `FEPConfig` for the `pg_hba.conf` data. All the lines that must be inserted into `pg_hba.conf` in addition to the default configuration.

customPgParams: Define the custom `postgresql.conf` parameters below to override the defaults. The current values are per the default FUJITSU Enterprise Postgres deployment.
`shared_preload_libraries='pgx_datamasking,vci,`
`pgaudit,pg_prewarm'`
`session_preload_libraries='vci,pg_prewarm'`
`max_prepared_transactions = 100`
`max_worker_processes = 20`
`max_connections = 10`
`work_mem = 1MB`
`maintenance_work_mem = 12MB`
`shared_buffers = 128MB`
`effective_cache_size = 384MB`
`checkpoint_completion_target = 0.8`
	The initial values for `FEPConfig` for the `postgresql.conf` data. This section defines the values of the parameters that would overwrite the values that are defined in the default `postgresql.conf`.
	The values are currently used for a typical deployment of a cluster. They can be modified as needed except for the value of **log_directory**.
	The **log_directory** location depends on **logVol**:
	► If **logVol** is defined, then `log_directory = '/database/log'`.
	► If logVol is not defined, then `log_directory = '/database/userdata/data/log'`.
	In a typical deployment, all FUJITSU Enterprise Postgres advanced features are enabled by default, and parameters are set. These features are TDE, data masking, VCI, and GMC.

FEPCluster CR parameters (with default values)	**Remarks**

Logging parameters in a default FUJITSU Enterprise Postgres installation. If the log volume is not defined, **`log_directory`** should be changed to `'/database/userdata/data/log'`.
`log_directory = '/database/log'`
`log_filename = 'logfile-%a.log'`
`log_file_mode = 0600`
`log_truncate_on_rotation = on`
`log_rotation_age = 1d`
`log_rotation_size = 0`
`log_checkpoints = on`
`log_line_prefix = '%e %t [%p]: [%l-1] user=%u,`
`db=%d,app=%a,client=%h'`
`log_lock_waits = on`
`log_autovacuum_min_duration = 60s`
`logging_collector = on`
`pgaudit.config_file='/database/pgaudit.conf'`
`log_replication_commands = on`
`log_min_messages = WARNING`
`log_destination = stderr`
`# tcp parameters`
`tcp_keepalives_idle = 30`
`tcp_keepalives_interval = 10`
`tcp_keepalives_count = 3`
The **`wal_archive`** parameters in a default FUJITSU Enterprise Postgres installation:
`archive_mode = on`
`archive_command = 'pgbackrest --stanza=backupstanza`
`--config=/database/userdata/pgbackrest.conf`
`archive-push %p'`
`wal_level = replica`
`max_wal_senders = 12`
`wal_keep_segments = 64`
	Typical values of `postgresql.conf` continue.

`storage`:	Defines start-up values for the `FEPVolume` CR. Volumes for a cluster can be defined *only* at the start. Volumes *cannot* be created and mounted later. Limited changes can be done by using the `FEPVolume` CR, and only if the underlying storage allows it. For example, the size of a volume may be increased if the underlying `storageClass` allows it. However, the size of a volume cannot be shrunk.
`dataVol`:	Mandatory to store a database.
`size`: `"2Gi"`	The size for the data volume, which must be defined.
`storageClass`: `default` (optional entry)	**`storageClass`** for a data volume is optional if it is omitted. It uses the default storage class of the system.

FEPCluster CR parameters (with default values)	Remarks
accessModes: [“ReadWriteOnce”] (optional entry)	**accessModes** for the data volume is optional. If it is omitted, use “ReadWriteOnce”. Other values are “ReadWriteMany” or “ReadOnly”, as defined by Kubernetes. Multiple modes can be used if separated by commas. ReadWriteMany works only if the underlying storage supports shared volumes.
tablespaceVol:	**tablespaceVol** is optional. Volumes are required to use TDE, but are optional for FUJITSU Enterprise Postgres clusters.
size: "512Mi"	The size for a table space volume must be defined.
storageClass: default (optional entry)	**storageClass** for the **tablespaceVol** is optional. If it is if omitted, the default storage class of system is used.
accessModes: [“ReadWriteOnce”] (optional entry)	**accessModes** for **tablespaceVol** is optional. If it is omitted, “ReadWriteOnce” is used. Other values, such as “ReadWriteMany” or “ReadOnly” as defined by Kubernetes, may be used. Multiple modes can be used if they are separated by commas. ReadWriteMany work only if the underlying storage supports shared volumes.
walVol:	**walVol** is mandatory.
size: "1200Mi"	The size for **walVol** must be defined.
storageClass: default (optional entry)	**storageClass** for **walVol** is optional. If it is omitted, use the default storage class of system.
accessModes: [“ReadWriteOnce”] (optional entry)	**accessModes** for **walVol** is optional. If it is omitted, use “ReadWriteOnce”. Other values like “ReadWriteMany” or “ReadOnly” as defined by Kubernetes may be used. Multiple modes can be used by being separated by commas. ReadWriteMany works only if the underlying storage supports shared volumes.
archivewalVol:	**archivewalVol** is optional.
size: "1Gi"	**size** for **archivewalVol** must be defined if **archivewalVol** is defined.
storageClass: default (optional entry)	**storageClass** for **archivewalVol** is optional. If it is omitted, use the default storage class of system.
accessModes: [“ReadWriteOnce”] (optional entry)	**accessModes** for **archivewalVol** is optional. If it is omitted, use “ReadWriteOnce”. Other values like “ReadWriteMany” or “ReadOnly” as defined by Kubernetes may be used. Multiple modes can be used if separated by commas. ReadWriteMany work only if the underlying storage supports shared volumes.

FEPCluster CR parameters (with default values)	**Remarks**
logVol:	**logVol** is optional.
size: "1Gi"	**size** for **logVol** must be defined if **logVol** is defined.
storageClass: default	**storageClass** for **logVol** is optional. If it is omitted, use the default storage class of system.
accessModes: ["ReadWriteOnce"] (optional entry)	**accessModes** for **logVol** is optional. if it is omitted, use "ReadWriteOnce". Other values like "ReadWriteMany" or "ReadOnly" as defined by Kubernetes may be used. Multiple modes can be used if separated by commas. ReadWriteMany work only if the underlying storage supports shared volumes.
backupVol:	**backupVol** is optional.
size: "2Gi"	**size** for **backupVol** must be defined if **backupVol** is defined.
storageClass: default (optional entry)	**storageClass** for **backupVol** is optional. If it is omitted, use the default storage class of system.
accessModes: ["ReadWriteOnce"] (optional entry)	**accessModes** for **backupVol** is optional. If it is omitted, use "ReadWriteOnce". Other values like "ReadWriteMany" or "ReadOnly" as defined by Kubernetes may be used. Multiple modes can be used if separated by commas. ReadWriteMany works only if the underlying storage supports shared volumes.
sysUsers:	This section defines start-up values for the FEPUser child CR. Subsequent changes to passwords and passphrase can be done by using the FEPUser CR. After the CRs are created, the passwords and passphrase are masked in both places for FEPCluster and FEPUser.
pgAdminPassword: admin-password	Password for the postgres superuser.
pgdb: mydb	Name of a created user DB (for example, **mydb**). This value cannot be changed after initial creation.
pgpassword: mydbpassword	Password for **pguser** (next field).
pguser: mydbuser	Name of the user for the created user DB (for example, the user for mydb) This value cannot be changed after initial creation.
pgrepluser: repluser	Name of a replication user. It is used for setting up replication between a primary and a replica in FUJITSU Enterprise Postgres Cluster. This value cannot be changed after initial creation.

