PERCONA

Distribution for PostgreSQL Documentation

15.5 (November 30, 2023)

Percona Technical Documentation Team

Table of contents

1. Percona Distribution for PostgreSQL 15 Documentation 3
2. Release Notes 5
   - 2.1 Percona Distribution for PostgreSQL release notes 5
   - 2.2 Percona Distribution for PostgreSQL 15.5 (2023-11-30) 6
   - 2.3 Percona Distribution for PostgreSQL 15.4 (2023-08-29) 9
   - 2.4 Percona Distribution for PostgreSQL 15.3 (2023-06-28) 12
   - 2.5 Percona Distribution for PostgreSQL 15.2 Update (2023-05-22) 14
   - 2.6 Percona Distribution for PostgreSQL 15.2 (2023-03-20) 15
   - 2.7 Percona Distribution for PostgreSQL 15.1 (2022-11-21) 17
   - 2.8 Percona Distribution for PostgreSQL 15.0 (2022-10-24) 19
3. Installation and Upgrade 22
   - 3.1 Install Percona Distribution for PostgreSQL 22
   - 3.2 Run Percona Distribution for PostgreSQL in a Docker container 41
   - 3.3 Migrate from PostgreSQL to Percona Distribution for PostgreSQL 45
   - 3.4 Upgrading Percona Distribution for PostgreSQL from 14 to 15 49
   - 3.5 Minor Upgrade of Percona Distribution for PostgreSQL 58
4. Extensions 60
   - 4.1 pg_stat_monitor 60
5. Solutions 67
   - 5.1 High availability 67
   - 5.2 Backup and disaster recovery 110
   - 5.3 Spatial data handling 125
   - 5.4 LDAP Authentication 136
6. Telemetry on Percona Distribution for PostgreSQL 137
   - 6.1 What information is collected 137
   - 6.2 Disable telemetry 137
7. Uninstalling Percona Distribution for PostgreSQL 139
8. Copyright and licensing information 141
   - 8.1 Documentation licensing 141
9. Trademark Policy 143

1. Percona Distribution for PostgreSQL 15 Documentation

Percona Distribution for PostgreSQL is a collection of tools to assist you in managing your PostgreSQL database system: it installs PostgreSQL and complements it by a selection of extensions that enable solving essential practical tasks efficiently:

- HAProxy - a high-availability and load-balancing solution
- Patroni is an HA (High Availability) solution for PostgreSQL.
- pgAudit provides detailed session or object audit logging via the standard PostgreSQL logging facility
- pgAudit set_user - The `set_user` part of `pgAudit` extension provides an additional layer of logging and control when unprivileged users must escalate themselves to superuser or object owner roles in order to perform needed maintenance tasks.
- pgBackRest is a backup and restore solution for PostgreSQL
- pgBadger - a fast PostgreSQL Log Analyzer.
- PgBouncer - a lightweight connection pooler for PostgreSQL
- pg_gather - an SQL script to assess the health of PostgreSQL cluster by gathering performance and configuration data from PostgreSQL databases.
- pgpool2 - a middleware between PostgreSQL server and client for high availability, connection pooling and load balancing.
- pg_repack rebuilds PostgreSQL database objects
- pg_stat_monitor collects and aggregates statistics for PostgreSQL and provides histogram information.
- PostGIS allows storing and manipulating spacial data in PostgreSQL.
- wal2json - a PostgreSQL logical decoding JSON output plugin.
- A collection of additional PostgreSQL contrib extensions

Get started **What's new**

See also

Percona Blog:

- pgBackRest - A Great Backup Solution and a Wonderful Year of Growth
- Securing PostgreSQL as an Enterprise-Grade Environment

Percona Distribution for PostgreSQL is also shipped with the libpq library. It contains "a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries."

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: August 29, 2023

Created: June 4, 2021

2. Release Notes

2.1 Percona Distribution for PostgreSQL release notes

- Percona Distribution for PostgreSQL 15.5 (2023-11-30)
- Percona Distribution for PostgreSQL 15.4 (2023-08-29)
- Percona Distribution for PostgreSQL 15.3 (2023-06-28)
- Percona Distribution for PostgreSQL 15.2 Update (2023-05-22)
- Percona Distribution for PostgreSQL 15.2 (2023-03-20)
- Percona Distribution for PostgreSQL 15.1 (2022-11-21)
- Percona Distribution for PostgreSQL 15 (2022-10-24)

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 30, 2023

Created: June 4, 2021

2.2 Percona Distribution for PostgreSQL 15.5 (2023-11-30)

Installation

Percona Distribution for PostgreSQL is a solution with the collection of tools from PostgreSQL community that are tested to work together and serve to assist you in deploying and managing PostgreSQL. The aim of Percona Distribution for PostgreSQL is to address the operational issues like High-Availability, Disaster Recovery, Security, Observability, Spatial data handling, Performance and Scalability and others that enterprises are facing.

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 15.5.

2.2.1 Release Highlights

- Docker images are now available for x86_64 architectures. They aim to simplify the developers' experience with the Distribution. Refer to the Docker guide for how to run Percona Distribution for PostgreSQL in Docker.
- Telemetry is now enabled in Percona Distribution for PostgreSQL to fill in the gaps in our understanding of how you use it and help us improve our products. Participation in the anonymous program is optional. You can opt-out if you prefer not to share this information. Find more information in the Telemetry on Percona Distribution for PostgreSQL document.
- The `percona-postgis33` and `percona-pgaudit` packages on YUM-based operating systems are renamed `percona-postgis33_15` and `percona-pgaudit15` respectively

The following is the list of extensions available in Percona Distribution for PostgreSQL.

Extension	Version	Description
HAProxy	2.8.3	a high-availability and load-balancing solution
Patroni	3.1.0	a HA (High Availability) solution for PostgreSQL
PgAudit	1.7.0	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
pgAudit set_user	4.0.1	provides an additional layer of logging and control when unprivileged users must escalate themselves to superusers or object owner roles in order to perform needed maintenance tasks.
pgBackRest	2.48	a backup and restore solution for PostgreSQL
pgBadger	12.2	a fast PostgreSQL Log Analyzer.
PgBouncer	1.21.0	a lightweight connection pooler for PostgreSQL
pg_gather	v23	an SQL script for running the diagnostics of the health of PostgreSQL cluster
pgpool2	4.4.4	a middleware between PostgreSQL server and client for high availability, connection pooling and load balancing.
pg_repack	1.4.8	rebuilds PostgreSQL database objects
pg_stat_monitor	2.0.3	collects and aggregates statistics for PostgreSQL and provides histogram information.
PostGIS	3.3.4	a spatial extension for PostgreSQL.
PostgreSQL Common	256	PostgreSQL database-cluster manager. It provides a structure under which multiple versions of PostgreSQL may be installed and/or multiple clusters maintained at one time.
wal2json	2.5	a PostgreSQL logical decoding JSON output plugin

Percona Distribution for PostgreSQL also includes the following packages:

- `llvm` 12.0.1 packages for Red Hat Enterprise Linux 8 and compatible derivatives. This fixes compatibility issues with LLVM from upstream.
- supplemental `ETCD` packages which can be used for setting up Patroni clusters. These packages are available for the following operating systems:

Operating System	Package	Version	Description
RHEL 8	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-python-etcd`	0.4.5	A Python client for ETCD

Percona Distribution for PostgreSQL is also shipped with the libpq library. It contains “a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries.”

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 30, 2023
Created: November 21, 2022

2.3 Percona Distribution for PostgreSQL 15.4 (2023-08-29)

Release date:	**August 29, 2023**
Installation:	Installing Percona Distribution for PostgreSQL

Percona Distribution for PostgreSQL is a solution with the collection of tools from PostgreSQL community that are tested to work together and serve to assist you in deploying and managing PostgreSQL. The aim of Percona Distribution for PostgreSQL is to address the operational issues like High-Availability, Disaster Recovery, Security, Observability, Spatial data handling, Performance and Scalability and others that enterprises are facing.

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 15.4.

2.3.1 Release Highlights

- Percona Distribution for PostgreSQL components now include pg_gather - the open source extension to assess the health of PostgreSQL cluster by gathering performance and configuration data from PostgreSQL databases. This tool helps you run diagnostics of your PostgreSQL cluster and is also actively used by Percona Support.
- Percona Distribution for PostgreSQL is now available on Debian 12 (bookworm).
- The support of Ubuntu 18.04 is deprecated.

The following is the list of extensions available in Percona Distribution for PostgreSQL.

Extension	Version	Description
HAProxy	2.8.1	a high-availability and load-balancing solution
Patroni	3.1.0	a HA (High Availability) solution for PostgreSQL
PgAudit	1.7.0	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
pgAudit set_user	4.0.1	provides an additional layer of logging and control when unprivileged users must escalate themselves to superusers or object owner roles in order to perform needed maintenance tasks.
pgBackRest	2.47	a backup and restore solution for PostgreSQL
pgBadger	12.1	a fast PostgreSQL Log Analyzer.
PgBouncer	1.20.0	a lightweight connection pooler for PostgreSQL
pg_gather	v22	an SQL script for running the diagnostics of the health of PostgreSQL cluster
pgpool2	4.4.3	a middleware between PostgreSQL server and client for high availability, connection pooling and load balancing.
pg_repack	1.4.8	rebuilds PostgreSQL database objects
pg_stat_monitor	2.0.1	collects and aggregates statistics for PostgreSQL and provides histogram information.
PostGIS	3.3.4	a spatial extension for PostgreSQL.
PostgreSQL Common	252	PostgreSQL database-cluster manager. It provides a structure under which multiple versions of PostgreSQL may be installed and/or multiple clusters maintained at one time.
wal2json	2.5	a PostgreSQL logical decoding JSON output plugin

Percona Distribution for PostgreSQL also includes the following packages:

- `llvm` 12.0.1 packages for Red Hat Enterprise Linux 8 / CentOS 8. This fixes compatibility issues with LLVM from upstream.
- supplemental `ETCD` packages which can be used for setting up Patroni clusters. These packages are available for the following operating systems:

Operating System	Package	Version	Description
CentOS 8	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-python-etcd`	0.4.5	A Python client for ETCD

Percona Distribution for PostgreSQL is also shipped with the libpq library. It contains "a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries."

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: August 31, 2023
Created: November 21, 2022

2.4 Percona Distribution for PostgreSQL 15.3 (2023-06-28)

Release date:	**June 28, 2023**
Installation:	Installing Percona Distribution for PostgreSQL

Percona Distribution for PostgreSQL is a solution with the collection of tools from PostgreSQL community that are tested to work together and serve to assist you in deploying and managing PostgreSQL. The aim of Percona Distribution for PostgreSQL is to address the operational issues like High-Availability, Disaster Recovery, Security, Performance and Scalability and others that enterprises are facing.

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 15.3.

2.4.1 Release Highlights

- Percona Distribution for PostgreSQL components now include PostGIS - the open source extension that allows storing and manipulating spatial data in PostgreSQL.

The following is the list of extensions available in Percona Distribution for PostgreSQL.

Extension	Version	Description
HAProxy	2.6.13	a high-availability and load-balancing solution
Patroni	3.0.2	a HA (High Availability) solution for PostgreSQL
PgAudit	1.7.0	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
pgAudit set_user	4.0.1	provides an additional layer of logging and control when unprivileged users must escalate themselves to superusers or object owner roles in order to perform needed maintenance tasks.
pgBackRest	2.44	a backup and restore solution for PostgreSQL
pgBadger	12.1	a fast PostgreSQL Log Analyzer.
PgBouncer	1.19.1	a lightweight connection pooler for PostgreSQL
pgpool2	4.4.3	a middleware between PostgreSQL server and client for high availability, connection pooling and load balancing.
pg_repack	1.4.8	rebuilds PostgreSQL database objects
pg_stat_monitor	2.0.1	collects and aggregates statistics for PostgreSQL and provides histogram information.
PostGIS	3.3.3	a spatial extension for PostgreSQL.
PostgreSQL Common	250	PostgreSQL database-cluster manager. It provides a structure under which multiple versions of PostgreSQL may be installed and/or multiple clusters maintained at one time.
wal2json	2.5	a PostgreSQL logical decoding JSON output plugin

Percona Distribution for PostgreSQL also includes the following packages:

- `llvm` 12.0.1 packages for Red Hat Enterprise Linux 8 / CentOS 8. This fixes compatibility issues with LLVM from upstream.
- supplemental `ETCD` packages which can be used for setting up Patroni clusters. These packages are available for the following operating systems:

Operating System	Package	Version	Description
CentOS 8	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-python-etcd`	0.4.3	A Python client for ETCD

Percona Distribution for PostgreSQL is also shipped with the libpq library. It contains "a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries."

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: June 30, 2023
Created: November 21, 2022

2.5 Percona Distribution for PostgreSQL 15.2 Update (2023-05-22)

Release date:	**May 22, 2023**
Installation:	Installing Percona Distribution for PostgreSQL

Percona Distribution for PostgreSQL is a solution with the collection of tools from PostgreSQL community that are tested to work together and serve to assist you in deploying and managing PostgreSQL. The aim of Percona Distribution for PostgreSQL is to address the operational issues like High-Availability, Disaster Recovery, Security, Performance and Scalability and others that enterprises are facing.

This update of Percona Distribution for PostgreSQL includes the new version of `pg_stat_monitor` 2.0.1 that fixes the issues with the database failure.

Last update: May 22, 2023
Created: May 17, 2023

2.6 Percona Distribution for PostgreSQL 15.2 (2023-03-20)

Release date:	**March 20, 2023**
Installation:	Installing Percona Distribution for PostgreSQL

Percona Distribution for PostgreSQL is a solution with the collection of tools from PostgreSQL community that are tested to work together and serve to assist you in deploying and managing PostgreSQL. The aim of Percona Distribution for PostgreSQL is to address the operational issues like High-Availability, Disaster Recovery, Security, Performance and Scalability and others that enterprises are facing.

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 15.2.

2.6.1 Release Highlights

- A new major version of `pg_stat_monitor` 2.0.0 has been released and is now generally available with Percona Distribution for PostgreSQL.
- A new extension `pgpool` - a middleware between PostgreSQL server and client for high availability, connection pooling and load balancing - is added.
- Percona Distribution for PostgreSQL is now available on Red Hat Enterprise Linux 9 and compatible derivatives

The following is the list of extensions available in Percona Distribution for PostgreSQL.

Extension	**Version**	**Description**
Patroni	3.0.1	a HA (High Availability) solution for PostgreSQL
PgAudit	1.7.0	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
pgAudit set_user	4.0.1	provides an additional layer of logging and control when unprivileged users must escalate themselves to superusers or object owner roles in order to perform needed maintenance tasks.
pgBackRest	2.43	a backup and restore solution for PostgreSQL
pgBadger	12.0	a fast PostgreSQL Log Analyzer.
PgBouncer	1.18.0	a lightweight connection pooler for PostgreSQL
pg_repack	1.4.8	rebuilds PostgreSQL database objects
pg_stat_monitor	2.0.0	collects and aggregates statistics for PostgreSQL and provides histogram information.
PostgreSQL Common	247	PostgreSQL database-cluster manager. It provides a structure under which multiple versions of PostgreSQL may be installed and/or multiple clusters maintained at one time.
wal2json	2.5	a PostgreSQL logical decoding JSON output plugin
HAProxy	2.5.11	a high-availability and load-balancing solution
pgpool2	4.4.2	a middleware between PostgreSQL server and client for high availability, connection pooling and load balancing.

Percona Distribution for PostgreSQL also includes the following packages:

- `llvm` 12.0.1 packages for Red Hat Enterprise Linux 8 / CentOS 8. This fixes compatibility issues with LLVM from upstream.
- supplemental `ETCD` packages which can be used for setting up Patroni clusters. These packages are available for the following operating systems:

Operating System	Package	Version	Description
CentOS 8	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-python-etcd`	0.4.3	A Python client for ETCD

Percona Distribution for PostgreSQL is also shipped with the libpq library. It contains "a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries."

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: April 14, 2023
Created: November 21, 2022

2.7 Percona Distribution for PostgreSQL 15.1 (2022-11-21)

Release date:	**November 21, 2022**
Installation:	Installing Percona Distribution for PostgreSQL

Percona Distribution for PostgreSQL is a solution with the collection of tools from PostgreSQL community that are tested to work together and serve to assist you in deploying and managing PostgreSQL. The aim of Percona Distribution for PostgreSQL is to address the operational issues like High-Availability, Disaster Recovery, Security, Performance and Scalability and others that enterprises are facing.

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 15.1.