FEPCluster CR parameters (with default values)	Remarks
pgreplpassword: `repluserpwd`	Password for the created user for replication (for example, `repluser`).
tdepassphrase: `tde-passphrase`	Passphrase for TDE. This field is mandatory because TDE is enabled by default.
systemCertificates:	This section defines the start-up values for the `FEPCert` child CR. Subsequent changes to `certs` and `key` can be done by using the `FEPCert` CR. After the CRs are created, `certs` and `key` are masked at both places in `FEPCluster` and `FEPCert`. These fields are mandatory to secure communication to FUJITSU Enterprise Postgres containers.
key: |- -----BEGIN RSA PRIVATE KEY----- *key value in terms of string of characters specified between the lines* -----END RSA PRIVATE KEY-----	Specify the RSA private key value between the `BEGIN` and `END` lines.
crt: |- -----BEGIN CERTIFICATE----- *certificate value in terms of* string of characters specified *between the lines* -----END CERTIFICATE-----	Specify the certificate value between the `BEGIN` and `END` lines.
cacrt: |- -----BEGIN CERTIFICATE----- *certificate value in terms of* string of characters specified *between the lines* -----END CERTIFICATE-----	Specify the CA certificate value between the `BEGIN` and `END` lines.

A

Version upgrade guide

This appendix provides a reference guide for upgrading FUJITSU Enterprise Postgres 11 Advanced Edition on IBM LinuxONE to FUJITSU Enterprise Postgres 12 Advanced Edition. We show the steps to use the **`pg_upgrade`** command to perform a version upgrade.

This appendix covers the following topics:

- Overview of pg_upgrade for major version upgrades
- Using pg_upgrade

Overview of pg_upgrade for major version upgrades

The **`pg_upgrade`** (formerly **`pg_migrator`**) utility was introduced to simplify the major version upgrade process. Major version upgrades can get complex because of the various new features and associated system tables changes. The **`pg_upgrade`** utility performs version upgrades without needing to dump and restore data between the existing database version to the new database version because dump and restore are handled internally by **`pg_upgrade`**. Figure A-1 shows a high-level internal flow of **`pg_upgrade`**.

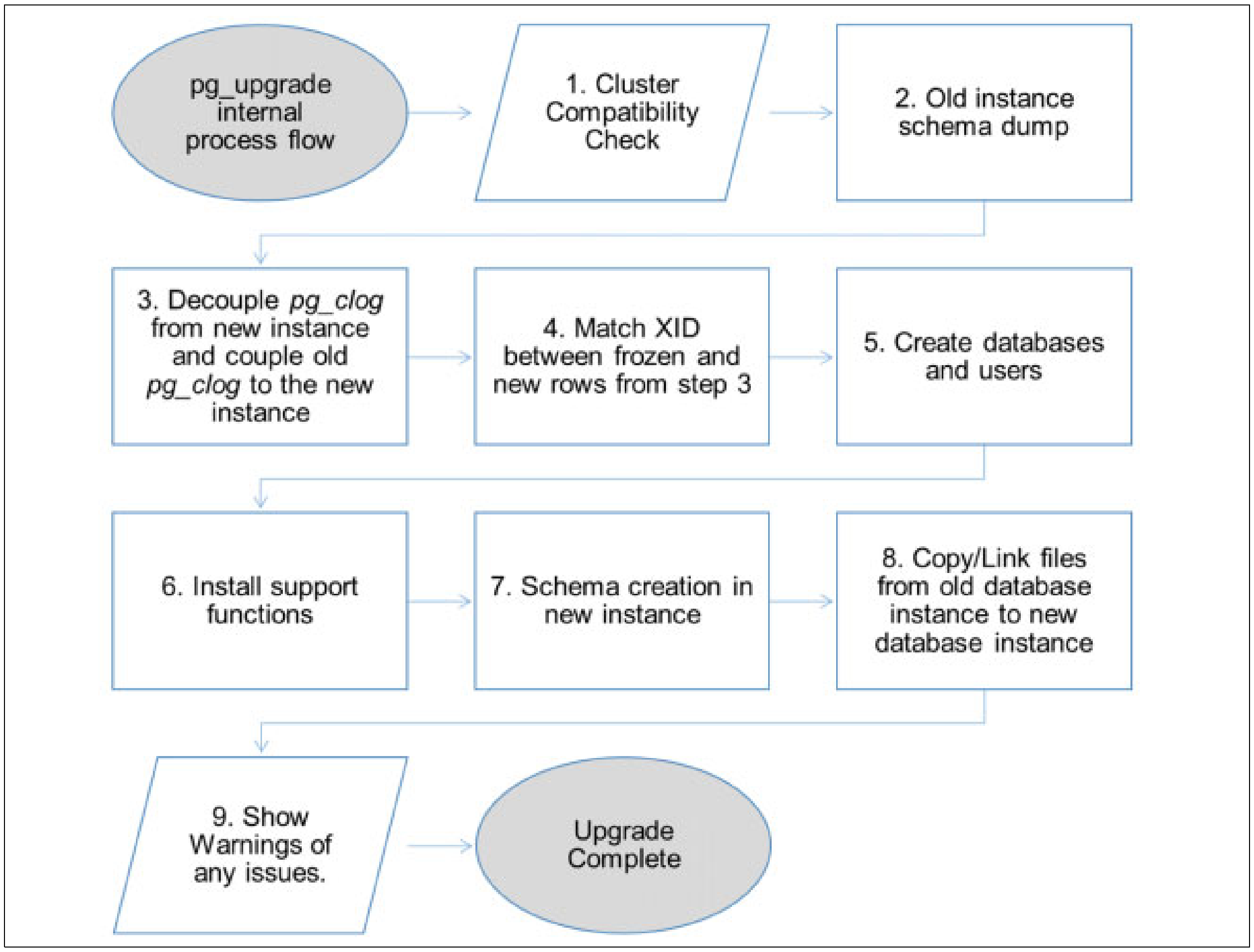

pg_upgrade internal process flow
1. Cluster Compatibility Check
2. Old instance schema dump
3. Decouple *pg_clog* from new instance and couple old *pg_clog* to the new instance
4. Match XID between frozen and new rows from step 3
5. Create databases and users
6. Install support functions
7. Schema creation in new instance
8. Copy/Link files from old database instance to new database instance
9. Show Warnings of any issues.
Upgrade Complete

Figure A-1 The pg_upgrade high-level process flow

Using pg_upgrade

In this section, we show how to use pg_upgrade to update FUJITSU Enterprise Postgres 11 to FUJITSU Enterprise Postgres 12 on a SUSE Linux Enterprise Server 12 SP4 operating system (OS). We perform the following steps on virtual machine (VM) RDBKPGS1 on logical partition (LPAR) ARIES32 on IBM LinuxONE Machine A:

1. Set up FUJITSU Enterprise Postgres 11.
2. Populate the sample application database that is represented by the **`pgbench`** database.
3. Set up FUJITSU Enterprise Postgres 12.

4. Perform the upgrade by using the pg_upgrade utility.
5. Drop FUJITSU Enterprise Postgres 11.

Note: The prerequisites for the steps that are shown in this chapter is that FUJITSU Enterprise Postgres 11 and 12 are already installed.

Setting up FUJITSU Enterprise Postgres 11

In this section, we prepare the system to upgrade FUJITSU Enterprise Postgres 11 Advanced Edition on IBM LinuxONE to FUJITSU Enterprise Postgres 12 Advanced Edition by completing the following steps:

1. Configure the environment variables on the server, as shown in Example A-1.

Example: A-1 Configuring the environment variables

```
# su - fsepuser
$ PATH=/opt/fsepv11server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv11server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv11server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

2. Create an instance as shown in Example A-2. Specify the transaction log storage destination and the locale setting option as required.

Example: A-2 Creating an instance

```
$ initdb -D /database/inst1 --waldir=/transaction/inst1 --lc-collate="C"
--lc-ctype="C" --encoding=UTF8
The files belonging to this database system will be owned by user "fsepuser".
This user must also own the server process.
The database cluster will be initialized with locales
  COLLATE:  C
  CTYPE:    C
  MESSAGES: en_US.UTF-8
  MONETARY: en_US.UTF-8
  NUMERIC:  en_US.UTF-8
  TIME:     en_US.UTF-8
The default text search configuration will be set to "english". (15541)

Data page checksums are disabled. (18153)
fixing permissions on existing directory /database/inst1 ... ok (15516)
fixing permissions on existing directory /transaction/inst1 ... ok (15516)
creating subdirectories ... ok (15516)
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default timezone ...  (19842)America/New_York
selecting dynamic shared memory implementation ... posix
creating configuration files ... ok (15516)
running bootstrap script ... ok (15516)
performing post-bootstrap initialization ... ok (15516)
syncing data to disk ... ok (15516)

WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
```

```
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    pg_ctl -D /database/inst1 -l logfile start
```

3. Set up the `postgresql.conf` configuration file, as shown in Example A-3.

Example: A-3 Configuration file

```
$ vim /database/inst1/postgresql.conf
listen_addresses = '*'
port = 27500
logging_collector = on
backup_destination = '/backup/inst1'
archive_mode = on
archive_command = '/opt/fsepv12server64/bin/pgx_walcopy.cmd "%p"
"/backup/inst1/archived_wal/%f"'
```

4. Set up `pg_hba.conf`, which is the client authentication configuration file. Change xx.xx.xx.0 in Example A-4 to the IP address range of your environment.

Example: A-4 Configuration file

```
$ vim /database/inst1/pg_hba.conf
host    all             all             xxx.xx.xx.0/24     trust
```

5. Start the FUJITSU Enterprise Postgres 11 database instance, as shown in Example A-5.

Example: A-5 Starting the database instance

```
$ pg_ctl start -D /database/inst1
waiting for server to start....2021-01-06 01:39:15.579 EST [24177] LOG:  listening
on IPv4 address "0.0.0.0", port 27500
2021-01-06 01:39:15.579 EST [24177] LOG:  listening on IPv6 address "::", port
27500
2021-01-06 01:39:15.580 EST [24177] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2021-01-06 01:39:15.585 EST [24177] LOG:  redirecting log output to logging
collector process
2021-01-06 01:39:15.585 EST [24177] HINT:  Future log output will appear in
directory "log".
 done
server started
```

6. Verify the connectivity by using the command that is shown in Example A-6.

Example: A-6 Verifying the connectivity

```
$ psql -h xxx.xx.xx.162 -p 27500 -d postgres -U fsepuser
psql (11.6)
Type "help" for help.

postgres=#
```

We have now set up the FUJITSU Enterprise Postgres 11 database instance.

Preparing FUJITSU Enterprise Postgres 11 for an upgrade

In this section, we prepare a FUJITSU Enterprise Postgres 11 installation for upgrading to a FUJITSU Enterprise Postgres 12 installation by completing the following steps:

1. To initialize the sample **pgbench** database, run **pgbench**, as shown in Example A-7.

Example: A-7 Initializing the sample pgbench database

```
$ pgbench -i postgres
dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping (11929)
NOTICE:  table "pgbench_branches" does not exist, skipping (11929)
NOTICE:  table "pgbench_history" does not exist, skipping (11929)
NOTICE:  table "pgbench_tellers" does not exist, skipping (11929)
creating tables...
generating data...
100000 of 100000 tuples (100%) done (elapsed 0.06 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done.
```

2. Back up the FUJITSU Enterprise Postgres 11 database instance by using the following command:

   ```
   $ pg_dumpall > fep11_postgres.sql
   ```

3. Drop the extension `pg_stat_statements`. The command along with the results is shown in Example A-8.

Example: A-8 Dropping the extension

```
$ psql -h xxx.xx.xx.162 -p 27500 -d postgres -U fsepuser -c "DROP EXTENSION
pg_stat_statements"
DROP EXTENSION
```

Note: If you have set up the following extensions, you must drop them from all databases other than the template0 database:

- `pg_stat_statements`
- `oracle_compatible`
- `pg_dbms_stats`
- `pg_hint_plan`

4. Stop the database instance (Example A-9).

Example: A-9 Stopping the instance

```
$ pg_ctl stop -D /database/inst1
waiting for server to shut down.... done
server stopped
```

5. Move the instance, Write-Ahead-Log (WAL), and backup directories by using the commands that are shown in Example A-10.

Example: A-10 Moving the instance

```
$ mv /database/inst1/ /database/inst1.fep11
$ mv /transaction/inst1/ /transaction/inst1.fep11
$ mv /backup/inst1/ /backup/inst11.fep11
```

6. Update the symbolic links by using the following command:

```
$ ln -nfs /transaction/inst1.fep11 /database/inst1.fep11/pg_wal
```

7. Update the `postgresql.conf` file, as shown in Example A-11.

Example: A-11 Updating the configuration file

```
$ vim /database/inst1/postgresql.conf
port = 27500
backup_destination = '/backup/inst1.fep11'
archive_command = '/opt/fsepv12server64/bin/pgx_walcopy.cmd "%p"
"/backup/inst1.fep11/archived_wal/%f"'
```

Setting up FUJITSU Enterprise Postgres 12

In this section, we provide the steps to set up FUJITSU Enterprise Postgres 12:

1. Prepare the directories, as shown in Example A-12.

Example: A-12 Preparing directories

```
# mkdir -p /database/inst1
# chown -R fsepuser:fsepuser /database
# chmod 700 /database/inst1
# mkdir -p /transaction/inst1
# chown -R fsepuser:fsepuser /transaction
# chmod 700 /transaction/inst1
# mkdir -p /backup/inst1
# chown -R fsepuser:fsepuser /backup
# chmod 700 /backup/inst1
```

2. Configure the environment variables, as shown in Example A-13.

Example: A-13 Configuring environment variables

```
# su - fsepuser
$ PATH=/opt/fsepv12server64/bin:$PATH ; export PATH
$ MANPATH=/opt/fsepv12server64/share/man:$MANPATH ; export MANPATH
$ LD_LIBRARY_PATH=/opt/fsepv12server64/lib:$LD_LIBRARY_PATH ; export
LD_LIBRARY_PATH
```

3. Create the database instance, as shown in Example A-14.

Example: A-14 Creating the database instance