Percona Distribution for PostgreSQL now includes the meta-packages that simplify its installation. The `percona-ppg-server` meta-package installs PostgreSQL and the extensions, while `percona-ppg-server-ha` package installs high-availability components that are recommended by Percona.

The following is the list of extensions available in Percona Distribution for PostgreSQL.

Extension	**Version**	**Description**
Patroni	2.1.4	a HA (High Availability) solution for PostgreSQL
pgaudit	1.7.0	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
`pgAudit set user`	4.0.0	provides an additional layer of logging and control when unprivileged users must escalate themselves to superuser or object owner roles in order to perform needed maintenance tasks.
pgBackRest	2.41	a backup and restore solution for PostgreSQL
`pgBadger`	12.0	a fast PostgreSQL Log Analyzer.
`pgBouncer`	1.17.0	lightweight connection pooler for PostgreSQL
pg_repack	1.4.8	rebuilds PostgreSQL database objects
pg_stat_monitor	1.1.1	collects and aggregates statistics for PostgreSQL and provides histogram information.
PostgreSQL Common	241	PostgreSQL database-cluster manager. It provides a structure under which multiple versions of PostgreSQL may be installed and/or multiple clusters maintained at one time.
`wal2json`	2.5	a PostgreSQL logical decoding JSON output plugin.
HAProxy	2.5.9	The high-availability and load balancing solution for PostgreSQL

Percona Distribution for PostgreSQL also includes the following packages:

- `llvm` 12.0.1 packages for Red Hat Enterprise Linux 8 / CentOS 8. This fixes compatibility issues with LLVM from upstream.
- supplemental `ETCD` packages which can be used for setting up Patroni clusters. These packages are available for the following operating systems:

Operating System	Package	Version	Description
CentOS 8	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-python-etcd`	0.4.3	A Python client for ETCD

Percona Distribution for PostgreSQL is also shipped with the libpq library. It contains "a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries."

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: December 8, 2022
Created: November 21, 2022

2.8 Percona Distribution for PostgreSQL 15.0 (2022-10-24)

Release date:	**October 24, 2022**
Installation:	Installing Percona Distribution for PostgreSQL
Upgrade	Upgrading Percona Distribution for PostgreSQL from 14 to 15

We are pleased to announce the launch of Percona Distribution for PostgreSQL 15.0 - a solution with the collection of tools from PostgreSQL community that are tested to work together and serve to assist you in deploying and managing PostgreSQL. The aim of Percona Distribution for PostgreSQL is to address the operational issues like High-Availability, Disaster Recovery, Security, Performance and Scalability and others that enterprises are facing.

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 15.

2.8.1 Release Highlights

Percona Distribution for PostgreSQL 15 features a lot of new functionalities and enhancements to performance, replication, statistics collection, logging and more. Among them are the following:

- Added the `MERGE` command, which allows your developers to write conditional SQL statements that can include `INSERT`, `UPDATE`, and `DELETE` actions within a single statement.
- A view can now be created with the permissions of the caller, instead of the view creator. This adds an additional layer of protection to ensure that view callers have the correct permissions for working with the underlying data.
- Enhanced logical replication management:
- Row filtering and column lists for publishers lets users choose to replicate a subset of data from a table
- The ability to skip replaying a conflicting transaction and to automatically disable a subscription if an error is detected simplifies the conflict management.
- The ability to use two-phase commit (2PC) with logical replication.
- A new logging format `jsonlog` outputs log data using a defined JSON structure, which allows PostgreSQL logs to be processed in structured logging systems
- Now DBAs can manage the PostgreSQL configuration by granting users permission to alter server-level configuration parameters. Users can also browse the configuration from `psql` using the `\dconfig` command.
- Server-level statistics are now collected in shared memory, eliminating both the statistics collector process and periodically writing this data to disk
- A new built-in extension, `pg_walinspect`, lets users inspect the contents of write-ahead log files directly from a SQL interface.
- Performance optimizations:
- Improved performance of window functions such as `rank()`, `row_number()` and `count()`. Read more about performance and benchmarking results in Introducing Performance Improvement of Window Functions in PostgreSQL 15.
- Parallel execution of queries using the `SELECT DISTINCT` command.
- The support for LZ4 and zstd compression methods to write-ahead logs (WAL) and backup files improves performance and storage capabilities of your database.

See also

- PostgreSQL 15 release notes
- Percona Blog:
- A Quick Peek at the PostgreSQL 15 Beta 1
- PostgreSQL 15 – New Features to Be Excited About
- PostgreSQL 15: Stats Collector Gone? What's New?
- Highly Available PostgreSQL From Percona

The following is the list of extensions available in Percona Distribution for PostgreSQL.

Extension	Version	Description
Patroni	2.1.4	a HA (High Availability) solution for PostgreSQL
pgaudit	1.7.0	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
`pgAudit set user`	3.0.0	provides an additional layer of logging and control when unprivileged users must escalate themselves to superuser or object owner roles in order to perform needed maintenance tasks.
pgBackRest	2.41	a backup and restore solution for PostgreSQL
`pgBadger`	12.0	a fast PostgreSQL Log Analyzer.
`pgBouncer`	1.17.0	lightweight connection pooler for PostgreSQL
pg_repack	1.4.8	rebuilds PostgreSQL database objects
pg_stat_monitor	1.1.1	collects and aggregates statistics for PostgreSQL and provides histogram information.
PostgreSQL Common	241	PostgreSQL database-cluster manager. It provides a structure under which multiple versions of PostgreSQL may be installed and/or multiple clusters maintained at one time.
`wal2json`	2.5	a PostgreSQL logical decoding JSON output plugin.
HAProxy	2.5.9	The high-availability and load balancing solution for PostgreSQL

Percona Distribution for PostgreSQL also includes the following packages:

- `llvm` 12.0.1 packages for Red Hat Enterprise Linux 8 / CentOS 8. This fixes compatibility issues with LLVM from upstream.
- supplemental `ETCD` packages which can be used for setting up Patroni clusters. These packages are available for the following operating systems:

Operating System	Package	Version	Description
CentOS 8	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-python-etcd`	0.4.3	A Python client for ETCD

Percona Distribution for PostgreSQL is also shipped with the libpq library. It contains "a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries."

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: October 24, 2022

Created: October 24, 2022

3. Installation and Upgrade

3.1 Install Percona Distribution for PostgreSQL

3.1.1 Install Percona Distribution for PostgreSQL

Percona Distribution for PostgreSQL is the solution with the collection of tools from PostgreSQL community that are tested to work together and serve to assist you in deploying and managing PostgreSQL. Read more ↗ .

You can select from multiple easy-to-follow installation options, but **we recommend using a Package Manager** for a convenient and quick way to try the software first.

Package manager   Docker   Kubernetes

Percona provides installation packages in `DEB` and `RPM` format for 64-bit Linux distributions. Find the full list of supported platforms and versions on the Percona Software and Platform Lifecycle page.

If you are on Debian or Ubuntu, use `apt` for installation.

If you are on Red Hat Enterprise Linux or compatible derivatives, use `yum`.

Choose your package manager below to get access to a detailed step-by-step guide.

Install via apt →   **Install via yum →**

Get our image from Docker Hub and spin up a cluster on a Docker container for quick evaluation.

Check below to get access to a detailed step-by-step guide.

Run in Docker

Percona Operator for Kubernetes is a controller introduced to simplify complex deployments that require meticulous and secure database expertise.

Check below to get access to a detailed step-by-step guide.

Get started with Percona Operator

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 30, 2023
Created: June 4, 2021

3.1.2 Install Percona Distribution for PostgreSQL on Debian and Ubuntu

This document describes how to install Percona Server for PostgreSQL from Percona repositories on DEB-based distributions such as Debian and Ubuntu. Read more about Percona repositories ↗.

Preconditions

1. Debian and other systems that use the apt package manager include the upstream PostgreSQL server package (postgresql-15) by default. The components of Percona Distribution for PostgreSQL 15 can only be installed together with the PostgreSQL server shipped by Percona (percona-postgresql-15). If you wish to use Percona Distribution for PostgreSQL, uninstall the PostgreSQL package provided by your distribution (postgresql-15) and then install the chosen components from Percona Distribution for PostgreSQL.
2. Install `curl` for Telemetry. We use it to better understand the use of our products and improve them.

Procedure

Run all the commands in the following sections as root or using the `sudo` command:

CONFIGURE PERCONA REPOSITORY

1. Install the `percona-release` repository management tool to subscribe to Percona repositories:

- Fetch `percona-release` packages from Percona web:

```
$ wget https://repo.percona.com/apt/percona-release_latest.$(lsb_release -sc)_all.deb
```

- Install the downloaded package with `dpkg`:

```
$ sudo dpkg -i percona-release_latest.$(lsb_release -sc)_all.deb
```

- Refresh the local cache:

```
$ sudo apt update
```

2. Enable the repository

   Percona provides two repositories for Percona Distribution for PostgreSQL. We recommend enabling the Major release repository to timely receive the latest updates.

```
$ sudo percona-release setup ppg-15
```

INSTALL PACKAGES

Install using meta-package **Install packages individually**

The meta package enables you to install several components of the distribution in one go.

```
$ sudo apt install percona-ppg-server-15
```

1. Install the PostgreSQL server package:

```
$ sudo apt install percona-postgresql-15
```

2. Install the components:

Install `pg_repack`:

```
$ sudo apt install percona-postgresql-15-repack
```

Install `pgAudit`:

```
$ sudo apt install percona-postgresql-15-pgaudit
```

Install `pgBackRest`:

```
$ sudo apt install percona-pgbackrest
```

Install `Patroni`:

```
$ sudo apt install percona-patroni
```

Install `pg_stat_monitor`

Install `pgBouncer`:

```
$ sudo apt install percona-pgbouncer
```

Install `pgAudit-set_user`:

```
$ sudo apt install percona-pgaudit15-set-user
```

Install `pgBadger`:

```
$ sudo apt install percona-pgbadger
```

Install `wal2json`:

```
$ sudo apt install percona-postgresql-15-wal2json
```

Install PostgreSQL contrib extensions:

```
$ sudo apt install percona-postgresql-contrib
```

Install HAProxy

```
$ sudo apt install percona-haproxy
```

Install pgpool2

```
$ sudo apt install percona-pgpool2
```

Install `pg_gather`

START THE SERVICE

The installation process automatically initializes and starts the default database. You can check the database status using the following command:

```
$ sudo systemctl status postgresql.service
```

CONNECT TO THE POSTGRESQL SERVER

By default, `postgres` user and `postgres` database are created in PostgreSQL upon its installation and initialization. This allows you to connect to the database as the `postgres` user.

```
$ sudo su postgres
```

Open the PostgreSQL interactive terminal:

```
$ psql
```

Hint

You can connect to `psql` as the `postgres` user in one go:

```
$ sudo su - postgres -c psql
```

To exit the `psql` terminal, use the following command:

```
$ \q
```

1. Are included in repositories for Debian 12 operating system ↩

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 30, 2023
Created: November 21, 2022

3.1.3 Install Percona Distribution for PostgreSQL on Red Hat Enterprise Linux and derivatives

This document describes how to install Percona Distribution for PostgreSQL from Percona repositories on RPM-based distributions such as Red Hat Enterprise Linux and compatible derivatives. Read more about Percona repositories ↗.

Platform specific notes

To install Percona Distribution for PostgreSQL, do the following:

FOR PERCONA DISTRIBUTION FOR POSTGRESQL PACKAGES

CentOS 7   RHEL8/Oracle Linux 8/Rocky Linux 8

Install the `epel-release` package:

```
$ sudo yum -y install epel-release
$ sudo yum repolist
```

Disable the `postgresql` and `llvm-toolset` modules:

```
$ sudo dnf module disable postgresql llvm-toolset
```

FOR `PERCONA-POSTGRESQL15-DEVEL` PACKAGE

You may need to install the `percona-postgresql15-devel` package when working with some extensions or creating programs that interface with PostgreSQL database. This package requires dependencies that are not part of the Distribution, but can be installed from the specific repositories:

RHEL8   Rocky Linux 8   Oracle Linux 8   Rocky Linux 9   Oracle Linux 9   Rocky Linux 8   Oracle Linux 8

```
$ sudo yum --enablerepo=codeready-builder-for-rhel-8-rhui-rpms install perl-IPC-Run -y

$ sudo dnf install dnf-plugins-core
$ sudo dnf module enable llvm-toolset
$ sudo dnf config-manager --set-enabled powertools

$ sudo dnf config-manager --set-enabled ol8_codeready_builder install perl-IPC-Run -y

$ sudo dnf install dnf-plugins-core
$ sudo dnf module enable llvm-toolset
$ sudo dnf config-manager --set-enabled crb
$ sudo dnf install perl-IPC-Run -y

$ sudo dnf config-manager --set-enabled ol9_codeready_builder install perl-IPC-Run -y

$ sudo dnf config-manager --set-enabled powertools install perl-IPC-Run -y

$ sudo dnf config-manager --set-enabled ol9_codeready_builder install perl-IPC-Run -y
```

FOR `PGPOOL2` EXTENSION

To install `pgpool2` on Red Hat Enterprise Linux and compatible derivatives, enable the codeready builder repository first to resolve dependencies conflict for `pgpool2`.

The following are commands for Red Hat Enterprise Linux 9 and derivatives. For Red Hat Enterprise Linux 8, replace the operating system version in the commands accordingly.

RHEL 9   **Rocky Linux 9**   **Oracle Linux 9**

```
$ sudo dnf config-manager --set-enabled codeready-builder-for-rhel-9-x86_64-rpms

$ sudo dnf config-manager --set-enabled crb

$ sudo dnf config-manager --set-enabled ol9_codeready_builder
```

FOR POSTGIS

For Red Hat Enterprise Linux 8 and derivatives, replace the operating system version in the following commands accordingly.

RHEL 9 **Rocky Linux 9** **Oracle Linux 9** **RHEL UBI 9**

1. Install `epel` repository

```
$ sudo yum install epel-release
```

2. Enable the `llvm-toolset dnf` module

```
$ sudo dnf module enable llvm-toolset
```

3. Enable the codeready builder repository to resolve dependencies conflict.

```
$ sudo dnf config-manager --set-enabled codeready-builder-for-rhel-9-x86_64-rpms
```

1. Install `epel` repository

```
$ sudo yum install epel-release
```

2. Enable the `llvm-toolset dnf` module

```
$ sudo dnf module enable llvm-toolset
```

3. Enable the codeready builder repository to resolve dependencies conflict.

```
$ sudo dnf install dnf-plugins-core
$ sudo dnf config-manager --set-enabled crb
```

1. Install `epel` repository

```
$ sudo yum install epel-release
```

2. Enable the `llvm-toolset dnf` module

```
$ sudo dnf module enable llvm-toolset
```

3. Enable the codeready builder repository to resolve dependencies conflict.

```
$ sudo dnf config-manager --set-enabled ol9_codeready_builder
```

1. Configure the Oracle-Linux repository. Create the `/etc/yum.repos.d/oracle-linux-ol9.repo` file to install the required dependencies:

/etc/yum.repos.d/oracle-linux-ol9.repo

```
[ol9_baseos_latest]
name=Oracle Linux 9 BaseOS Latest ($basearch)
baseurl=https://yum.oracle.com/repo/OracleLinux/OL9/baseos/latest/$basearch/
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
gpgcheck=1
enabled=1

[ol9_appstream]
name=Oracle Linux 9 Application Stream ($basearch)
baseurl=https://yum.oracle.com/repo/OracleLinux/OL9/appstream/$basearch/
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-oracle
gpgcheck=1
enabled=1
```

Procedure

Run all the commands in the following sections as root or using the `sudo` command.

INSTALL DEPENDENCIES

Install `curl` for Telemetry. We use it to better understand the use of our products and improve them.

```
$ sudo yum -y install curl
```

CONFIGURE THE REPOSITORY

1. Install the `percona-release` repository management tool to subscribe to Percona repositories:

```
$ sudo yum install https://repo.percona.com/yum/percona-release-latest.noarch.rpm
```

2. Enable the repository

   Percona provides two repositories for Percona Distribution for PostgreSQL. We recommend enabling the Major release repository to timely receive the latest updates.

```
$ sudo percona-release setup ppg-15
```

INSTALL PACKAGES

Install using meta-package **Install packages individually**

The meta package enables you to install several components of the distribution in one go.

```
$ sudo yum install percona-ppg-server15
```

1. Install the PostgreSQL server package:

```
$ sudo yum install percona-postgresql15-server
```

2. Install the components:

Install `pg_repack`:

```
$ sudo yum install percona-pg_repack15
```

Install `pgaudit`:

```
$ sudo yum install percona-pgaudit15
```

Install `pgBackRest`:

```
$ sudo yum install percona-pgbackrest
```

Install `Patroni`:

```
$ sudo yum install percona-patroni
```

Install `pg_stat_monitor`:

Install `pgBouncer`:

```
$ sudo yum install percona-pgbouncer
```

Install `pgAudit-set_user`:

```
$ sudo yum install percona-pgaudit15_set_user
```

Install `pgBadger`:

```
$ sudo yum install percona-pgbadger
```

Install `wal2json`:

```
$ sudo yum install percona-wal2json15
```

Install PostgreSQL contrib extensions:

```
$ sudo yum install percona-postgresql15-contrib
```

Install HAProxy

```
$ sudo yum install percona-haproxy
```

Install `pg_gather`

```
$ sudo yum install percona-pg_gather
```

Install pgpool2

Some extensions require additional setup in order to use them with Percona Distribution for PostgreSQL. For more information, refer to Enabling extensions.