```
$ initdb -D /database/inst1 --waldir=/transaction/inst1 --lc-collate="C"
--lc-ctype="C" --encoding=UTF8
The files belonging to this database system will be owned by user "fsepuser".
This user must also own the server process.
```

```
The database cluster will be initialized with locales
  COLLATE:  C
  CTYPE:    C
  MESSAGES: en_US.UTF-8
  MONETARY: en_US.UTF-8
  NUMERIC:  en_US.UTF-8
  TIME:     en_US.UTF-8
The default text search configuration will be set to "english". (15541)

Data page checksums are disabled. (18153)

fixing permissions on existing directory /database/inst1 ... ok (15516)
fixing permissions on existing directory /transaction/inst1 ... ok (15516)
creating subdirectories ... ok (15516)
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ...  (19842)America/New_York
creating configuration files ... ok (15516)
running bootstrap script ... ok (15516)
performing post-bootstrap initialization ... ok (15516)
syncing data to disk ... ok (15516)

initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    pg_ctl -D /database/inst1 -l logfile start
```

4. Modify the `postgresql.conf` configuration file, as shown in Example A-15.

Example: A-15 Modifying the configuration file

```
$ vim /database/inst1/postgresql.conf
listen_addresses = '*'
port = 27500
logging_collector = on
backup_destination = '/backup/inst1'
archive_mode = on
archive_command = '/opt/fsepv12server64/bin/pgx_walcopy.cmd "%p"
"/backup/inst1/archived_wal/%f"'
```

5. Set up `pg_hba.conf`, which is the client authentication configuration file, for connectivity. Update the xx.xx.xx.0 with the IP address range of your environment. Also, use authentication that is more appropriate than the `trust` authentication that is used in Example A-16.

Example: A-16 Client authentication configuration edit

```
$ vim /database/inst1/pg_hba.conf
host    all             all             xxx.xx.xx.0/24     trust
```

6. Start the database instance, as shown in Example A-17.

Example: A-17 Starting the database instance

```
$ pg_ctl start -D /database/inst1
waiting for server to start....2021-01-06 02:36:12.413 EST [25135] LOG:  starting
PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (SUSE Linux) 4.8.5, 64-bit
2021-01-06 02:36:12.413 EST [25135] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2021-01-06 02:36:12.413 EST [25135] LOG:  listening on IPv6 address "::", port
27500
2021-01-06 02:36:12.414 EST [25135] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2021-01-06 02:36:12.421 EST [25135] LOG:  redirecting log output to logging
collector process
2021-01-06 02:36:12.421 EST [25135] HINT:  Future log output will appear in
directory "log".
 done
server started
```

7. Verify the connectivity to the FUJITSU Enterprise Postgres 12 database instance, as shown in Example A-18.

Example: A-18 Connectivity verification

```
$ psql -h xxx.xx.xx.162 -p 27500 -d postgres -U fsepuser
psql (12.1)
Type "help" for help.

postgres=#
```

8. Stop the database instance, as shown in Example A-19.

Example: A-19 Stopping the database instance

```
$ pg_ctl stop -D /database/inst1
waiting for server to shut down.... done
server stopped
```

Performing the upgrade

In this section, we list the steps that are required to perform the upgrade by using **pg_upgrade**:

1. Both the instances of FUJITSU Enterprise Postgres 11 and 12 should now be in a shutdown state. Check the compatibility of the databases instances for an upgrade, as show in Example A-20.

Example: A-20 Checking compatibility

```
$ pg_upgrade --old-datadir "/database/inst1.fep11" --new-datadir "/database/inst1"
--old-bindir "/opt/fsepv11server64/bin" --new-bindir "/opt/fsepv12server64/bin"
--old-port=27500 --new-port=27500 --user=fsepuser --check
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   (19587)ok (19491)
Checking database user is the install user                  (19587)ok (19491)
Checking database connection settings                       (19587)ok (19491)
Checking for prepared transactions                          (19587)ok (19491)
Checking for reg* data types in user tables                  (19587)ok (19491)
Checking for contrib/isn with bigint-passing mismatch        (19587)ok (19491)
Checking for tables WITH OIDS                                (19587)ok (19491)
Checking for invalid "sql_identifier" user columns           (19587)ok (19491)
Checking for presence of required libraries                  (19587)ok (19491)
Checking database user is the install user                   (19587)ok (19491)
Checking for prepared transactions                           (19587)ok (19491)

*Clusters are compatible*
```

The two database instances of FUJITSU Enterprise Postgres 11 and 12 are compatible for an upgrade.

Note: For more information about the **pg_upgrade** command utility, see pg_upgrade.

2. Perform the upgrade by using **pg_upgrade**, as shown in Example A-21.

Example: A-21 Performing the upgrade

```
$ pg_upgrade --old-datadir "/database/inst1.fep11" --new-datadir "/database/inst1"
--old-bindir "/opt/fsepv11server64/bin" --new-bindir "/opt/fsepv12server64/bin"
--old-port=27500 --new-port=27500 --user=fsepuser
Performing Consistency Checks
-----------------------------
Checking cluster versions                                    (19587)ok (19491)
Checking database user is the install user                   (19587)ok (19491)
Checking database connection settings                        (19587)ok (19491)
Checking for prepared transactions                           (19587)ok (19491)
Checking for reg* data types in user tables                  (19587)ok (19491)
Checking for contrib/isn with bigint-passing mismatch        (19587)ok (19491)
Checking for tables WITH OIDS                                (19587)ok (19491)
Checking for invalid "sql_identifier" user columns           (19587)ok (19491)
Creating dump of global objects                              (19587)ok (19491)
Creating dump of database schemas
                                                             (19587)ok (19491)
Checking for presence of required libraries                  (19587)ok (19491)
Checking database user is the install user                   (19587)ok (19491)
Checking for prepared transactions                           (19587)ok (19491)

If pg_upgrade fails after this point, you must re-initdb the
new cluster before continuing.

Performing Upgrade
------------------
Analyzing all rows in the new cluster                        (19587)ok (19491)
Freezing all rows in the new cluster                         (19587)ok (19491)
Deleting files from new pg_xact                              (19587)ok (19491)
Copying old pg_xact to new server                            (19587)ok (19491)
Setting next transaction ID and epoch for new cluster        (19587)ok (19491)
Deleting files from new pg_multixact/offsets                 (19587)ok (19491)
Copying old pg_multixact/offsets to new server               (19587)ok (19491)
Deleting files from new pg_multixact/members                 (19587)ok (19491)
Copying old pg_multixact/members to new server               (19587)ok (19491)
Setting next multixact ID and offset for new cluster         (19587)ok (19491)
Resetting WAL archives                                       (19587)ok (19491)
```

```
Setting frozenxid and minmxid counters in new cluster       (19587)ok (19491)
Restoring global objects in the new cluster                 (19587)ok (19491)
Restoring database schemas in the new cluster
                                                            (19587)ok (19491)
Copying user relation files (19567)
                                                            (19587)ok (19491)
Setting next OID for new cluster                            (19587)ok (19491)
Sync data directory to disk                                 (19587)ok (19491)
Creating script to analyze new cluster                      (19587)ok (19491)
Creating script to delete old cluster                       (19587)ok (19491)