START THE SERVICE

After the installation, the default database storage is not automatically initialized. To complete the installation and start Percona Distribution for PostgreSQL, initialize the database using the following command:

```
$ /usr/pgsql-15/bin/postgresql-15-setup initdb
```

Start the PostgreSQL service:

```
$ sudo systemctl start postgresql-15
```

CONNECT TO THE POSTGRESQL SERVER

By default, `postgres` user and `postgres` database are created in PostgreSQL upon its installation and initialization. This allows you to connect to the database as the `postgres` user.

```
$ sudo su postgres
```

Open the PostgreSQL interactive terminal:

```
$ psql
```

Hint

You can connect to `psql` as the `postgres` user in one go:

```
$ sudo su - postgres -c psql
```

To exit the `psql` terminal, use the following command:

```
$ \q
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 30, 2023

Created: November 21, 2022

3.1.4 Enable Percona Distribution for PostgreSQL extensions

Some extensions require additional configuration before using them with Percona Distribution for PostgreSQL. This sections provides configuration instructions per extension.

Patroni

Patroni is the third-party high availability solution for PostgreSQL. The High Availability in PostgreSQL with Patroni chapter provides details about the solution overview and architecture deployment.

While setting up a high availability PostgreSQL cluster with Patroni, you will need the following components:

- Patroni installed on every `postresql` node.
- Distributed Configuration Store (DCS). Patroni supports such DCSs as ETCD, zookeeper, Kubernetes though ETCD is the most popular one. It is available upstream as DEB packages for Debian 10, 11 and Ubuntu 18.04, 20.04, 22.04.

  For CentOS 8, RPM packages for ETCD is available within Percona Distribution for PostreSQL. You can install it using the following command:

  ```
  $ sudo yum install etcd python3-python-etcd
  ```

- HAProxy.

See the configuration guidelines for Debian and Ubuntu and RHEL and CentOS.

See also

- Patroni documentation
- Percona Blog:
- PostgreSQL HA with Patroni: Your Turn to Test Failure Scenarios

pgBadger

Enable the following options in `postgresql.conf` configuration file before starting the service:

```
log_min_duration_statement = 0
log_line_prefix = '%t [%p]: '
log_checkpoints = on
log_connections = on
log_disconnections = on
log_lock_waits = on
log_temp_files = 0
log_autovacuum_min_duration = 0
log_error_verbosity = default
```

For details about each option, see pdBadger documentation.

pgAudit set-user

Add the `set-user` to `shared_preload_libraries` in `postgresql.conf`. The recommended way is to use the ALTER SYSTEM command. Connect to psql and use the following command:

```
ALTER SYSTEM SET shared_preload_libraries = 'set-user';
```

Start / restart the server to apply the configuration.

You can fine-tune user behavior with the custom parameters supplied with the extension.

wal2json

After the installation, enable the following option in `postgresql.conf` configuration file before starting the service:

```
wal_level = logical
```

Last update: May 22, 2023
Created: November 21, 2022

3.1.5 Repositories overview

Percona provides two repositories for Percona Distribution for PostgreSQL.

Major release repository

Major Release repository (`ppg-15`) it includes the latest version packages. Whenever a package is updated, the package manager of your operating system detects that and prompts you to update. As long as you update all Distribution packages at the same time, you can ensure that the packages you're using have been tested and verified by Percona.

We recommend installing Percona Distribution for PostgreSQL from the *Major Release repository*

Minor release repository

Minor Release repository includes a particular minor release of the database and all of the packages that were tested and verified to work with that minor release (e.g. `ppg-15.1`). You may choose to install Percona Distribution for PostgreSQL from the Minor Release repository if you have decided to standardize on a particular release which has passed rigorous testing procedures and which has been verified to work with your applications. This allows you to deploy to a new host and ensure that you'll be using the same version of all the Distribution packages, even if newer releases exist in other repositories.

The disadvantage of using a Minor Release repository is that you are locked in this particular release. When potentially critical fixes are released in a later minor version of the database, you will not be prompted for an upgrade by the package manager of your operating system. You would need to change the configured repository in order to install the upgrade.

Repository contents

Percona Distribution for PostgreSQL provides individual packages for its components. It also includes two meta-packages: `percona-ppg-server` and `percona-ppg-server-ha` .

Using a meta-package, you can install all components it contains in one go.

`PERCONA-PPG-SERVER`

Package name on Debian/Ubuntu	Package name on RHEL/derivatives
`percona-ppg-server-15`	`percona-ppg-server15`

The `percona-ppg-server` meta-package installs the PostgreSQL server with the following packages:

Package contents	Description
`percona-postgresql15-server`	The PostgreSQL server package.
`percona-postgresql-common`	PostgreSQL database-cluster manager. It provides a structure under which multiple versions of PostgreSQL may be installed and/or multiple clusters maintained at one time.
`percona-postgresql15-contrib`	A collection of additional PostgreSQLcontrib extensions
`percona-pg-stat-monitor15`	A Query Performance Monitoring tool for PostgreSQL.
`percona-pgaudit`	Provides detailed session or object audit logging via the standard PostgreSQL logging facility.
`percona-pg_repack15`	rebuilds PostgreSQL database objects.
`percona-wal2json15`	a PostgreSQL logical decoding JSON output plugin.

`PERCONA-PPG-SERVER-HA`

Package name on Debian/Ubuntu	Package name on RHEL/derivatives
`percona-ppg-server-ha-15`	
`percona-ppg-server-15`	

The `percona-ppg-server-ha` meta-package installs high-availability components that are recommended by Percona:

Package contents	Description
`percona-patroni`	A high-availability solution for PostgreSQL.
`percona-haproxy`	A high-availability and load-balancing solution
`etcd`	A consistent, distributed key-value store
`python3-python-etcd`	A Python client for ETCD.[1]
`etcd-client`, `etcd-server`	The client/server of the distributed key-value store. [2]

1. Is included in repositories for RHEL 8 / CentOS 8 operating systems ↩
2. Are included in repositories for Debian 12 operating system ↩

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 30, 2023

Created: November 21, 2022

3.2 Run Percona Distribution for PostgreSQL in a Docker container

Docker images of Percona Distribution for PostgreSQL are hosted publicly on Docker Hub.

For more information about using Docker, see the Docker Docs.

Make sure that you are using the latest version of Docker. The ones provided via `apt` and `yum` may be outdated and cause errors.

By default, Docker pulls the image from Docker Hub if it is not available locally.

Docker image contents

The Docker image of Percona Distribution for PostgreSQL includes the following components:

Component name	Description
`percona-postgresql15`	A metapackage that installs the latest version of PostgreSQL
`percona-postgresql15-server`	The PostgreSQL server package.
`percona-postgresql-common`	PostgreSQL database-cluster manager. It provides a structure under which multiple versions of PostgreSQL may be installed and/or multiple clusters maintained at one time.
`percona-postgresql-client-common`	The manager for multiple PostgreSQL client versions.
`percona-postgresql15-contrib`	A collection of additional PostgreSQLcontrib extensions
`percona-postgresql15-libs`	Libraries for use with PostgreSQL.
`percona-pg-stat-monitor15`	A Query Performance Monitoring tool for PostgreSQL.
`percona-pgaudit15`	Provides detailed session or object audit logging via the standard PostgreSQL logging facility.
`percona-pgaudit15_set_user`	An additional layer of logging and control when unprivileged users must escalate themselves to superuser or object owner roles in order to perform needed maintenance tasks.
`percona-pg_repack15`	rebuilds PostgreSQL database objects.
`percona-wal2json15`	a PostgreSQL logical decoding JSON output plugin.

3.2.1 Start the container

1. Start a Percona Distribution for PostgreSQL container as follows:

```
$ docker run --name container-name -e POSTGRES_PASSWORD=secret -d percona/percona-
distribution-postgresql:tag
```

Where:

- `container-name` is the name you assign to your container
- `POSTGRES_PASSWORD` is the superuser password
- `tag` is the tag specifying the version you want.

Check the full list of tags.

Tip

You can secure the password by exporting it to the environment file and using that to start the container.

a. Export the password to the environment file:

```
$ echo "POSTGRES_PASSWORD=secret" > .my-pg.env
```

b. Start the container:

```
$ docker run --name container-name --env-file ./.my-pg.env -d percona/percona-distribution-
postgresql:tag
```

2. Connect to the container's interactive terminal:

```
$ docker exec -it container-name bash
```

The `container-name` is the name of the container that you started in the previous step.

3.2.2 Connect to Percona Distribution for PostgreSQL from an application in another Docker container

This image exposes the standard PostgreSQL port (`5432`), so container linking makes the instance available to other containers. Start other containers like this in order to link it to the Percona Distribution for PostgreSQL container:

```
$ docker run --name app-container-name --network container:container-name -d app-that-uses-
postgresql
```

where:

- `app-container-name` is the name of the container where your application is running,
- `container name` is the name of your Percona Distribution for PostgreSQL container, and
- `app-that-uses-postgresql` is the name of your PostgreSQL client.

3.2.3 Connect to Percona Distribution for PostgreSQL from the `psql` command line client

The following command starts another container instance and runs the `psql` command line client against your original container, allowing you to execute SQL statements against your database:

```
$ docker run -it --network container:db-container-name --name container-name percona/
percona-distribution-postgresql:tag psql -h address -U postgres
```

Where:

- `db-container-name` is the name of your database container
- `container-name` is the name of your container that you will use to connect to the database container using the `psql` command line client
- `tag` is the tag specifying the Docker image version you want to use.
- `address` is the network address where your database container is running. Use 127.0.0.1, if the database container is running on the local machine/host.

3.2.4 Enable `pg_stat_monitor`

To enable the `pg_stat_monitor` extension after launching the container, do the following:

- connect to the server,
- select the desired database and enable the `pg_stat_monitor` view for that database:

```
create extension pg_stat_monitor;
```

- to ensure that everything is set up correctly, run:

```
\d pg_stat_monitor;
```

Output

```
                          View "public.pg_stat_monitor"
      Column         |           Type           | Collation | Nullable | Default
---------------------+--------------------------+-----------+----------+---------
bucket               | integer                  |           |          |
bucket_start_time    | timestamp with time zone |           |          |
userid               | oid                      |           |          |
dbid                 | oid                      |           |          |
queryid              | text                     |           |          |
query                | text                     |           |          |
plan_calls           | bigint                   |           |          |
plan_total_time      | numeric                  |           |          |
plan_min_timei       | numeric                  |           |          |
plan_max_time        | numeric                  |           |          |
plan_mean_time       | numeric                  |           |          |
plan_stddev_time     | numeric                  |           |          |
plan_rows            | bigint                   |           |          |
calls                | bigint                   |           |          |
total_time           | numeric                  |           |          |
min_time             | numeric                  |           |          |
max_time             | numeric                  |           |          |
mean_time            | numeric                  |           |          |
stddev_time          | numeric                  |           |          |
rows                 | bigint                   |           |          |
shared_blks_hit      | bigint                   |           |          |
shared_blks_read     | bigint                   |           |          |
shared_blks_dirtied  | bigint                   |           |          |
shared_blks_written  | bigint                   |           |          |
local_blks_hit       | bigint                   |           |          |
local_blks_read      | bigint                   |           |          |
local_blks_dirtied   | bigint                   |           |          |
local_blks_written   | bigint                   |           |          |
temp_blks_read       | bigint                   |           |          |
temp_blks_written    | bigint                   |           |          |
blk_read_time        | double precision         |           |          |
blk_write_time       | double precision         |           |          |
host                 | bigint                   |           |          |
client_ip            | inet                     |           |          |
resp_calls           | text[]                   |           |          |
cpu_user_time        | double precision         |           |          |
cpu_sys_time         | double precision         |           |          |
tables_names         | text[]                   |           |          |
wait_event           | text                     |           |          |
wait_event_type      | text                     |           |          |
```

Note that the `pg_stat_monitor` view is available only for the databases where you enabled it. If you create a new database, make sure to create the view for it to see its statistics data.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 30, 2023

Created: November 30, 2023

3.3 Migrate from PostgreSQL to Percona Distribution for PostgreSQL

Percona Distribution for PostgreSQL includes the PostgreSQL database and additional extensions that have been selected to cover the needs of the enterprise and are guaranteed to work together. Percona Distribution for PostgreSQL is available as a software collection that is easy to deploy.

We encourage users to migrate from their PostgreSQL deployments based on community binaries to Percona Distribution for PostgreSQL. This document provides the migration instructions.

Depending on your business requirements, you may migrate to Percona Distribution for PostgreSQL either on the same server or onto a different server.

3.3.1 Migrate on the same server

On Debian and Ubuntu Linux **On RHEL and derivatives**

To ensure that your data is safe during the migration, we recommend to make a backup of your data and all configuration files (such as `pg_hba.conf`, `postgresql.conf`, `postgresql.auto.conf`) using the tool of your choice. The backup process is out of scope of this document. You can use `pg_dumpall` or other tools of your choice.

1. Stop the `postgresql` server

```
$ sudo systemctl stop postgresql.service
```

2. Remove community packages

```
$ sudo apt-get --purge remove postgresql
```

3. Install percona-release
4. Enable the repository

```
$ sudo percona-release setup ppg15
```

5. Install Percona Distribution for PostgreSQL packages
6. (Optional) Restore the data from the backup.
7. Start the `postgresql` service. The installation process starts and initializes the default cluster automatically. You can check its status with:

```
$ sudo systemctl status postgresql
```

If `postresql` service is not started, start it manually:

```
$ sudo systemctl start postgresql.service
```

To ensure that your data is safe during the migration, we recommend to make a backup of your data and all configuration files (such as `pg_hba.conf`, `postgresql.conf`, `postgresql.auto.conf`) using the tool of your choice. The backup process is out of scope of this document. You can use `pg_dumpall` or other tools of your choice.

1. Stop the `postgresql` server

```
$ sudo systemctl stop postgresql-15
```

2. Remove community packages

```
$ sudo yum remove postgresql
```

3. Install percona-release
4. Enable the repository

```
$ sudo percona-release setup ppg15
```

5. Install Percona Distribution for PostgreSQL packages
6. (Optional) Restore the data from the backup.
7. Start the `postgresql` service

```
$ sudo systemctl start postgresql-15
```

3.3.2 Migrate on a different server

In this scenario, we will refer to the server with PostgreSQL Community as the “source” and to the server with Percona Distribution for PostgreSQL as the “target”.

To migrate from PostgreSQL Community to Percona Distribution for PostgreSQL on a different server, do the following:

On the source server:

1. Back up your data and all configuration files (such as `pg_hba.conf`, `postgresql.conf`, `postgresql.auto.conf`) using the tool of your choice.
2. Stop the `postgresql` service

   On Debian and Ubuntu **On RHEL and derivatives**

   ```
   $ sudo systemctl stop postgresql.service

   $ sudo systemctl stop postgresql-15
   ```

3. Optionally, remove PostgreSQL Community packages

On the target server:

1. Install percona-release
2. Enable the repository

   ```
   $ sudo percona-release setup ppg15
   ```

3. Install Percona Distribution for PostgreSQL packages on the target server.
4. Restore the data from the backup
5. Start `postgresql` service

   On Debian and Ubuntu **On RHEL and derivatives**

   ```
   $ sudo systemctl start postgresql.service

   $ sudo systemctl start postgresql-15
   ```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: December 5, 2022

Created: July 22, 2022

3.4 Upgrading Percona Distribution for PostgreSQL from 14 to 15

This document describes the in-place upgrade of Percona Distribution for PostgreSQL using the `pg_upgrade` tool.

The in-place upgrade means installing a new version without removing the old version and keeping the data files on the server.

See also

`pg_upgrade` Documentation

Similar to installing, we recommend you to upgrade Percona Distribution for PostgreSQL from Percona repositories.

Important

A major upgrade is a risky process because of many changes between versions and issues that might occur during or after the upgrade. Therefore, make sure to back up your data first. The backup tools are out of scope of this document. Use the backup tool of your choice.

The general in-place upgrade flow for Percona Distribution for PostgreSQL is the following:

1. Install Percona Distribution for PostgreSQL 15 packages.
2. Stop the PostgreSQL service.
3. Check the upgrade without modifying the data.
4. Upgrade Percona Distribution for PostgreSQL.
5. Start PostgreSQL service.
6. Execute the **analyze_new_cluster.sh** script to generate statistics so the system is usable.
7. Delete old packages and configuration files.

The exact steps may differ depending on the package manager of your operating system.

3.4.1 On Debian and Ubuntu using `apt`

Important

Run **all** commands as root or via **sudo**.

1. Install Percona Distribution for PostgreSQL 15 packages.

- Install percona-release
- Enable Percona repository:

```
$ sudo percona-release setup ppg-15
```

- Install Percona Distribution for PostgreSQL 15 package:

```
$ sudo apt install percona-postgresql-15
```

2. Stop the `postgresql` service.