Upgrade Complete
----------------
Optimizer statistics are not transferred by pg_upgrade so,
once you start the new server, consider running:
    ./analyze_new_cluster.sh

Running this script will delete the old cluster's data files:
    ./delete_old_cluster.sh
```

The upgrade is marked as complete.

3. Start the database instance, as shown in Example A-22.

Example: A-22 Starting the database

```
$ pg_ctl start -D /database/inst1
waiting for server to start....2021-01-06 02:52:15.941 EST [25844] LOG:  starting
PostgreSQL 12.1 on s390x-ibm-linux-gnu, compiled by gcc (SUSE Linux) 4.8.5, 64-bit
2021-01-06 02:52:15.941 EST [25844] LOG:  listening on IPv4 address "0.0.0.0",
port 27500
2021-01-06 02:52:15.941 EST [25844] LOG:  listening on IPv6 address "::", port
27500
2021-01-06 02:52:15.943 EST [25844] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.27500"
2021-01-06 02:52:15.949 EST [25844] LOG:  redirecting log output to logging
collector process
2021-01-06 02:52:15.949 EST [25844] HINT:  Future log output will appear in
directory "log".
 done
server started
```

4. Check that the **pgbench** tables that were in FUJITSU Enterprise Postgres 11 before the upgrade to ensure that they are now available in FUJITSU Enterprise Postgres 12 after the upgrade, as shown in Example A-23.

Example: A-23 Upgrading the verification

```
$ psql -h xxx.xx.xx.162 -p 27500 -d postgres -U fsepuser
psql (12.1)
Type "help" for help.

postgres=# \d
              List of relations
 Schema |       Name       | Type  |  Owner
--------+------------------+-------+----------
 public | pgbench_accounts | table | fsepuser
 public | pgbench_branches | table | fsepuser
 public | pgbench_history  | table | fsepuser
 public | pgbench_tellers  | table | fsepuser
(Four rows)
```

5. Re-create the `pg_stat_statements` extension that was dropped before the upgrade process (Example A-24).

Example: A-24 Re-creating the extension

```
$ psql -h xxx.xx.xx.162 -p 27500 -d postgres -U fsepuser -c "CREATE EXTENSION
pg_stat_statements"
CREATE EXTENSION
```

6. Analyze the upgraded database instance by using the **`analyze_new_cluster.sh`** script to generate minimal statistics and then gather the statistics, as shown in Example A-25.

Example: A-25 Analyzing the instance

```
$ ./analyze_new_cluster.sh
This script will generate minimal optimizer statistics rapidly
so your system is usable, and then gather statistics twice more
with increasing accuracy.  When it is done, your system will
have the default level of optimizer statistics.

If you have used ALTER TABLE to modify the statistics target for
any tables, you might want to remove them and restore them after
running this script because they will delay fast statistics generation.

If you would like default statistics as quickly as possible, cancel
this script and run:
    "/opt/fsepv12server64/bin/vacuumdb" -U fsepuser --all --analyze-only

vacuumdb: processing database "postgres": Generating minimal optimizer statistics
(1 target)
vacuumdb: processing database "template1": Generating minimal optimizer statistics
(1 target)
vacuumdb: processing database "postgres": Generating medium optimizer statistics
(10 targets)
vacuumdb: processing database "template1": Generating medium optimizer statistics
(10 targets)
vacuumdb: processing database "postgres": Generating default (full) optimizer
statistics
vacuumdb: processing database "template1": Generating default (full) optimizer
statistics

Done
```

Dropping FUJITSU Enterprise Postgres 11

Example A-26 shows the commands that are used to clear the obsolete FUJITSU Enterprise Postgres 11 directories.

Example: A-26 Deleting directories