```
$ sudo systemctl stop postgresql.service
```

This stops both Percona Distribution for PostgreSQL 14 and 15.

3. Run the database upgrade.

- Log in as the `postgres` user.

```
$ sudo su postgres
```

- Change the current directory to the `tmp` directory where logs and some scripts will be recorded:

```
$ cd tmp/
```

- Check the ability to upgrade Percona Distribution for PostgreSQL from 14 to 15:

```
$ /usr/lib/postgresql/15/bin/pg_upgrade \
--old-datadir=/var/lib/postgresql/14/main \
--new-datadir=/var/lib/postgresql/15/main  \
--old-bindir=/usr/lib/postgresql/14/bin  \
--new-bindir=/usr/lib/postgresql/15/bin  \
--old-options '-c config_file=/etc/postgresql/14/main/postgresql.conf' \
--new-options '-c config_file=/etc/postgresql/15/main/postgresql.conf' \
--check
```

The `--check` flag here instructs `pg_upgrade` to only check the upgrade without changing any data.

Sample output

```
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok
Checking database user is the install user                  ok
Checking database connection settings                       ok
Checking for prepared transactions                          ok
Checking for reg* data types in user tables                 ok
Checking for contrib/isn with bigint-passing mismatch       ok
Checking for tables WITH OIDS                               ok
Checking for invalid "sql_identifier" user columns          ok
Checking for presence of required libraries                 ok
Checking database user is the install user                  ok
Checking for prepared transactions                          ok
```

```
*Clusters are compatible*
```

- Upgrade the Percona Distribution for PostgreSQL

```
$ /usr/lib/postgresql/15/bin/pg_upgrade
--old-datadir=/var/lib/postgresql/14/main \
--new-datadir=/var/lib/postgresql/15/main  \
--old-bindir=/usr/lib/postgresql/14/bin  \
--new-bindir=/usr/lib/postgresql/15/bin  \
--old-options '-c config_file=/etc/postgresql/14/main/postgresql.conf' \
--new-options '-c config_file=/etc/postgresql/15/main/postgresql.conf' \
--link
```

The `--link` flag creates hard links to the files on the old version cluster so you don't need to copy data.

If you don't wish to use the `--link` option, make sure that you have enough disk space to store 2 copies of files for both old version and new version clusters.

- Go back to the regular user:

```
$ exit
```

- The Percona Distribution for PostgreSQL 14 uses the `5432` port while the Percona Distribution for PostgreSQL 15 is set up to use the `5433` port by default. To start the Percona Distribution for PostgreSQL 15, swap ports in the configuration files of both versions.

```
$ sudo vim /etc/postgresql/15/main/postgresql.conf
$ port = 5433 # Change to 5432 here
$ sudo vim /etc/postgresql/14/main/postgresql.conf
$ port = 5432 # Change to 5433 here
```

4. Start the `postgreqsl` service.

```
$ sudo systemctl start postgresql.service
```

5. Check the `postgresql` version.

- Log in as a postgres user

```
$ sudo su postgres
```

- Check the database version

```
$ psql -c "SELECT version();"
```

6. After the upgrade, the Optimizer statistics are not transferred to the new cluster. Run the `vaccumdb` command to analyze the new cluster:

```
$ /usr/lib/postgresql/15/bin/vacuumdb --all --analyze-in-stages
```

7. Delete the old cluster's data files:

```
$ ./delete_old_cluster.sh
$ sudo rm -rf /etc/postgresql/14/main
$ #Logout
$ exit
```

3.4.2 On Red Hat Enterprise Linux and CentOS using `yum`

Important

Run **all** commands as root or via **sudo**.

1. Install Percona Distribution for PostgreSQL 15 packages

- Install percona-release
- Enable Percona repository:

```
$ sudo percona-release setup ppg-15
```

- Install Percona Distribution for PostgreSQL 15:

```
$ sudo yum install percona-postgresql15-server
```

2. Set up Percona Distribution for PostgreSQL 15 cluster
3. Log is as the postgres user

```
$ sudo su postgres
```

4. Set up locale settings

```
export LC_ALL="en_US.UTF-8"
export LC_CTYPE="en_US.UTF-8"
```

5. Initialize cluster with the new data directory

```
$ /usr/pgsql-15/bin/initdb -D /var/lib/pgsql/15/data
```

6. Stop the `postgresql` 14 service

```
$ systemctl stop postgresql-14
```

7. Run the database upgrade.

- Log in as the `postgres` user

```
$ sudo su postgres
```

- Check the ability to upgrade Percona Distribution for PostgreSQL from 14 to 15:

```
$ /usr/pgsql-15/bin/pg_upgrade \
--old-bindir /usr/pgsql-14/bin \
--new-bindir /usr/pgsql-15/bin  \
--old-datadir /var/lib/pgsql/14/data \
--new-datadir /var/lib/pgsql/15/data \
--check
```

The `--check` flag here instructs `pg_upgrade` to only check the upgrade without changing any data.

Sample output

```
Performing Consistency Checks
-----------------------------
Checking cluster versions                                   ok
Checking database user is the install user                  ok
Checking database connection settings                       ok
Checking for prepared transactions                          ok
Checking for reg* data types in user tables                 ok
Checking for contrib/isn with bigint-passing mismatch       ok
Checking for tables WITH OIDS                               ok
Checking for invalid "sql_identifier" user columns          ok
Checking for presence of required libraries                 ok
Checking database user is the install user                  ok
Checking for prepared transactions                          ok

*Clusters are compatible*
```

- Upgrade the Percona Distribution for PostgreSQL

```
$ /usr/pgsql-15/bin/pg_upgrade \
--old-bindir /usr/pgsql-14/bin \
--new-bindir /usr/pgsql-15/bin  \
--old-datadir /var/lib/pgsql/14/data \
--new-datadir /var/lib/pgsql/15/data \
--link
```

The `--link` flag creates hard links to the files on the old version cluster so you don't need to copy data. If you don't wish to use the `--link` option, make sure that you have enough disk space to store 2 copies of files for both old version and new version clusters.

8. Start the `postgresql` 15 service.

```
$ systemctl start postgresql-15
```

9. Check postgresql status

```
$ systemctl status postgresql-15
```

10. After the upgrade, the Optimizer statistics are not transferred to the new cluster. Run the `vaccumdb` command to analyze the new cluster:

- Log in as the postgres user

```
$ sudo su postgres
```

- Run the `vaccumdb` command

```
$ /usr/pgsql-15/bin/vacuumdb --all --analyze-in-stages
```

11. Delete Percona Distribution for PostgreSQL 14 configuration files

```
$ ./delete_old_cluster.sh
```

12. Delete Percona Distribution for PostgreSQL old data files

```
$ rm -rf /var/lib/pgsql/14/data
```

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: October 31, 2023

Created: June 4, 2021

3.5 Minor Upgrade of Percona Distribution for PostgreSQL

Minor releases of PostgreSQL include bug fixes and feature enhancements. We recommend that you keep your Percona Distribution for PostgreSQL updated to the latest minor version.

Though minor upgrades do not change the behavior, we recommend you to back up your data first, in order to be on the safe side.

Minor upgrade of Percona Distribution for PostgreSQL includes the following steps:

1. Stop the `postgresql` cluster;
2. Install new version packages;
3. Restart the `postgresql` cluster.

Note

These steps apply if you installed Percona Distribution for PostgreSQL from the Major Release repository. In this case, you are always upgraded to the latest available release.

If you installed Percona Distribution for PostgreSQL from the Minor Release repository, you will need to enable a new version repository to upgrade.

For more information about Percona repositories, refer to Installing Percona Distribution for PostgreSQL.

Before the upgrade, update the **percona-release** utility to the latest version. This is required to install the new version packages of Percona Distribution for PostgreSQL. Refer to Percona Software Repositories Documentation for update instructions.

Important

Run all commands as root or via **sudo**.

1. Stop the `postgresql` service.

   On Debian / Ubuntu **On Red Hat Enterprise Linux / derivatives**

   ```
   $ sudo systemctl stop postgresql.service

   $ sudo systemctl stop postgresql-15
   ```

2. Install new version packages. See Installing Percona Distribution for PostgreSQL.
3. Restart the `postgresql` service.

   On Debian / Ubuntu **On Red Hat Enterprise Linux / derivatives**

   ```
   $ sudo systemctl start postgresql.service

   $ sudo systemctl start postgresql-15
   ```

If you wish to upgrade Percona Distribution for PostgreSQL to the major version, refer to Upgrading Percona Distribution for PostgreSQL from 14 to 15.

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: December 5, 2022

Created: June 4, 2021

4. Extensions

4.1 pg_stat_monitor

Note

This document describes the functionality of pg_stat_monitor 2.0.0.

4.1.1 Overview

`pg_stat_monitor` is a Query Performance Monitoring tool for PostgreSQL. It collects various statistics data such as query statistics, query plan, SQL comments and other performance insights. The collected data is aggregated and presented in a single view. This allows you to view queries from performance, application and analysis perspectives.

`pg_stat_monitor` groups statistics data and writes it in a storage unit called *bucket*. The data is added and stored in a bucket for the defined period – the bucket lifetime. This allows you to identify performance issues and patterns based on time.

You can specify the following:

- The number of buckets. Together they form a bucket chain.
- Bucket size. This is the amount of shared memory allocated for buckets. Memory is divided equally among buckets.
- Bucket lifetime.

When a bucket lifetime expires, `pg_stat_monitor` resets all statistics and writes the data in the next bucket in the chain. When the last bucket's lifetime expires, `pg_stat_monitor` returns to the first bucket.

Important

The contents of the bucket will be overwritten. In order not to lose the data, make sure to read the bucket before `pg_stat_monitor` starts writing new data to it.

Views

PG_STAT_MONITOR VIEW

The `pg_stat_monitor` view contains all the statistics collected and aggregated by the extension. This view contains one row for each distinct combination of metrics and whether it is a top-level statement or not (up to the maximum number of distinct statements that the module can track). For details about available metrics, refer to the `pg_stat_monitor` view reference.

The following are the primary keys for pg_stat_monitor:

- `bucket`
- `userid`
- `datname`
- `queryid`
- `client_ip`
- `planid`
- `application_name`

A new row is created for each key in the `pg_stat_monitor` view.

For security reasons, only superusers and members of the `pg_read_all_stats` role are allowed to see the SQL text, `client_ip` and `queryid` of queries executed by other users. Other users can see the statistics, however, if the view has been installed in their database.

PG_STAT_MONITOR_SETTINGS VIEW (DROPPED)

Starting with version 2.0.0, the `pg_stat_monitor_settings` view is deprecated and removed. All `pg_stat_monitor` configuration parameters are now available though the `pg_settings` view using the following query:

```
SELECT name, setting, unit, context, vartype, source, min_val, max_val, enumvals, boot_val,
reset_val, pending_restart FROM pg_settings WHERE name LIKE '%pg_stat_monitor%';
```

For backward compatibility, you can create the `pg_stat_monitor_settings` view using the following SQL statement:

```
CREATE VIEW pg_stat_monitor_settings

AS

SELECT *

FROM pg_settings

WHERE name like 'pg_stat_monitor.%';
```

In `pg_stat_monitor` version 1.1.1 and earlier, the `pg_stat_monitor_settings` view shows one row per `pg_stat_monitor` configuration parameter. It displays configuration parameter name, value, default value, description, minimum and maximum values, and whether a restart is required for a change in value to be effective.

To learn more, see the Changing the configuration section.

4.1.2 Installation

This section describes how to install `pg_stat_monitor` from Percona repositories. To learn about other installation methods, see the Installation section in the `pg_stat_monitor` documentation.

Preconditions:

To install `pg_stat_monitor` from Percona repositories, you need to subscribe to them. To do this, you must have the `percona-release` repository management tool up and running.

To install `pg_stat_monitor`, run the following commands:

On Debian and Ubuntu **On Red Hat Enterprise Linux and derivatives**

1. Enable the repository

```
$ sudo percona-release setup ppg15
```

2. Install the package:

```
$ sudo apt-get install percona-pg-stat-monitor15
```

1. Enable the repository

```
$ sudo percona-release setup ppg15
```

2. Install the package:

```
$ sudo yum install percona-pg-stat-monitor15
```

4.1.3 Setup

`pg_stat_monitor` requires additional setup in order to use it with PostgreSQL. The setup steps are the following:

1. Add `pg_stat_monitor` in the `shared_preload_libraries` configuration parameter.

   The recommended way to modify PostgreSQL configuration file is using the ALTER SYSTEM command. Connect to psql and use the following command:

   ```
   ALTER SYSTEM SET shared_preload_libraries = 'pg_stat_monitor';
   ```

   The parameter value is written to the `postgresql.auto.conf` file which is read in addition with `postgresql.conf` file.

   Note

   To use `pg_stat_monitor` together with `pg_stat_statements`, specify both modules separated by commas for the `ALTER SYSTEM SET` command.

   The order of modules is important: `pg_stat_monitor` must be specified **after** `pg_stat_statements`:

   ```
   ALTER SYSTEM SET shared_preload_libraries = ‘pg_stat_statements, pg_stat_monitor’
   ```

2. Start or restart the `postgresql` instance to enable `pg_stat_monitor`. Use the following command for restart:

   On Debian and Ubuntu **On Red Hat Enterprise Linux and derivatives**

   ```
   $ sudo systemctl restart postgresql.service
   ```

   ```
   $ sudo systemctl restart postgresql-15
   ```

3. Create the extension. Connect to `psql` and use the following command:

   ```
   CREATE EXTENSION pg_stat_monitor;
   ```

   By default, the extension is created against the `postgres` database. You need to create the extension on every database where you want to collect statistics.

   Tip

   To check the version of the extension, run the following command in the `psql` session:

   ```
   SELECT pg_stat_monitor_version();
   ```

4.1.4 Usage

For example, to view the IP address of the client application that made the query, run the following command:

```
SELECT DISTINCT userid::regrole, pg_stat_monitor.datname, substr(query,0, 50) AS query,
calls, bucket, bucket_start_time, queryid, client_ip
```

```
FROM pg_stat_monitor, pg_database
WHERE pg_database.oid = oid;
```

Output:

```
  userid  | datname  |                        query                        | calls | bucket |
bucket_start_time  |     queryid      | client_ip
----------+----------+-----------------------------------------------------+-------+--------
+---------------------+------------------+-----------
 postgres | postgres | SELECT name,description FROM pg_stat_monitor_sett |     1 |      9 |
2022-10-24 07:29:00 | AD536A8DEA7F0C73 | 127.0.0.1
 postgres | postgres | SELECT c.oid,                                      +|     1 |      9 |
2022-10-24 07:29:00 | 34B888E5C844519C | 127.0.0.1
          |          |   n.nspname,                                       +|       |
|                     |                  |
          |          |   c.relname                                        +|       |
|                     |                  |
          |          | FROM pg_ca                                          |       |
|                     |                  |
 postgres | postgres | SELECT DISTINCT userid::regrole, pg_stat_monitor. |     1 |      1 |
2022-10-24 07:31:00 | 6230793895381F1D | 127.0.0.1
 postgres | postgres | SELECT pg_stat_monitor_version()                    |     1 |      9 |
2022-10-24 07:29:00 | B617F5F12931F388 | 127.0.0.1
 postgres | postgres | CREATE EXTENSION pg_stat_monitor                    |     1 |      8 |
2022-10-24 07:28:00 | 14B98AF0776BAF7B | 127.0.0.1
 postgres | postgres | SELECT a.attname,                                  +|     1 |      9 |
2022-10-24 07:29:00 | 96F8E4B589EF148F | 127.0.0.1
          |          |   pg_catalog.format_type(a.att                      |       |
|                     |                  |
 postgres | postgres | SELECT c.relchecks, c.relkind, c.relhasindex, c.r |     1 |      9 |
2022-10-24 07:29:00 | CCC51D018AC96A25 | 127.0.0.1
```

Find more usage examples in the `pg_stat_monitor` user guide.

4.1.5 Changing the configuration

Run the following query to list available configuration parameters.