```
$ rm -rf /database/inst1.fep11/
$ rm -rf /transaction/inst1.fep11/
$ rm -rf /backup/inst1.fep11/
```

Sizing guide

This appendix presents the sizing guide for FUJITSU Enterprise Postgres on IBM LinuxONE. This appendix provides the formulas for sizing FUJITSU Enterprise Postgres Server components.

This appendix covers the following topics:

- General recommendations
- Sizing a database server
- Proof of concept architecture
- FUJITSU Enterprise Postgres benchmark on IBM LinuxONE white paper

General recommendations

Sizing is a recursive process. The goal of sizing is to eventually have an optimally performing FUJITSU Enterprise Postgres Database environment. Sizing can be classified into two categories:

- Sizing the hardware
- Sizing the database software

Both categories are interdependent, each with different factors that determine sizing. For example, database sizing depends on factors like application workload, operations, forecasted growth, and housekeeping, and hardware sizing depends on factors like back-end and front-end network throughput, replication, resilience, and availability requirements.

Database sizes typically always grow, and for various reasons. These reasons might be of business origin, poorly implemented database management practices, or bad database design. Irrespective of the reasons, it is important to correctly size the database, planning for growth during implementation because unplanned and unmanaged database growth leads to application performance issues.

When sizing the compute resources for FUJITSU Enterprise Postgres, there are a few general guidelines to consider:

- Always try to use scalable hardware to cater for likely future requirements, such as a need to add more users or database functions from time to time.
- The number of Integrated Facility for Linux (IFL) processors (IBM LinuxONE CPUs), memory, and usable storage that is allocated should be larger than the initial installation requirements to cater for planned growth and provide a buffer against unplanned growth.
- Check for database features, peripheral tools, and operating system (OS) incompatibilities that might affect performance before you do the implementation. Contact your customer service representatives if you require assistance from IBM and Fujitsu with validating any third-party software compatibility.
- Database performance is defined by four parameters:
  - CPU performance and number of CPUs
  - Memory size
  - Disk I/O
  - Client/server connection speed.

  Contact your customer service representatives if you require assistance from IBM and Fujitsu with sizing hardware or database software sizing.

Sizing a database server

Database sizing depends on the application workloads and the system from which it is migrating. If you are planning to migrate from x86 to IBM LinuxONE, contact your customer service representatives for assistance from IBM and Fujitsu with sizing the hardware and database software for best fit with your requirements. For more information, see *Practical Migration from x86 to LinuxONE*, SG24-8377.

Memory

When more memory is allocated for the database, disk I/O is reduced. A simple guideline is to reserve memory for a database per the following formula:

If you have a dedicated database server with 1 GB or more of RAM, a reasonable starting value for **`shared_buffers`** is 25% of the memory in your system. There are some workloads where even larger settings for **`shared_buffers`** are effective, but because PostgreSQL also relies on the OS cache, it is unlikely that an allocation of more than 40% of RAM to **`shared_buffers`** works better than a smaller amount.

For example, for a server with 32 GB installed memory, 24 GB is for the OS, and 8 GB is for database server operations.

Note: For FUJITSU Enterprise Postgres, a minimum of 512 MB of memory is required. For more information about resizing memory, see Appendix F, "Estimating Memory Requirements" of the *FUJITSU Enterprise Postgres 12: Installation and Setup Guide for Server*.

Disks

When sizing storage capacity, consider the following database components:

- Database backups: Physical, logical, and incremental backups
- Transaction log files
- Copies of database dumps for production, and rebasing for test and development environments

Figure B-1 shows three different storage areas that should be considered when planning a database installation on a server.

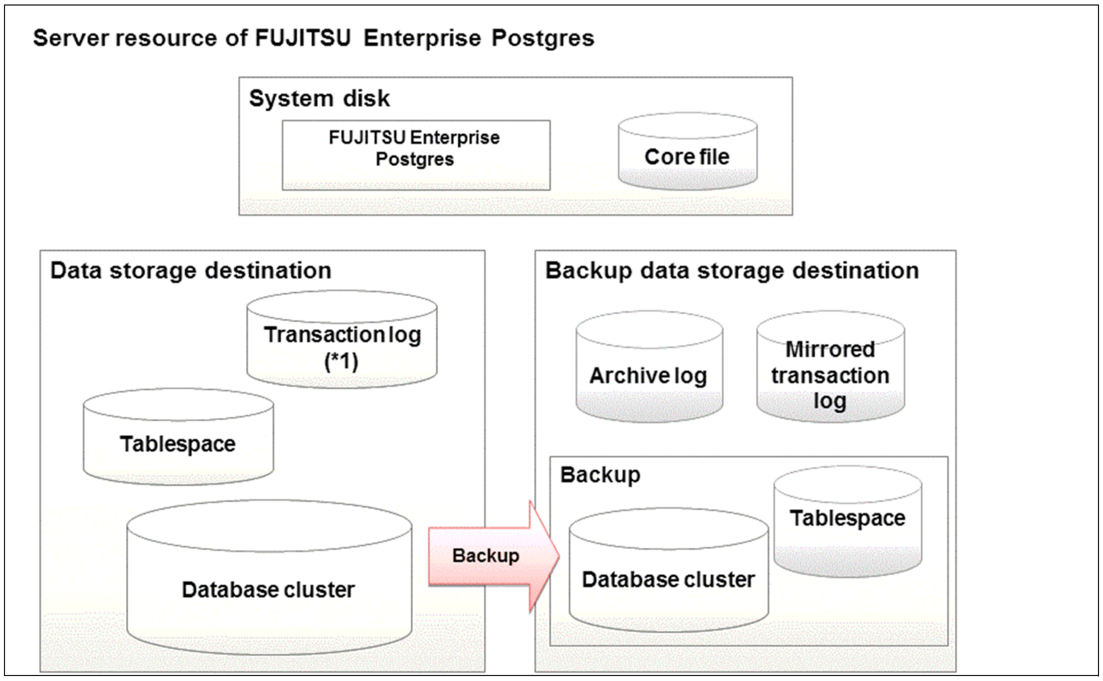


Figure B-1 FUJITSU Enterprise Postgres server resources

- The disk storage for the OS and database should be separate.
- For distributing the I/O, the transaction log and data storage destinations should be placed on different storage disks.
- The table space and the database cluster should be placed into separate storage areas.
- For backup disk sizing, 2x the database cluster should be added to the transaction log space requirement and archive log space requirement.

Note: Backup disk space requirements = size of the database cluster x 2 + transaction log space requirements + archive log space requirements.

- The Vertical Clustered Index (VCI) disk size can be estimated by using the following formula:

  Disk space = (number of rows in tables) x (number of bytes per row) x (compression ratio) + (Write Optimized Store (WOS) size)

  Number of bytes per row = (19 + (number of columns that are specified in CREATE INDEX) / 8 + (number of bytes per single column value)) x 1.1

  WOS size = (number of WOS rows) / 185 x 8096

Note: For more information about estimating disk space requirements, see Appendix E. “Estimating Database Disk Space Requirements” in the *FUJITSU Enterprise Postgres 12 Installation and Setup Guide for Server.*

Sizing is a continuous process. As the database activity increases, it is likely that the database size also increases. For more information about sizing FUJITSU Enterprise Postgres on IBM LinuxONE, contact your IBM and FUJITSU representatives.

Proof of concept architecture

In this section, we provide details about the hardware architecture that you might want to use to test your applications with FUJITSU Enterprise Postgres on IBM z15™ / IBM LinuxONE. This configuration is provided as a guide only, and it is based on the configuration that used by the IBM and Fujitsu teams for performance benchmarking use cases. Contact your customer service representatives for assistance from IBM and FUJITSU with sizing the hardware and database software environments for best fit with your requirements.

Figure B-2 on page 447 shows the IBM z15, storage, and network configuration that are used for a proof of concept (PoC) of FUJITSU Enterprise Postgres.

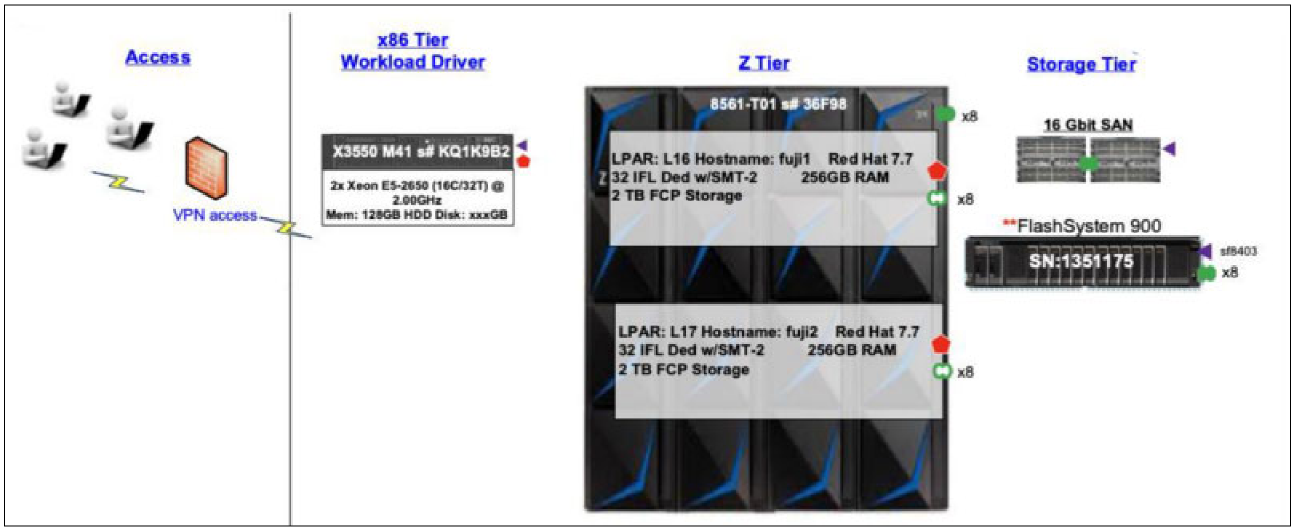

Access
VPN access
x86 Tier
Workload Driver
X3550 M41 s# KQ1K9B2
2x Xeon E5-2650 (16C/32T) @ 2.00GHz
Mem: 128GB HDD Disk: xxxGB
Z Tier
8561-T01 s# 36F98
LPAR: L16 Hostname: fuji1 Red Hat 7.7
32 IFL Ded w/SMT-2 256GB RAM
2 TB FCP Storage
LPAR: L17 Hostname: fuji2 Red Hat 7.7
32 IFL Ded w/SMT-2 256GB RAM
2 TB FCP Storage
x8
Storage Tier
16 Gbit SAN
**FlashSystem 900
SN:1351175
sf8403

Figure B-2 Proof of concept for a hardware configuration for FUJITSU Enterprise Postgres on IBM z15 / IBM LinuxONE

In this configuration, two logical partitions (LPARs) are shown as configured with 32 IFL processors with symmetric multi-threading and 256 GB of real memory (RAM). The two LPARs were used for three benchmarks:

- Online Transaction Processing (OLTP) workload
- Hybrid Transactional Analytical Processing (HTAP) benchmark with VCI
- FUJITSU High-Speed Data Load

The OLTP workload is tested with 128 – 512 clients. The HTAP workload is tested by overlapping the OLTP workload with Online Analytical Processing (OLAP) queries. The OLAP queries performance is further enhanced by creating a VCI on candidate OLAP queries. In both workloads, a fixed size database (120 GB) is used.

Storage systems

In this section, we describe various storage systems and their capacities.

IBM FlashSystem 900

The IBM FlashSystem 900 has the following specifications:

- Serial #: 1351175
- MTM: 9840-AE2
- Flash modules: Twelve 5.2 TiB modules
- Capacity: 52 TiB
- Usable RAID 5 switches: Eight 16 Gb FC ports

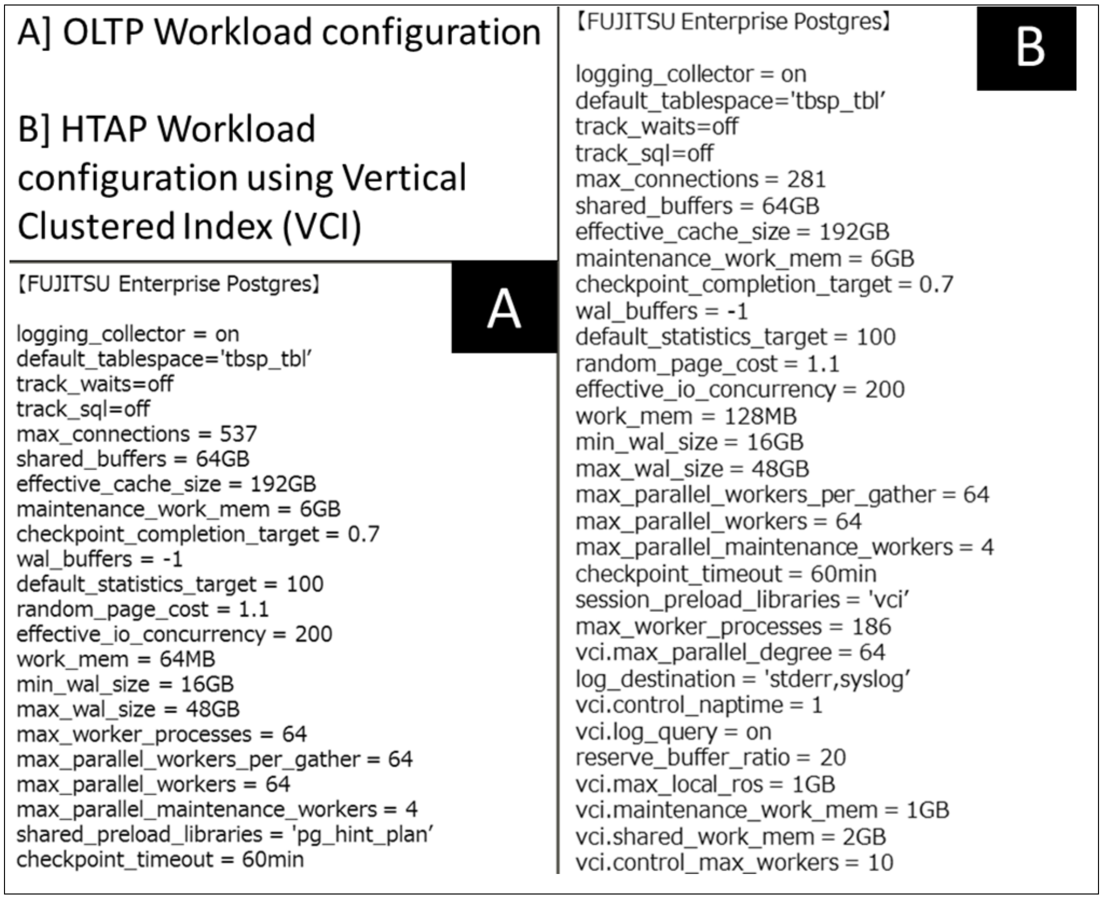
Table B-1 IBM FlashSystem 900 specifications

Name	Capacity (GiB)	SCSI ID	Paths
PELCP01_L16_boot	50	0	8
PELCP01_L16_db_tablespace	600	1	8
PELCP01_L16_trans_logs	100	2	8

Name	Capacity (GiB)	SCSI ID	Paths
PELCP01_L16_db	100	3	8
PELCP01_L16_default	1800	4	8

FUJITSU Enterprise Postgres configuration

Figure B-3 provides sample `postgresql.conf` settings for OLTP and OLAP with VCI workloads.



A] OLTP Workload configuration

B] HTAP Workload configuration using Vertical Clustered Index (VCI)

A