```
SELECT name, short_desc FROM pg_settings WHERE name LIKE '%pg_stat_monitor%';
```

Output

```
                   name
|                                                            short_desc
-------------------------------------------
+-----------------------------------------------------------------------------------------
 pg_stat_monitor.pgsm_bucket_time          | Sets the time in seconds per bucket.
 pg_stat_monitor.pgsm_enable_overflow      | Enable/Disable pg_stat_monitor to grow beyond
shared memory into swap space.
 pg_stat_monitor.pgsm_enable_pgsm_query_id | Enable/disable PGSM specific query id
calculation which is very useful in comparing same query across databases and clusters..
 pg_stat_monitor.pgsm_enable_query_plan    | Enable/Disable query plan monitoring.
 pg_stat_monitor.pgsm_extract_comments     | Enable/Disable extracting comments from
queries.
 pg_stat_monitor.pgsm_histogram_buckets    | Sets the maximum number of histogram buckets.
 pg_stat_monitor.pgsm_histogram_max        | Sets the time in millisecond.
 pg_stat_monitor.pgsm_histogram_min        | Sets the time in millisecond.
 pg_stat_monitor.pgsm_max                  | Sets the maximum size of shared memory in (MB)
used for statement's metadata tracked by pg_stat_monitor.
 pg_stat_monitor.pgsm_max_buckets          | Sets the maximum number of buckets.
 pg_stat_monitor.pgsm_normalized_query     | Selects whether save query in normalized
format.
 pg_stat_monitor.pgsm_overflow_target      | Sets the overflow target for pg_stat_monitor.
(Deprecated, use pgsm_enable_overflow)
 pg_stat_monitor.pgsm_query_max_len        | Sets the maximum length of query.
 pg_stat_monitor.pgsm_query_shared_buffer  | Sets the maximum size of shared memory in (MB)
used for query tracked by pg_stat_monitor.
 pg_stat_monitor.pgsm_track                | Selects which statements are tracked by
pg_stat_monitor.
 pg_stat_monitor.pgsm_track_planning       | Selects whether planning statistics are
tracked.
 pg_stat_monitor.pgsm_track_utility        | Selects whether utility commands are tracked.
```

You can change a parameter by setting a new value in the configuration file. Some parameters require server restart to apply a new value. For others, configuration reload is enough. Refer to the configuration parameters of the `pg_stat_monitor` documentation for the parameters' description, how you can change their values and if the server restart is required to apply them.

As an example, let's set the bucket lifetime from default 60 seconds to 40 seconds. Use the **ALTER SYSTEM** command:

```
ALTER SYSTEM set pg_stat_monitor.pgsm_bucket_time = 40;
```

Restart the server to apply the change:

On Debian and Ubuntu **On Red Hat Enterprise Linux and derivatives**

```
$ sudo systemctl restart postgresql.service

$ sudo systemctl restart postgresql-15
```

Verify the updated parameter:

```
SELECT name, setting
FROM pg_settings
WHERE name = 'pg_stat_monitor.pgsm_bucket_time';

                 name                | setting
  ----------------------------------+---------
   pg_stat_monitor.pgsm_bucket_time |   40
```

See also

`pg_stat_monitor` Documentation

Percona Blog:

- pg_stat_monitor: A New Way Of Looking At PostgreSQL Metrics
- Improve PostgreSQL Query Performance Insights with pg_stat_monitor

CONTACT US

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: May 22, 2023

Created: June 4, 2021

5. Solutions

5.1 High availability

5.1.1 High Availability in PostgreSQL with Patroni

PostgreSQL has been widely adopted as a modern, high-performance transactional database. A highly available PostgreSQL cluster can withstand failures caused by network outages, resource saturation, hardware failures, operating system crashes or unexpected reboots. Such cluster is often a critical component of the enterprise application landscape, where four nines of availability is a minimum requirement.

There are several methods to achieve high availability in PostgreSQL. This solution document provides Patroni - the open-source extension to facilitate and manage the deployment of high availability in PostgreSQL.

Hi availability methods

There are several native methods for achieving high availability with PostgreSQL:

- shared disk failover,
- file system replication,
- trigger-based replication,
- statement-based replication,
- logical replication,
- Write-Ahead Log (WAL) shipping, and
- streaming replication

Streaming replication

Streaming replication is part of Write-Ahead Log shipping, where changes to the WALs are immediately made available to standby replicas. With this approach, a standby instance is always up-to-date with changes from the primary node and can assume the role of primary in case of a failover.

WHY NATIVE STREAMING REPLICATION IS NOT ENOUGH

Although the native streaming replication in PostgreSQL supports failing over to the primary node, it lacks some key features expected from a truly highly-available solution. These include:

- No consensus-based promotion of a “leader” node during a failover
- No decent capability for monitoring cluster status
- No automated way to bring back the failed primary node to the cluster
- A manual or scheduled switchover is not easy to manage

To address these shortcomings, there are a multitude of third-party, open-source extensions for PostgreSQL. The challenge for a database administrator here is to select the right utility for the current scenario.

Percona Distribution for PostgreSQL solves this challenge by providing the Patroni extension for achieving PostgreSQL high availability.

Patroni

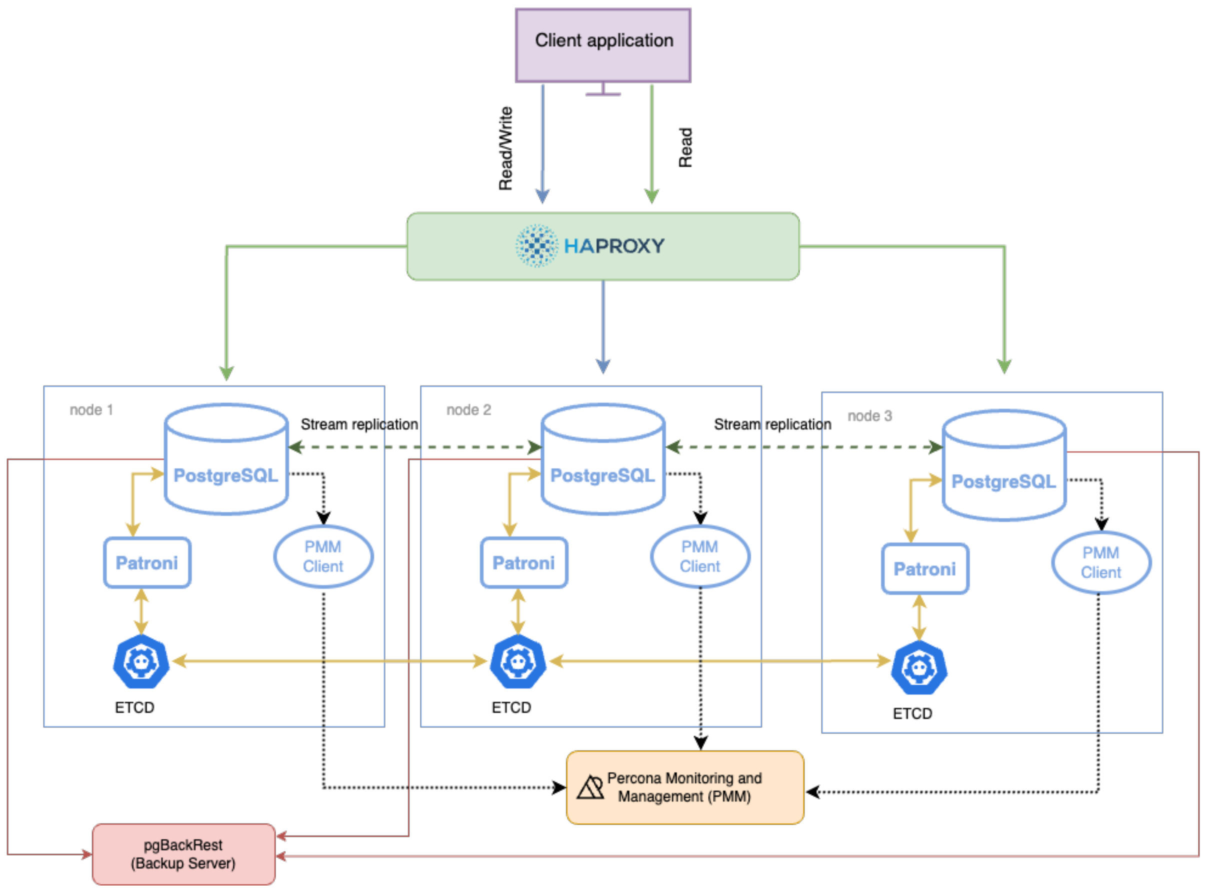
Patroni is a template for you to create your own customized, high-availability solution using Python and - for maximum accessibility - a distributed configuration store like ZooKeeper, etcd, Consul or Kubernetes.

KEY BENEFITS OF PATRONI:

- Continuous monitoring and automatic failover
- Manual/scheduled switchover with a single command
- Built-in automation for bringing back a failed node to cluster again.
- REST APIs for entire cluster configuration and further tooling.
- Provides infrastructure for transparent application failover
- Distributed consensus for every action and configuration.
- Integration with Linux watchdog for avoiding split-brain syndrome.

Architecture layout

The following diagram shows the architecture of a three-node PostgreSQL cluster with a single-leader node.



COMPONENTS

The components in this architecture are:

- PostgreSQL nodes
- Patroni - a template for configuring a highly available PostgreSQL cluster.
- ETCD - a Distributed Configuration store that stores the state of the PostgreSQL cluster.
- HAProxy - the load balancer for the cluster and is the single point of entry to client applications.
- pgBackRest - the backup and restore solution for PostgreSQL
- Percona Monitoring and Management (PMM) - the solution to monitor the health of your cluster

HOW COMPONENTS WORK TOGETHER

Each PostgreSQL instance in the cluster maintains consistency with other members through streaming replication. Each instance hosts Patroni - a cluster manager that monitors the cluster health. Patroni relies on the operational ETCD cluster to store the cluster configuration and sensitive data about the cluster health there.

Patroni periodically sends heartbeat requests with the cluster status to ETCD. ETCD writes this information to disk and sends the response back to Patroni. If the current primary fails to renew its status as leader within the specified timeout, Patroni updates the state change in ETCD, which uses this information to elect the new primary and keep the cluster up and running.

The connections to the cluster do not happen directly to the database nodes but are routed via a connection proxy like HAProxy. This proxy determines the active node by querying the Patroni REST API.

Next steps

Deploy on Debian or Ubuntu **Deploy on RHEL or derivatives**

Last update: November 1, 2023
Created: December 15, 2021

5.1.2 Deploying PostgreSQL for high availability with Patroni on Debian or Ubuntu

This guide provides instructions on how to set up a highly available PostgreSQL cluster with Patroni on Debian or Ubuntu.

Considerations

1. This is the example deployment suitable to be used for testing purposes in non-production environments.
2. In this setup ETCD resides on the same hosts as Patroni. In production, consider deploying ETCD cluster on dedicated hosts or at least have separate disks for ETCD and PostgreSQL. This is because ETCD writes every request from the cluster to disk which can be CPU intensive and affects disk performance. See hardware recommendations for details.
3. For this setup, we will use the nodes running on Ubuntu 22.04 as the base operating system:

Node name	Application	IP address
node1	Patroni, PostgreSQL, ETCD	10.104.0.1
node2	Patroni, PostgreSQL, ETCD	10.104.0.2
node3	Patroni, PostgreSQL, ETCD	10.104.0.3
HAProxy-demo	HAProxy	10.104.0.6

Note

Ideally, in a production (or even non-production) setup, the PostgreSQL nodes will be within a private subnet without any public connectivity to the Internet, and the HAProxy will be in a different subnet that allows client traffic coming only from a selected IP range. To keep things simple, we have implemented this architecture in a private environment, and each node can access the other by its internal, private IP.

Initial setup

SET UP HOSTNAMES IN THE `/ETC/HOSTS` FILE

It’s not necessary to have name resolution, but it makes the whole setup more readable and less error prone. Here, instead of configuring a DNS, we use a local name resolution by updating the file `/etc/hosts`. By resolving their hostnames to their IP addresses, we make the nodes aware of each other's names and allow their seamless communication.

1. Run the following command on each node. Change the node name to `node1`, `node2` and `node3` respectively:

```
$ sudo hostnamectl set-hostname node-1
```

2. Modify the `/etc/hosts` file of each PostgreSQL node to include the hostnames and IP addresses of the remaining nodes. Add the following at the end of the `/etc/hosts` file on all nodes:

node1 **node2** **node3** **HAproxy-demo**

```
# Cluster IP and names
10.104.0.1 node1
10.104.0.2 node2
10.104.0.3 node3

# Cluster IP and names
10.104.0.1 node1
10.104.0.2 node2
10.104.0.3 node3

# Cluster IP and names
10.104.0.1 node1
10.104.0.2 node2
10.104.0.3 node3
```

The HAProxy instance should have the name resolution for all the three nodes in its `/etc/hosts` file. Add the following lines at the end of the file:

```
# Cluster IP and names
10.104.0.6 HAProxy-demo
10.104.0.1 node1
10.104.0.2 node2
10.104.0.3 node3
```

INSTALL THE SOFTWARE

Run the following commands on node1 `,` node2 `and` node3`:

1. Install Percona Distribution for PostgreSQL

- Install `percona-release`.
- Enable the repository:

```
$ sudo percona-release setup ppg15
```

* Install Percona Distribution for PostgreSQL packages.

2. Install some Python and auxiliary packages to help with Patroni and ETCD

```
{.bash data-prompt="$"}
$ sudo apt install python3-pip python3-dev binutils
```

3. Install ETCD, Patroni, pgBackRest packages:

```
$ sudo apt install percona-patroni \
etcd etcd-server etcd-client \
percona-pgbackrest
```

4. Stop and disable all installed services:

```
$ sudo systemctl stop {etcd,patroni,postgresql}
$ systemctl disable {etcd,patroni,postgresql}
```

5. Even though Patroni can use an existing Postgres installation, remove the data directory to force it to initialize a new Postgres cluster instance.

```
$ sudo rm -rf /var/lib/postgresql/15/main
```

Configure ETCD distributed store

The distributed configuration store provides a reliable way to store data that needs to be accessed by large scale distributed systems. The most popular implementation of the distributed configuration store is ETCD. ETCD is deployed as a cluster for fault-tolerance and requires an odd number of members (n/2+1) to agree on updates to the cluster state. An ETCD cluster helps establish a consensus among nodes during a failover and manages the configuration for the three PostgreSQL instances.

The `etcd` cluster is first started in one node and then the subsequent nodes are added to the first node using the `add` command. The configuration is stored in the `/etc/default/etcd` file.

CONFIGURE `NODE1`

1. Back up the configuration file

```
$ sudo mv /etc/default/etcd /etc/default/etcd.orig
```

2. Export environment variables to simplify the config file creation

- Node name:

```
$ export NODE_NAME=`hostname -f`
```

- Node IP:

```
$ export NODE_IP=`hostname -i | awk '{print $1}'`
```

- Initial cluster token for the ETCD cluster during bootstrap:

```
$ export ETCD_TOKEN='PostgreSQL_HA_Cluster_1'
```

- ETCD data directory:

```
$ export ETCD_DATA_DIR='/var/lib/etcd/postgresql'
```

3. Modify the `/etc/default/etcd` configuration file as follows:.

```
ETCD_NAME=${NODE_NAME}
ETCD_INITIAL_CLUSTER="${NODE_NAME}=http://${NODE_IP}:2380"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="${ETCD_TOKEN}"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://${NODE_IP}:2380"
ETCD_DATA_DIR="${ETCD_DATA_DIR}"
ETCD_LISTEN_PEER_URLS="http://${NODE_IP}:2380"
ETCD_LISTEN_CLIENT_URLS="http://${NODE_IP}:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://${NODE_IP}:2379"
…
```

4. Start the `etcd` service to apply the changes on `node1`.

```
$ sudo systemctl enable --now etcd
$ sudo systemctl start etcd
$ sudo systemctl status etcd
```

5. Check the etcd cluster members on `node1`:

```
$ sudo etcdctl member list
```

Sample output:

```
21d50d7f768f153a: name=default peerURLs=http://10.104.0.1:2380 clientURLs=http://
10.104.0.1:2379 isLeader=true
```

6. Add the `node2` to the cluster. Run the following command on `node1`:

```
$ sudo etcdctl member add node2 http://10.104.0.2:2380
```

The output resembles the following one:

```
Added member named node2 with ID 10042578c504d052 to cluster

ETCD_NAME="node2"
ETCD_INITIAL_CLUSTER="node2=http://10.104.0.2:2380,node1=http://10.104.0.1:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

CONFIGURE `NODE2`

1. Back up the configuration file and export environment variables as described in steps 1-2 of the `node1` configuration
2. Edit the `/etc/default/etcd` configuration file on `node2`. Use the result of the `add` command on `node1` to change the configuration file as follows:

```
ETCD_NAME=${NODE_NAME}
ETCD_INITIAL_CLUSTER="node-1=http://10.0.100.1:2380,node-2=http://10.0.100.2:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"

ETCD_INITIAL_CLUSTER_TOKEN="${ETCD_TOKEN}"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://${NODE_IP}:2380"
ETCD_DATA_DIR="${ETCD_DATA_DIR}"
ETCD_LISTEN_PEER_URLS="http://${NODE_IP}:2380"
ETCD_LISTEN_CLIENT_URLS="http://${NODE_IP}:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://${NODE_IP}:2379"
```

3. Start the `etcd` service to apply the changes on `node2`:

```
$ sudo systemctl enable --now etcd
$ sudo systemctl start etcd
$ sudo systemctl status etcd
```

CONFIGURE `NODE3`

1. Add `node3` to the cluster. **Run the following command on** `node1`

```
$ sudo etcdctl member add node3 http://10.104.0.3:2380
```

2. On `node3`, back up the configuration file and export environment variables as described in steps 1-2 of the `node1` configuration
3. Modify the `/etc/default/etcd` configuration file and add the output of the `add` command:

```
ETCD_NAME=${NODE_NAME}
ETCD_INITIAL_CLUSTER="node1=http://10.104.0.1:2380,node2=http://10.104.0.2:2380,node3=http://
10.104.0.3:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"

ETCD_INITIAL_CLUSTER_TOKEN="${ETCD_TOKEN}"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://${NODE_IP}:2380"
ETCD_DATA_DIR="${ETCD_DATA_DIR}"
ETCD_LISTEN_PEER_URLS="http://${NODE_IP}:2380"
ETCD_LISTEN_CLIENT_URLS="http://${NODE_IP}:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://${NODE_IP}:2379"
…
```

4. Start the `etcd` service on `node3`:

```
$ sudo systemctl enable --now etcd
$ sudo systemctl start etcd
$ sudo systemctl status etcd
```

5. Check the etcd cluster members.

```
$ sudo etcdctl member list
```

The output resembles the following:

```
2d346bd3ae7f07c4: name=node2 peerURLs=http://10.104.0.2:2380 clientURLs=http://10.104.0.2:2379
isLeader=false
8bacb519ebdee8db: name=node3 peerURLs=http://10.104.0.3:2380 clientURLs=http://10.104.0.3:2379
isLeader=false
c5f52ea2ade25e1b: name=node1 peerURLs=http://10.104.0.1:2380 clientURLs=http://10.104.0.1:2379
isLeader=true
```

Configure Patroni

Run the following commands on all nodes. You can do this in parallel:

1. Export and create environment variables to simplify the config file creation:

- Node name:

```
$ export NODE_NAME=`hostname -f`
```

- Node IP:

```
$ export NODE_IP=`hostname -i | awk '{print $1}'`
```

- Create variables to store the PATH:

```
DATA_DIR="/var/lib/postgresql/15/main"
PG_BIN_DIR="/usr/lib/postgresql/15/bin"
```

NOTE: Check the path to the data and bin folders on your operating system and change it for the variables accordingly.

- Patroni information:

```
NAMESPACE="percona_lab"
SCOPE="cluster_1
```

2. Create the `/etc/patroni/patroni.yml` configuration file and add the following configuration for `node1`:

/etc/patroni/patroni.yml

```
namespace: ${NAMESPACE}
scope: ${SCOPE}
name: ${NODE_NAME}

restapi:
    listen: 0.0.0.0:8008
    connect_address: ${NODE_IP}:8008

etcd:
    host: ${NODE_IP}:2379

bootstrap:
  # this section will be written into Etcd:/<namespace>/<scope>/config after initializing new
cluster
  dcs:
      ttl: 30
      loop_wait: 10
      retry_timeout: 10
      maximum_lag_on_failover: 1048576
      slots:
          percona_cluster_1:
          type: physical

      postgresql:
          use_pg_rewind: true
          use_slots: true
          parameters:
              wal_level: replica
              hot_standby: "on"
              wal_keep_segments: 10
              max_wal_senders: 5
              max_replication_slots: 10
              wal_log_hints: "on"
              logging_collector: 'on'

  # some desired options for 'initdb'
  initdb: # Note: It needs to be a list (some options need values, others are switches)
      - encoding: UTF8
      - data-checksums

  pg_hba: # Add following lines to pg_hba.conf after running 'initdb'
      - host replication replicator 127.0.0.1/32 trust
      - host replication replicator 0.0.0.0/0 md5
      - host all all 0.0.0.0/0 md5
      - host all all ::0/0 md5

  # Some additional users which needs to be created after initializing new cluster
  users:
      admin:
          password: qaz123
          options:
              - createrole
              - createdb
      percona:
          password: qaz123
          options:
              - createrole
              - createdb

postgresql:
    cluster_name: cluster_1
    listen: 0.0.0.0:5432
    connect_address: ${NODE_IP}:5432
    data_dir: ${DATADIR}
    bin_dir: ${PG_BIN_DIR}
    pgpass: /tmp/pgpass
    authentication:
        replication:
            username: replicator
            password: replPasswd
        superuser:
            username: postgres
            password: qaz123
    parameters:
        unix_socket_directories: "/var/run/postgresql/"
    create_replica_methods:
        - basebackup
    basebackup:
        checkpoint: 'fast'

tags:
    nofailover: false
    noloadbalance: false
    clonefrom: false
    nosync: false
```

Patroni configuration file

Let's take a moment to understand the contents of the `patroni.yml` file.

The first section provides the details of the node and its connection ports. After that, we have the `etcd` service and its port details.

Following these, there is a `bootstrap` section that contains the PostgreSQL configurations and the steps to run once the database is initialized. The `pg_hba.conf` entries specify all the other nodes that can connect to this node and their authentication mechanism.

3. Check that the systemd unit file `patroni.service` is created in `/etc/systemd/system`. If it is created, skip this step.

   If it's **not** created, create it manually and specify the following contents within:

   ~~~
   ```ini title="/etc/systemd/system/patroni.service"
   [Unit]
   Description=Runners to orchestrate a high-availability PostgreSQL
   After=syslog.target network.target

   [Service]
   Type=simple

   User=postgres
   Group=postgres

   # Start the patroni process
   ExecStart=/bin/patroni /etc/patroni/patroni.yml

   # Send HUP to reload from patroni.yml
   ExecReload=/bin/kill -s HUP $MAINPID

   # only kill the patroni process, not its children, so it will gracefully stop postgres
   KillMode=process

   # Give a reasonable amount of time for the server to start up/shut down
   TimeoutSec=30

   # Do not restart the service if it crashes, we want to manually inspect database on failure
   Restart=no

   [Install]
   WantedBy=multi-user.target
   ```
   ~~~

1. Make systemd aware of the new service:

   ```
   $ sudo systemctl daemon-reload
   ```

2. Now it's time to start Patroni. You need the following commands on all nodes but not in parallel. Start with the `node1` first, wait for the service to come to live, and then proceed with the other nodes one-by-one, always waiting for them to sync with the primary node:

   ```
   $ sudo systemctl enable --now patroni
   $ sudo systemctl restart patroni
   ```

When Patroni starts, it initializes PostgreSQL (because the service is not currently running and the data directory is empty) following the directives in the bootstrap section of the configuration file.

1. Check the service to see if there are errors:

   ```
   $ sudo journalctl -fu patroni
   ```

   A common error is Patroni complaining about the lack of proper entries in the pg_hba.conf file. If you see such errors, you must manually add or fix the entries in that file and then restart the service.

   Changing the patroni.yml file and restarting the service will not have any effect here because the bootstrap section specifies the configuration to apply when PostgreSQL is first started in the node. It will not repeat the process even if the Patroni configuration file is modified and the service is restarted.

2. Check the cluster:

   ```
   $ patronictl -c /etc/patroni/patroni.yml list $SCOPE
   ```

   The output on `node1` resembles the following:

   ```
   + Cluster: cluster_1 --+---------+---------+----+-----------+
   | Member | Host        | Role    | State   | TL | Lag in MB |
   +--------+------------+---------+---------+----+-----------+
   | node-1 | 10.0.100.1  | Leader  | running |  1 |           |
   +--------+------------+---------+---------+----+-----------+
   ```

   On the remaining nodes:

   ```
   + Cluster: cluster_1 --+---------+---------+----+-----------+
   | Member | Host        | Role    | State   | TL | Lag in MB |
   +--------+------------+---------+---------+----+-----------+
   | node-1 | 10.0.100.1  | Leader  | running |  1 |           |
   | node-2 | 10.0.100.2  | Replica | running |  1 |         0 |
   +--------+------------+---------+---------+----+-----------+
   ```

If Patroni has started properly, you should be able to locally connect to a PostgreSQL node using the following command:

```
$ sudo psql -U postgres
```

The command output is the following:

```
psql (15.4)
Type "help" for help.

postgres=#
```

Configure HAProxy

HAproxy is the load balancer and the single point of entry to your PostgreSQL cluster for client applications. A client application accesses the HAPpoxy URL and sends its read/write requests there. Behind-the-scene, HAProxy routes write requests to the primary node and read requests - to the secondaries in a round-robin fashion so that no secondary instance is unnecessarily loaded. To make this happen, provide different ports in the HAProxy configuration file. In this deployment, writes are routed to port 5000 and reads - to port 5001

This way, a client application doesn't know what node in the underlying cluster is the current primary. HAProxy sends connections to a healthy node (as long as there is at least one healthy node available) and ensures that client application requests are never rejected.

1. Install HAProxy on the `HAProxy-demo` node:

```
$ sudo apt install percona-haproxy
```

2. The HAProxy configuration file path is: `/etc/haproxy/haproxy.cfg`. Specify the following configuration in this file.

```
global
    maxconn 100

defaults
    log global
    mode tcp
    retries 2
    timeout client 30m
    timeout connect 4s
    timeout server 30m
    timeout check 5s

listen stats
    mode http
    bind *:7000
    stats enable
    stats uri /

listen primary
    bind *:5000
    option httpchk /primary
    http-check expect status 200
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
    server node1 node1:5432 maxconn 100 check port 8008
    server node2 node2:5432 maxconn 100 check port 8008
    server node3 node3:5432 maxconn 100 check port 8008

listen standbys
    balance roundrobin
    bind *:5001
    option httpchk /replica
    http-check expect status 200
    default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
    server node1 node1:5432 maxconn 100 check port 8008
    server node2 node2:5432 maxconn 100 check port 8008
    server node3 node3:5432 maxconn 100 check port 8008
```

HAProxy will use the REST APIs hosted by Patroni to check the health status of each PostgreSQL node and route the requests appropriately.

3. Restart HAProxy:

```
$ sudo systemctl restart haproxy
```

4. Check the HAProxy logs to see if there are any errors:

```
$ sudo journalctl -u haproxy.service -n 100 -f
```

Next steps

Configure pgBackRest

Last update: November 1, 2023
Created: December 15, 2021

5.1.3 Deploying PostgreSQL for high availability with Patroni on RHEL or CentOS

This guide provides instructions on how to set up a highly available PostgreSQL cluster with Patroni on Red Hat Enterprise Linux or CentOS.

Considerations

1. This is the example deployment suitable to be used for testing purposes in non-production environments.
2. In this setup ETCD resides on the same hosts as Patroni. In production, consider deploying ETCD cluster on dedicated hosts because ETCD writes every request from the cluster to disk which requires significant amount of disk space. See hardware recommendations for details.
3. For this setup, we use the nodes running on Red Hat Enterprise Linux 8 as the base operating system:

Node name	Application	IP address
node1	Patroni, PostgreSQL, ETCD	10.104.0.1
node2	Patroni, PostgreSQL, ETCD	10.104.0.2
node3	Patroni, PostgreSQL, ETCD	10.104.0.3
HAProxy-demo	HAProxy	10.104.0.6

Note

Ideally, in a production (or even non-production) setup, the PostgreSQL and ETCD nodes will be within a private subnet without any public connectivity to the Internet, and the HAProxy will be in a different subnet that allows client traffic coming only from a selected IP range. To keep things simple, we have implemented this architecture in a private environment, and each node can access the other by its internal, private IP.

Initial setup

SET UP HOSTNAMES IN THE `/ETC/HOSTS` FILE

It's not necessary to have name resolution, but it makes the whole setup more readable and less error prone. Here, instead of configuring a DNS, we use a local name resolution by updating the file `/etc/hosts`. By resolving their hostnames to their IP addresses, we make the nodes aware of each other's names and allow their seamless communication.

1. Run the following command on each node. Change the node name to `node1`, `node2` and `node3` respectively:

```
$ sudo hostnamectl set-hostname node-1
```

2. Modify the `/etc/hosts` file of each PostgreSQL node to include the hostnames and IP addresses of the remaining nodes. Add the following at the end of the `/etc/hosts` file on all nodes:

node1 **node2** **node3** **HAproxy-demo**

```
# Cluster IP and names
10.104.0.1 node1
10.104.0.2 node2
10.104.0.3 node3
```

```
# Cluster IP and names
10.104.0.1 node1
10.104.0.2 node2
10.104.0.3 node3
```

```
# Cluster IP and names
10.104.0.1 node1
10.104.0.2 node2
10.104.0.3 node3
```

The HAProxy instance should have the name resolution for all the three nodes in its `/etc/hosts` file. Add the following lines at the end of the file:

```
# Cluster IP and names
10.104.0.6 HAProxy-demo
10.104.0.1 node1
10.104.0.2 node2
10.104.0.3 node3
```

INSTALL THE SOFTWARE

1. Install Percona Distribution for PostgreSQL on `node1`, `node2` and `node3` from Percona repository:

- Install `percona-release`.
- Enable the repository:

```
$ sudo percona-release setup ppg15
```

- Install Percona Distribution for PostgreSQL packages.

> **Important**
>
> **Don't** initialize the cluster and start the `postgresql` service. The cluster initialization and setup are handled by Patroni during the bootsrapping stage.

2. Install some Python and auxiliary packages to help with Patroni and ETCD

```
$ sudo yum install python3-pip python3-dev binutils
```

3. Install ETCD, Patroni, pgBackRest packages:

```
$ sudo yum install percona-patroni \
etcd python3-python-etcd\
percona-pgbackrest
```

4. Stop and disable all installed services:

```
$ sudo systemctl stop {etcd,patroni,postgresql}
$ systemctl disable {etcd,patroni,postgresql}
```

Configure ETCD distributed store

The distributed configuration store provides a reliable way to store data that needs to be accessed by large scale distributed systems. The most popular implementation of the distributed configuration store is ETCD. ETCD is deployed as a cluster for fault-tolerance and requires an odd number of members (n/2+1) to agree on updates to the cluster state. An ETCD cluster helps establish a consensus among nodes during a failover and manages the configuration for the three PostgreSQL instances.

The `etcd` cluster is first started in one node and then the subsequent nodes are added to the first node using the `add` command. The configuration is stored in the `/etc/etcd/etcd.conf` configuration file.