```
【FUJITSU Enterprise Postgres】

logging_collector = on
default_tablespace='tbsp_tbl'
track_waits=off
track_sql=off
max_connections = 537
shared_buffers = 64GB
effective_cache_size = 192GB
maintenance_work_mem = 6GB
checkpoint_completion_target = 0.7
wal_buffers = -1
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
work_mem = 64MB
min_wal_size = 16GB
max_wal_size = 48GB
max_worker_processes = 64
max_parallel_workers_per_gather = 64
max_parallel_workers = 64
max_parallel_maintenance_workers = 4
shared_preload_libraries = 'pg_hint_plan'
checkpoint_timeout = 60min
```

B

```
【FUJITSU Enterprise Postgres】

logging_collector = on
default_tablespace='tbsp_tbl'
track_waits=off
track_sql=off
max_connections = 281
shared_buffers = 64GB
effective_cache_size = 192GB
maintenance_work_mem = 6GB
checkpoint_completion_target = 0.7
wal_buffers = -1
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
work_mem = 128MB
min_wal_size = 16GB
max_wal_size = 48GB
max_parallel_workers_per_gather = 64
max_parallel_workers = 64
max_parallel_maintenance_workers = 4
checkpoint_timeout = 60min
session_preload_libraries = 'vci'
max_worker_processes = 186
vci.max_parallel_degree = 64
log_destination = 'stderr,syslog'
vci.control_naptime = 1
vci.log_query = on
reserve_buffer_ratio = 20
vci.max_local_ros = 1GB
vci.maintenance_work_mem = 1GB
vci.shared_work_mem = 2GB
vci.control_max_workers = 10
```

Figure B-3 Sample FUJITSU postgresql.conf for OLTP and OLAP workloads that use VCI

FUJITSU Enterprise Postgres benchmark on IBM LinuxONE white paper

Sizing and performance go together. The purpose of sizing is to achieve optimal performance. Sizing requirements and performance goals depend on the application workload and business requirements. Business requirements vary with user demand volatility, so a robust system should be sized in a way that it meets the business requirements and performance goals, and it is scalable to cater to fluctuations or growth in user demand.

IBM and FUJITSU are continuously working together to create high-performant Enterprise Postgres on IBM LinuxONE. Together, IBM and FUJITSU conducted benchmarks, which are described in *FUJITSU Enterprise Postgres on IBM LinuxONE: Improving Your Organization's Data Performance*.

Patching guide

This appendix provides the steps to apply patches to FUJITSU Enterprise Postgres Server on IBM LinuxONE.

Note: This appendix assumes that the user has the patch. For more information about patches, contact your customer service representatives.

This appendix covers the following topic:

- Package operations by using a downloaded RPM

Package operations by using a downloaded RPM

This section describes how to update a package by using the rpm command and a downloaded RPM:

1. Check for preupdated packages by using the command that is shown in Example C-1.

Example: C-1 Command to check for preupdated packages with results