CONFIGURE `NODE1`

1. Backup the `etcd.conf` file:

```
$ sudo mv /etc/etcd/etcd.conf /etc/etcd/etcd.conf.orig
```

2. Export environment variables to simplify the config file creation

- Node name:

```
$ export NODE_NAME=`hostname -f`
```

- Node IP:

```
$ export NODE_IP=`hostname -i | awk '{print $1}'`
```

- Initial cluster token for the ETCD cluster during bootstrap:

```
$ export ETCD_TOKEN='PostgreSQL_HA_Cluster_1'
```

- ETCD data directory:

```
$ export ETCD_DATA_DIR='/var/lib/etcd/postgresql'
```

3. Modify the `/etc/etcd/etcd.conf` configuration file:

```
ETCD_NAME=${NODE_NAME}
ETCD_INITIAL_CLUSTER="${NODE_NAME}=http://${NODE_IP}:2380"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="${ETCD_TOKEN}"
```

```
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://${NODE_IP}:2380"
ETCD_DATA_DIR="${ETCD_DATA_DIR}"
ETCD_LISTEN_PEER_URLS="http://${NODE_IP}:2380"
ETCD_LISTEN_CLIENT_URLS="http://${NODE_IP}:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://${NODE_IP}:2379"
…
```

4. Start the `etcd` to apply the changes on `node1`:

```
$ sudo systemctl enable --now etcd
$ sudo systemctl start etcd
$ sudo systemctl status etcd
```

5. Check the etcd cluster members on `node1`.

```
$ sudo etcdctl member list
```

The output resembles the following:

```
21d50d7f768f153a: name=default peerURLs=http://10.104.0.5:2380 clientURLs=http://
10.104.0.5:2379 isLeader=true
```

6. Add `node2` to the cluster. Run the following command on `node1`:

```
$ sudo etcdctl member add node2 http://10.104.0.2:2380
```

The output resembles the following one:

```
Added member named node2 with ID 10042578c504d052 to cluster

ETCD_NAME="node2"
ETCD_INITIAL_CLUSTER="node2=http://10.104.0.2:2380,node1=http://10.104.0.1:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"
```

CONFIGURE `NODE2`

1. Back up the configuration file and export environment variables as described in steps 1-2 of the `node1` configuration
2. Edit the `/etc/etcd/etcd.conf` configuration file on `node2` and add the output from the `add` command:

```
[Member]
ETCD_NAME=${NODE_NAME}
ETCD_INITIAL_CLUSTER="node-1=http://10.0.100.1:2380,node-2=http://10.0.100.2:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"       ETCD_INITIAL_CLUSTER_TOKEN="${ETCD_TOKEN}"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://${NODE_IP}:2380"
ETCD_DATA_DIR="${ETCD_DATA_DIR}"
ETCD_LISTEN_PEER_URLS="http://${NODE_IP}:2380"
ETCD_LISTEN_CLIENT_URLS="http://${NODE_IP}:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://${NODE_IP}:2379"
```

3. Start the `etcd` to apply the changes on `node2`:

```
$ sudo systemctl enable --now etcd
$ sudo systemctl start etcd
$ sudo systemctl status etcd
```

CONFIGURE `NODE3`

1. Add `node3` to the cluster. **Run the following command on** `node1`:

```
$ sudo etcdctl member add node3 http://10.104.0.3:2380
```

2. On `node3`, back up the configuration file and export environment variables as described in steps 1-2 of the `node1` configuration
3. Modify the `/etc/etcd/etcd.conf` configuration file on `node3` and add the output from the `add` command as follows:

```
ETCD_NAME=${NODE_NAME}
ETCD_INITIAL_CLUSTER="node1=http://10.104.0.1:2380,node2=http://10.104.0.2:2380,node3=http://
10.104.0.3:2380"
ETCD_INITIAL_CLUSTER_STATE="existing"

ETCD_INITIAL_CLUSTER_TOKEN="${ETCD_TOKEN}"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://${NODE_IP}:2380"
ETCD_DATA_DIR="${ETCD_DATA_DIR}"
ETCD_LISTEN_PEER_URLS="http://${NODE_IP}:2380"
ETCD_LISTEN_CLIENT_URLS="http://${NODE_IP}:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://${NODE_IP}:2379"
…
```

4. Start the `etcd` service on `node3`:

```
$ sudo systemctl enable --now etcd
$ sudo systemctl start etcd
$ sudo systemctl status etcd
```

5. Check the etcd cluster members.

```
$ sudo etcdctl member list
```

The output resembles the following:

```
2d346bd3ae7f07c4: name=node2 peerURLs=http://10.104.0.2:2380 clientURLs=http://10.104.0.2:2379
isLeader=false
8bacb519ebdee8db: name=node3 peerURLs=http://10.104.0.3:2380 clientURLs=http://10.104.0.3:2379
isLeader=false
c5f52ea2ade25e1b: name=node1 peerURLs=http://10.104.0.1:2380 clientURLs=http://10.104.0.1:2379
isLeader=true
```

Configure Patroni

Run the following commands on all nodes. You can do this in parallel:

1. Export and create environment variables to simplify the config file creation:

- Node name:

```
$ export NODE_NAME=`hostname -f`
```

- Node IP:

```
$ export NODE_IP=`hostname -i | awk '{print $1}'`
```

- Create variables to store the PATH:

```
DATA_DIR="/var/lib/pgsql/data/"
PG_BIN_DIR="/usr/pgsql-15/bin"
```

NOTE: Check the path to the data and bin folders on your operating system and change it for the variables accordingly.

- Patroni information:

```
NAMESPACE="percona_lab"
SCOPE="cluster_1
```

2. Create the directories required by Patroni

- Create the directory to store the configuration file and make it owned by the `postgres` user.

```
$ sudo mkdir -p /etc/patroni/
$ sudo chown -R  postgres:postgres /etc/patroni/
```

- Create the data directory to store PostgreSQL data. Change its ownership to the `postgres` user and restrict the access to it

```
$ sudo mkdir /data/pgsql -p
$ sudo chown -R postgres:postgres /data/pgsql
$ sudo chmod 700 /data/pgsql
```

3. Create the `/etc/patroni/patroni.yml` with the following configuration:

/etc/patroni/patroni.yml

```
namespace: ${NAMESPACE}
scope: ${SCOPE}
name: ${NODE_NAME}

restapi:
    listen: 0.0.0.0:8008
    connect_address: ${NODE_IP}:8008

etcd:
    host: ${NODE_IP}:2379

bootstrap:
  # this section will be written into Etcd:/<namespace>/<scope>/config after initializing new
cluster
  dcs:
      ttl: 30
      loop_wait: 10
      retry_timeout: 10
```

```
        maximum_lag_on_failover: 1048576
        slots:
            percona_cluster_1:
            type: physical

        postgresql:
            use_pg_rewind: true
            use_slots: true
            parameters:
                wal_level: replica
                hot_standby: "on"
                wal_keep_segments: 10
                max_wal_senders: 5
                max_replication_slots: 10
                wal_log_hints: "on"
                logging_collector: 'on'

    # some desired options for 'initdb'
    initdb: # Note: It needs to be a list (some options need values, others are switches)
        - encoding: UTF8
        - data-checksums

    pg_hba: # Add following lines to pg_hba.conf after running 'initdb'
        - host replication replicator 127.0.0.1/32 trust
        - host replication replicator 0.0.0.0/0 md5
        - host all all 0.0.0.0/0 md5
        - host all all ::0/0 md5

    # Some additional users which needs to be created after initializing new cluster
    users:
        admin:
            password: qaz123
            options:
                - createrole
                - createdb
        percona:
            password: qaz123
            options:
                - createrole
                - createdb

postgresql:
    cluster_name: cluster_1
    listen: 0.0.0.0:5432
    connect_address: ${NODE_IP}:5432
    data_dir: ${DATADIR}
    bin_dir: ${PG_BIN_DIR}
    pgpass: /tmp/pgpass
    authentication:
        replication:
            username: replicator
            password: replPasswd
        superuser:
            username: postgres
            password: qaz123
    parameters:
        unix_socket_directories: "/var/run/postgresql/"
    create_replica_methods:
        - basebackup
    basebackup:
        checkpoint: 'fast'

tags:
    nofailover: false
    noloadbalance: false
    clonefrom: false
    nosync: false
```

4. Check that the systemd unit file `patroni.service` is created in `/etc/systemd/system`. If it is created, skip this step.

   If it's **not** created, create it manually and specify the following contents within:

   /etc/systemd/system/patroni.service

```
[Unit]
Description=Runners to orchestrate a high-availability PostgreSQL
After=syslog.target network.target

[Service]
Type=simple

User=postgres
Group=postgres

# Start the patroni process
ExecStart=/bin/patroni /etc/patroni/patroni.yml

# Send HUP to reload from patroni.yml
ExecReload=/bin/kill -s HUP $MAINPID

# only kill the patroni process, not its children, so it will gracefully stop postgres
KillMode=process

# Give a reasonable amount of time for the server to start up/shut down
TimeoutSec=30

# Do not restart the service if it crashes, we want to manually inspect database on failure
Restart=no

[Install]
WantedBy=multi-user.target
```

5. Make `systemd` aware of the new service:

```
$ sudo systemctl daemon-reload
```

6. Now it's time to start Patroni. You need the following commands on all nodes but not in parallel. Start with the `node1` first, wait for the service to come to live, and then proceed with the other nodes one-by-one, always waiting for them to sync with the primary node:

```
$ sudo systemctl enable --now patroni
$ sudo systemctl restart patroni
```

When Patroni starts, it initializes PostgreSQL (because the service is not currently running and the data directory is empty) following the directives in the bootstrap section of the configuration file.

1. Check the service to see if there are errors:

   ```
   $ sudo journalctl -fu patroni
   ```

   A common error is Patroni complaining about the lack of proper entries in the pg_hba.conf file. If you see such errors, you must manually add or fix the entries in that file and then restart the service.

   Changing the patroni.yml file and restarting the service will not have any effect here because the bootstrap section specifies the configuration to apply when PostgreSQL is first started in the node. It will not repeat the process even if the Patroni configuration file is modified and the service is restarted.

   If Patroni has started properly, you should be able to locally connect to a PostgreSQL node using the following command:

   ```
   $ sudo psql -U postgres

   psql (15.4)
   Type "help" for help.

   postgres=#
   ```

2. When all nodes are up and running, you can check the cluster status using the following command:

   ```
   $ sudo patronictl -c /etc/patroni/patroni.yml list
   ```

   The output on `node1` resembles the following:

   ```
   + Cluster: cluster_1 --+---------+---------+----+-----------+
   | Member | Host        | Role    | State   | TL | Lag in MB |
   +--------+-------------+---------+---------+----+-----------+
   | node-1 | 10.0.100.1  | Leader  | running |  1 |           |
   +--------+-------------+---------+---------+----+-----------+
   ```

   On the remaining nodes:

   ```
   + Cluster: cluster_1 --+---------+---------+----+-----------+
   | Member | Host        | Role    | State   | TL | Lag in MB |
   +--------+-------------+---------+---------+----+-----------+
   | node-1 | 10.0.100.1  | Leader  | running |  1 |           |
   | node-2 | 10.0.100.2  | Replica | running |  1 |         0 |
   +--------+-------------+---------+---------+----+-----------+
   ```

Configure HAProxy

HAproxy is the load balancer and the single point of entry to your PostgreSQL cluster for client applications. A client application accesses the HAPpoxy URL and sends its read/write requests there. Behind-the-scene, HAProxy routes write requests to the primary node and read requests - to the secondaries in a round-robin fashion so that no secondary instance is unnecessarily loaded. To make this happen, provide different ports in the HAProxy configuration file. In this deployment, writes are routed to port 5000 and reads - to port 5001

This way, a client application doesn’t know what node in the underlying cluster is the current primary. HAProxy sends connections to a healthy node (as long as there is at least one healthy node available) and ensures that client application requests are never rejected.

1. Install HAProxy on the `HAProxy-demo` node:

   ```
   $ sudo yum install percona-haproxy
   ```

2. The HAProxy configuration file path is: `/etc/haproxy/haproxy.cfg`. Specify the following configuration in this file.

   ```
   global
       maxconn 100

   defaults
       log global
       mode tcp
       retries 2
       timeout client 30m
       timeout connect 4s
       timeout server 30m
       timeout check 5s

   listen stats
       mode http
       bind *:7000
       stats enable
       stats uri /

   listen primary
       bind *:5000
       option httpchk /primary
       http-check expect status 200
       default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
       server node1 node1:5432 maxconn 100 check port 8008
       server node2 node2:5432 maxconn 100 check port 8008
       server node3 node3:5432 maxconn 100 check port 8008

   listen standbys
       balance roundrobin
       bind *:5001
       option httpchk /replica
       http-check expect status 200
       default-server inter 3s fall 3 rise 2 on-marked-down shutdown-sessions
       server node1 node1:5432 maxconn 100 check port 8008
       server node2 node2:5432 maxconn 100 check port 8008
       server node3 node3:5432 maxconn 100 check port 8008
   ```

   HAProxy will use the REST APIs hosted by Patroni to check the health status of each PostgreSQL node and route the requests appropriately.

3. Enable a SELinux boolean to allow HAProxy to bind to non standard ports:

   ```
   $ sudo setsebool -P haproxy_connect_any on
   ```

4. Restart HAProxy:

   ```
   $ sudo systemctl restart haproxy
   ```

5. Check the HAProxy logs to see if there are any errors:

   ```
   $ sudo journalctl -u haproxy.service -n 100 -f
   ```

Next steps

Configure pgBackRest

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 1, 2023
Created: December 15, 2021

5.1.4 pgBackRest setup

pgBackRest is the backup tool used to perform Postgres database backup, restoration, and point-in-time recovery. It is a server-client application, where the server runs on a dedicated host and a client runs on every PostgreSQL node.

You also need a backup storage to store the backups. It can either be a remote storage such as AWS S3, S3-compatible storages or Azure blob storage, or a filesystem-based one.

Configure backup server

INSTALL PGBACKREST

1. Enable the repository with percona-release

```
$ sudo percona-release setup ppg-15
```

2. Install pgBackRest package

Debian/Ubuntu RHEL/derivatives

```
$ sudo apt install percona-pgbackrest

$ sudo yum install percona-pgbackrest
```

CREATE THE CONFIGURATION FILE

1. Create environment variables to simplify the config file creation:

```
export SRV_NAME="bkp-srv"
export NODE1_NAME="node-1"
export NODE2_NAME="node-2"
export NODE3_NAME="node-3"
```

2. Create the `pgBackRest` repository

A repository is where `pgBackRest` stores backups. In this example, the backups will be saved to `/var/lib/pgbackrest`

```
$ sudo mkdir -p /var/lib/pgbackrest
$ sudo chmod 750 /var/lib/pgbackrest
$ sudo chown postgres:postgres /var/lib/pgbackrest
```

3. The default pgBackRest configuration file location is `/etc/pgbackrest/pgbackrest.conf`. If it does not exist, then `/etc/pgbackrest.conf` is used next. Edit the `pgbackrest.conf` file to include the following configuration:

```
[global]

# Server repo details
repo1-path=/var/lib/pgbackrest

### Retention ###
#  - repo1-retention-archive-type
#  - If set to full pgBackRest will keep archive logs for the number of full backups defined
by repo-retention-archive
repo1-retention-archive-type=full
```

```
# repo1-retention-archive
#  - Number of backups worth of continuous WAL to retain
#  - NOTE: WAL segments required to make a backup consistent are always retained until the
backup is expired regardless of how this option is configured
#  - If this value is not set and repo-retention-full-type is count (default), then the
archive to expire will default to the repo-retention-full
# repo1-retention-archive=2

# repo1-retention-full
#  - Full backup retention count/time.
#  - When a full backup expires, all differential and incremental backups associated with the
full backup will also expire.
#  - When the option is not defined a warning will be issued.
#  - If indefinite retention is desired then set the option to the max value.
repo1-retention-full=4

# Server general options
process-max=12
log-level-console=info
#log-level-file=debug
log-level-file=info
start-fast=y
delta=y
backup-standby=y

########## Server TLS options ##########
tls-server-address=*
tls-server-cert-file=/pg_ha/certs/${SRV_NAME}.crt
tls-server-key-file=/pg_ha/certs/${SRV_NAME}.key
tls-server-ca-file=/pg_ha/certs/ca.crt

### Auth entry ###
tls-server-auth=${NODE1_NAME}=cluster_1
tls-server-auth=${NODE2_NAME}=cluster_1
tls-server-auth=${NODE3_NAME}=cluster_1

### Clusters and nodes ###
[cluster_1]
pg1-host=${NODE1_NAME}
pg1-host-port=8432
pg1-port=5432
pg1-path=/var/lib/postgresql/11/
pg1-host-type=tls
pg1-host-cert-file=/pg_ha/certs/${SRV_NAME}.crt
pg1-host-key-file=/pg_ha/certs/${SRV_NAME}.key
pg1-host-ca-file=/pg_ha/certs/ca.crt
pg1-socket-path=/var/run/postgresql


pg2-host=${NODE2_NAME}
pg2-host-port=8432
pg2-port=5432
pg2-path=/var/lib/postgresql/11/
pg2-host-type=tls
pg2-host-cert-file=/pg_ha/certs/${SRV_NAME}.crt
pg2-host-key-file=/pg_ha/certs/${SRV_NAME}.key
pg2-host-ca-file=/pg_ha/certs/ca.crt
pg2-socket-path=/var/run/postgresql

pg3-host=${NODE3_NAME}
pg3-host-port=8432
pg3-port=5432
```

```
pg3-path=/var/lib/postgresql/11/
pg3-host-type=tls
pg3-host-cert-file=/pg_ha/certs/${SRV_NAME}.crt
pg3-host-key-file=/pg_ha/certs/${SRV_NAME}.key
pg3-host-ca-file=/pg_ha/certs/ca.crt
pg3-socket-path=/var/run/postgresql
```