```
# rpm -qa | grep FJSVfsep-SV-12
FJSVfsep-SV-12-1200-0.s390x
```

2. Update the package by using the downloaded RPM. Use the commands that are shown in Example C-2.

Example: C-2 Updating the package by using the downloaded RPM

```
// Checking the rpm File Before and After the Update
# ls /rpm
FJSVfsep-SV-12-1200-0.s390x.rpm  FJSVfsep-SV-12-1200-1.s390x.rpm
// Update the package
# rpm -U FJSVfsep-SV-12-1200-1.s390x.rpm
```

Note: Save the RPM file of the currently installed package as a backup.

3. Check for updated packages by using the command that is shown in Example C-3.

Example: C-3 Checking for updated packages

```
# rpm -qa | grep FJSVfsep-SV-12
FJSVfsep-SV-12-1200-1.s390x
```

4. If you need to back out the package to the original RPM or a previous version, use the commands that are shown in Example C-4.

Example: C-4 Backing out the package

```
// Check for pre-backout packages
# rpm -qa | grep FJSVfsep-SV-12
FJSVfsep-SV-12-1200-1.s390x
// Backout the package
# rpm -U --oldpackage FJSVfsep-SV-12-1200-0.s390x.rpm
// Check for backout packages
# rpm -qa | grep FJSVfsep-SV-12
FJSVfsep-SV-12-1200-0.s390x
```

Abbreviations and acronyms

ACP	access control point
AES	Advanced Encryption Standard
AKS	auto-open keystore
AP	adjunct processor
APQN	adjunct processor queue number
BPR	Bulk Power Regulator
CA	certificate authority
CaaS	container-as-a-service
CBU	Capacity Backup
CCA	Common Cryptographic Architecture
CCPA	California Consumer Privacy Act
CHPID	Channel Path Identifier
CIO	chief information officer
CLI	command-line interface
CN	Common Name
CoD	Capacity on Demand
CP	control program
CPACF	Central Processor Assist for Cryptographic Function
CPC	central processor complex
CR	Custom Resource
CRD	Custom Resource Definition
CSI	Container Storage Interface
CTC	channel-to-channel
CTO	chief technology officer
CUI	command-line user interface
DASD	direct-access storage device
DEK	Data Encryption Key
DIY	do-it-yourself
DML	data manipulation language
DR	disaster recovery
EAV	extended address volume
EP11	Enterprise PKCS #11
ETL	Extract, Transform, and Load
EAL	Evaluation Assurance Level
FAST	FUJITSU Australia Software Technology
FCP	Fibre Channel Protocol
GDPR	General Data Protection Regulation
GMC	Global Meta Cache
GSSAPI	Generic Security Standard application programming interface
GUI	graphical user interface
HA	high availability of highly available
HSM	hardware security module
HTAP	Hybrid Transactional Analytical Processing
IBM	International Business Machines Corporation
IFL	Integrated Facility for Linux
IMCS	in-memory columnar storage
ISFC	Inter-System Facility for Communications
ISV	independent software vendor
JiT	just-in-time
KDC	Key Distribution Center
KEK	key-encrypting-key
LCSS	logical channel subsystem
LGR	live guest relocation
LPAR	logical partition
LUN	logical unit
MBTF	mean time between failures
MC	Mirroring Controller
MEK	Master Encryption Key
ODBC	Open Database Connectivity
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
OS	operating system
OSA	Open System Adapter
OSS	open-source software
PaaS	platform-as-a-service
PCHID	physical channel identifier
PDR	Persistent Data Record
PiT	point-in-time
PITR	point-in-time recovery
PoC	proof of concept
PVC	Persistent Volume Claim
RAIM	Redundant Array of Independent Memory
RDBMS	Relational Database Management System
RHCOS	Red Hat Enterprise Linux CoreOS

RHOCP	Red Hat OpenShift Container Platform
ROI	return on investment
ROS	Read Optimized Store
RPO	recovery point objective
RTO	recovery time objective
SaaS	software-as-a-service
SAN	storage area network
SCRAM	Salted Challenge Response Authentication Mechanism
SE	Support Element
SLA	service-level agreement
SSI	single system image
SSL	Secure Sockets Layer
SVC	SAN Volume Controller
TDE	Transparent Data Encryption
TGT	Ticket Granting Ticket
TKE	Trusted Key Entry
UDX	User-Defined Extension
UPI	user provisioned infrastructure
VCI	Vertical Clustered Index
VIPA	Virtual IP Address
VM	virtual machine
VPN	virtual private network
WAL	Write-Ahead-Log
WOS	Write Optimized Store
zEDC	zEnterprise Data Compression

Related publications

The publications that are listed in this section are considered suitable for a more detailed description of the topics that are covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topics in this document. Some publications that are referenced in this list might be available in softcopy only.

- *Practical Migration from x86 to LinuxONE*, SG24-8377
- *Scale up for Linux on LinuxONE*, REDP-5540

You can search for, view, download, or order these documents and other Redbooks, Redpapers, web docs, drafts, and additional materials at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- FUJITSU Enterprise Postgres

  https://www.postgresql.fastware.com/
- FUJITSU Enterprise Postgres Blogs

  https://www.postgresql.fastware.com/blog
- FUJITSU Enterprise Postgres Compare Versions

  https://www.postgresql.fastware.com/compare-postgresql-versions
- FUJITSU Enterprise Postgres Container and Operator

  https://www.postgresql.fastware.com/fep-operator-announcement
- FUJITSU Enterprise Postgres on IBM LinuxONE Overview

  https://www.postgresql.fastware.com/fujitsu-enterprise-postgres-on-ibm-linuxone
- FUJITSU Enterprise Postgres on IBM LinuxONE Resource Center

  https://www.postgresql.fastware.com/resources-fujitsu-enterprise-postgres-optimised-for-ibm-linuxone
- FUJITSU Enterprise Postgres Manuals

  https://www.postgresql.fastware.com/product-manuals
- FUJITSU Enterprise Postgres Overview

  https://www.postgresql.fastware.com/fep-overview
- FUJITSU Enterprise Postgres Resource Center

  https://www.postgresql.fastware.com/resource-center

- FUJITSU Enterprise Postgres Trial Versions

  https://www.postgresql.fastware.com/fujitsu-enterprise-postgres-trial-version
- FUJITSU Enterprise Postgres white papers

  https://www.postgresql.fastware.com/white-papers

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

Data Serving with FUJITSU Enterprise Postgres on IBM

SG24-8499-00

ISBN 0738459747

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

SG24-8499-00

ISBN 0738459747

Printed in U.S.A.