4. Create the `systemd` unit file at the path `/etc/systemd/system/pgbackrest.service`

/etc/systemd/system/pgbackrest.service

```
[Unit]
Description=pgBackRest Server
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
User=postgres
Restart=always
RestartSec=1
ExecStart=/usr/bin/pgbackrest server
#ExecStartPost=/bin/sleep 3
#ExecStartPost=/bin/bash -c "[ ! -z $MAINPID ]"
ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target
```

CREATE THE CERTIFICATE FILES

1. Create the folder where to store the certificates. For example, `/pg_ha/certs`
2. Define the variable for the certificates path:

```
export CA_PATH="/pg_ha/certs"
```

3. Create the certificates and keys

```
$ sudo -iu postgres openssl req -new -x509 -days 365 -nodes -out ${CA_PATH}/ca.crt -keyout $
{CA_PATH}/ca.key -subj "/CN=root-ca"
```

4. Create the certificate for the backup server

```
$ sudo -iu postgres openssl req -new -nodes -out ${CA_PATH}/${SRV_NAME}.csr -keyout $
{CA_PATH}/${SRV_NAME}.key -subj "/CN=${SRV_NAME}"
```

5. Create the certificates for each node: `node1`, `node2`, `node3`

```
$ sudo -iu postgres openssl req -new -nodes -out ${CA_PATH}/${NODE1_NAME}.csr -keyout $
{CA_PATH}/${NODE1_NAME}.key -subj "/CN=${NODE1_NAME}"
$ sudo -iu postgres openssl req -new -nodes -out ${CA_PATH}/${NODE2_NAME}.csr -keyout $
{CA_PATH}/${NODE2_NAME}.key -subj "/CN=${NODE2_NAME}"
$ sudo -iu postgres openssl req -new -nodes -out ${CA_PATH}/${NODE3_NAME}.csr -keyout $
{CA_PATH}/${NODE3_NAME}.key -subj "/CN=${NODE3_NAME}"
```

6. Sign the certificates with the `root-ca` key

```
$ sudo -iu postgres openssl x509 -req -in ${CA_PATH}/${SRV_NAME}.csr -days 365 -CA ${CA_PATH}/
ca.crt -CAkey ${CA_PATH}/ca.key -CAcreateserial -out ${CA_PATH}/${SRV_NAME}.crt
$ sudo -iu postgres openssl x509 -req -in ${CA_PATH}/${NODE1_NAME}.csr -days 365 -CA $
{CA_PATH}/ca.crt -CAkey ${CA_PATH}/ca.key -CAcreateserial -out ${CA_PATH}/${NODE1_NAME}.crt
$ sudo -iu postgres openssl x509 -req -in ${CA_PATH}/${NODE2_NAME}.csr -days 365 -CA $
{CA_PATH}/ca.crt -CAkey ${CA_PATH}/ca.key -CAcreateserial -out ${CA_PATH}/${NODE2_NAME}.crt
$ sudo -iu postgres openssl x509 -req -in ${CA_PATH}/${NODE3_NAME}.csr -days 365 -CA $
{CA_PATH}/ca.crt -CAkey ${CA_PATH}/ca.key -CAcreateserial -out ${CA_PATH}/${NODE3_NAME}.crt
```

7. Remove temporary files

```
$ rm ${CA_PATH}/*.csr
```

8. Reload, enable, and start the service

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now pgbackrest
```

Configure database servers

Run the following command on `node1`, `node2` and `node3`.

1. Create the certificates folder. For example, `/pg_ha/certs`

   ```
   $ sudo mkdir -p /pg_ha/certs
   ```

2. Export environment variables to simplify config file creation

   ```
   export NODE_NAME=`hostname -f`
   ```

3. Create the configuration file. The default path is `/etc/pgbackrest.conf`

   /etc/pgbackrest.conf

   ```
   [global]
   repo1-host=bkp-srv
   repo1-host-user=postgres
   repo1-host-type=tls
   repo1-host-cert-file=/pg_ha/certs/${NODE_NAME}.crt
   repo1-host-key-file=/pg_ha/certs/${NODE_NAME}.key
   repo1-host-ca-file=/pg_ha/certs/ca.crt

   # general options
   process-max=16
   log-level-console=info
   log-level-file=debug

   # tls server options
   tls-server-address=*
   tls-server-cert-file=/pg_ha/certs/${NODE_NAME}.crt
   tls-server-key-file=/pg_ha/certs/${NODE_NAME}.key
   tls-server-ca-file=/pg_ha/certs/ca.crt
   tls-server-auth=bkp-srv=cluster_1

   [cluster_1]
   pg1-path=/var/lib/postgresql/11
   ```

4. Create the `systemd` unit file at the path `/etc/systemd/system/pgbackrest.service`

   /etc/systemd/system/pgbackrest.service

   ```
   [Unit]
   Description=pgBackRest Server
   After=network.target
   StartLimitIntervalSec=0

   [Service]
   Type=simple
   User=postgres
   Restart=always
   RestartSec=1
   ExecStart=/usr/bin/pgbackrest server
   #ExecStartPost=/bin/sleep 3
   #ExecStartPost=/bin/bash -c "[ ! -z $MAINPID ]"
   ExecReload=/bin/kill -HUP $MAINPID

   [Install]
   WantedBy=multi-user.target
   ```

5. Reload, enable, and start the service

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now pgbackrest
```

6. Change Patroni configuration to use pgBackRest. Run this command on one node only, for example, on `node1`. Edit the `/etc/patroni/patroni.yml` file :

/etc/patroni/patroni.yml

```
loop_wait: 10
maximum_lag_on_failover: 1048576
postgresql:
  parameters:
    archive_command: pgbackrest --stanza=cluster_1 archive-push "/var/lib/postgresql/15/main/
pg_wal/%f"
    archive_mode: true
    archive_timeout: 1800s
    hot_standby: true
    logging_collector: 'on'
    max_replication_slots: 10
    max_wal_senders: 5
    wal_keep_size: 4096
    wal_level: logical
    wal_log_hints: true
  recovery_conf:
    recovery_target_timeline: latest
    restore_command: pgbackrest --config=/etc/pgbackrest.conf --stanza=cluster_1 archive-get
%f "%p"
  use_pg_rewind: true
  use_slots: true
retry_timeout: 10
slots:
  percona_cluster_1:
    type: physical
ttl: 30
```

Create backups

Run the following commands on the **backup server**

1. Create the stanza. A stanza is the configuration for a PostgreSQL database cluster that defines where it is located, how it will be backed up, archiving options, etc.

   ```
   $ sudo -iu postgres pgbackrest --stanza=cluster_1 stanza-create
   ```

2. Create a full backup

   ```
   $ sudo -iu postgres pgbackrest --stanza=cluster_1 --type=full backup
   ```

3. Create an incremental backup

   ```
   $ sudo -iu postgres pgbackrest --stanza=cluster_1 --type=incr backup
   ```

4. Check backup info

   ```
   $ sudo -iu postgres pgbackrest --stanza=cluster_1 info
   ```

5. Expire (remove) a backup. Be careful with removal, because removing a full backup also removes dependent incremental backups

   ```
   $ sudo -iu postgres pgbackrest --stanza=cluster_1 expire --set=20230617-021338F
   ```

Test PostgreSQL cluster

Last update: November 1, 2023
Created: November 1, 2023

5.1.5 Testing the Patroni PostgreSQL Cluster

This document covers the following scenarios to test the PostgreSQL cluster:

- replication,
- connectivity,
- failover, and
- manual switchover.

TESTING REPLICATION

1. Connect to the cluster and establish the `psql` session from a client machine that can connect to the HAProxy node. Use the HAProxy-demo node's public IP address:

```
$ psql -U postgres -h 134.209.111.138 -p 5000
```

2. Run the following commands to create a table and insert a few rows:

```
CREATE TABLE customer(name text,age integer);
INSERT INTO CUSTOMER VALUES('john',30);
INSERT INTO CUSTOMER VALUES('dawson',35);
```

3. To ensure that the replication is working, we can log in to each PostgreSQL node and run a simple SQL statement against the locally running instance:

```
$ sudo psql -U postgres -c "SELECT * FROM CUSTOMER;"
```

The results on each node should be the following:

```
  name  | age
--------+-----
 john   |  30
 dawson |  35
(2 rows)
```

TESTING FAILOVER

In a proper setup, client applications won't have issues connecting to the cluster, even if one or even two of the nodes go down. We will test the cluster for failover in the following scenarios:

Scenario 1. Intentionally stop the PostgreSQL on the primary node and verify access to PostgreSQL.

1. Run the following command on any node to check the current cluster status:

```
$ sudo patronictl -c /etc/patroni/patroni.yml list
```

Output:

```
+ Cluster: stampede1 (7011110722654005156) -----------+
| Member | Host  | Role    | State   | TL | Lag in MB |
+--------+-------+---------+---------+----+-----------+
| node1  | node1 | Leader  | running |  1 |           |
| node2  | node2 | Replica | running |  1 |         0 |
| node3  | node3 | Replica | running |  1 |          0 |
+--------+-------+---------+---------+----+-----------+
```

2. `node1` is the current leader. Stop Patroni in `node1` to see how it changes the cluster:

```
$ sudo systemctl stop patroni
```

3. Once the service stops in `node1`, check the logs in `node2` and `node3` using the following command:

```
$ sudo journalctl -u patroni.service -n 100 -f
```

Output

```
Sep 23 14:18:13 node03 patroni[10042]: 2021-09-23 14:18:13,905 INFO: no action. I am a secondary
(node3) and following a leader (node1)
Sep 23 14:18:20 node03 patroni[10042]: 2021-09-23 14:18:20,011 INFO: Got response from node2
http://node2:8008/patroni: {"state": "running", "postprimary_start_time": "2021-09-23
12:50:29.460027+00:00", "role": "replica", "server_version": 130003, "cluster_unlocked": true,
"xlog": {"received_location": 67219152, "replayed_location": 67219152, "replayed_timestamp":
"2021-09-23 13:19:50.329387+00:00", "paused": false}, "timeline": 1, "database_system_identifier":
"7011110722654005156", "patroni": {"version": "2.1.0", "scope": "stampede1"}}
Sep 23 14:18:20 node03 patroni[10042]: 2021-09-23 14:18:20,031 WARNING: Request failed to node1:
GET http://node1:8008/patroni (HTTPConnectionPool(host='node1', port=8008): Max retries exceeded
with url: /patroni (Caused by ProtocolError('Connection aborted.', ConnectionResetError(104,
'Connection reset by peer'))))
Sep 23 14:18:20 node03 patroni[10042]: 2021-09-23 14:18:20,038 INFO: Software Watchdog activated
with 25 second timeout, timing slack 15 seconds
Sep 23 14:18:20 node03 patroni[10042]: 2021-09-23 14:18:20,043 INFO: promoted self to leader by
acquiring session lock
Sep 23 14:18:20 node03 patroni[13641]: server promoting
Sep 23 14:18:20 node03 patroni[10042]: 2021-09-23 14:18:20,049 INFO: cleared rewind state after
becoming the leader
Sep 23 14:18:21 node03 patroni[10042]: 2021-09-23 14:18:21,101 INFO: no action. I am (node3) the
leader with the lock
Sep 23 14:18:21 node03 patroni[10042]: 2021-09-23 14:18:21,117 INFO: no action. I am (node3) the
leader with the lock
Sep 23 14:18:31 node03 patroni[10042]: 2021-09-23 14:18:31,114 INFO: no action. I am (node3) the
leader with the lock
...
```

The logs in `node3` show that the requests to `node1` are failing, the watchdog is coming into action, and `node3` is promoting itself as the leader:

4. Verify that you can still access the cluster through the HAProxy instance and read data:

```
$ psql -U postgres -h 10.104.0.6 -p 5000 -c "SELECT * FROM CUSTOMER;"

  name   | age
--------+-----
 john    |  30
 dawson |  35
(2 rows)
```

5. Restart the Patroni service in `node1`

```
$ sudo systemctl start patroni
```

6. Check the current cluster status:

```
$ sudo patronictl -c /etc/patroni/patroni.yml list
```

Output:

```
+ Cluster: stampede1 (7011110722654005156) -----------+
| Member | Host  | Role    | State   | TL | Lag in MB |
+--------+-------+---------+---------+----+-----------+
| node1  | node1 | Replica | running |  2 |         0 |
| node2  | node2 | Replica | running |  2 |         0 |
| node3  | node3 | Leader  | running |  2 |           |
+--------+-------+---------+---------+----+-----------+
```

As we see, `node3` remains the leader and the rest are replicas.

Scenario 2. Abrupt machine shutdown or power outage

To emulate the power outage, let’s kill the service in `node3` and see what happens in `node1` and `node2`.

1. Identify the process ID of Patroni and then kill it with a `-9` switch.

```
$ ps aux | grep -i patroni

postgres   10042  0.1  2.1 647132 43948 ?        Ssl  12:50   0:09 /usr/bin/python3 /usr/bin/
patroni /etc/patroni/patroni.yml

$ sudo kill -9 10042
```

2. Check the logs on `node2`:

```
$ sudo journalctl -u patroni.service -n 100 -f
```

Output

```
Sep 23 14:40:41 node02 patroni[10577]: 2021-09-23 14:40:41,656 INFO: no action. I am a secondary
(node2) and following a leader (node3)
…
Sep 23 14:41:01 node02 patroni[10577]: 2021-09-23 14:41:01,373 INFO: Got response from node1
http://node1:8008/patroni: {"state": "running", "postprimary_start_time": "2021-09-23
14:25:30.076762+00:00", "role": "replica", "server_version": 130003, "cluster_unlocked": true,
"xlog": {"received_location": 67221352, "replayed_location": 67221352, "replayed_timestamp": null,
"paused": false}, "timeline": 2, "database_system_identifier": "7011110722654005156", "patroni":
{"version": "2.1.0", "scope": "stampede1"}}
Sep 23 14:41:03 node02 patroni[10577]: 2021-09-23 14:41:03,364 WARNING: Request failed to node3:
GET http://node3:8008/patroni (HTTPConnectionPool(host='node3', port=8008): Max retries exceeded
with url: /patroni (Caused by ConnectTimeoutError(<urllib3.connection.HTTPConnection object at
0x7f57e06dffa0>, 'Connection to node3 timed out. (connect timeout=2)')))
Sep 23 14:41:03 node02 patroni[10577]: 2021-09-23 14:41:03,373 INFO: Software Watchdog activated
with 25 second timeout, timing slack 15 seconds
Sep 23 14:41:03 node02 patroni[10577]: 2021-09-23 14:41:03,385 INFO: promoted self to leader by
acquiring session lock
Sep 23 14:41:03 node02 patroni[15478]: server promoting
Sep 23 14:41:03 node02 patroni[10577]: 2021-09-23 14:41:03,397 INFO: cleared rewind state after
becoming the leader
Sep 23 14:41:04 node02 patroni[10577]: 2021-09-23 14:41:04,450 INFO: no action. I am (node2) the
leader with the lock
Sep 23 14:41:04 node02 patroni[10577]: 2021-09-23 14:41:04,475 INFO: no action. I am (node2) the
leader with the lock
…
…
```

`node2` realizes that the leader is dead, and promotes itself as the leader.

3. Try accessing the cluster using the HAProxy endpoint at any point in time between these operations. The cluster is still accepting connections.

MANUAL SWITCHOVER

Typically, a manual switchover is needed for planned downtime to perform maintenance activity on the leader node. Patroni provides the `switchover` command to manually switch over from the leader node.

Run the following command on `node2` (the current leader node):

```
$ sudo patronictl -c /etc/patroni/patroni.yml switchover
```

Patroni asks the name of the current primary node and then the node that should take over as the switched-over primary. You can also specify the time at which the switchover should happen. To trigger the process immediately, specify the value *now*:

```
primary [node2]: node2
Candidate ['node1', 'node3'] []: node1
When should the switchover take place (e.g. 2021-09-23T15:56 )  [now]: now
Current cluster topology
+ Cluster: stampede1 (7011110722654005156) -----------+
| Member | Host  | Role    | State   | TL | Lag in MB |
+--------+-------+---------+---------+----+-----------+
| node1  | node1 | Replica | running |  3 |         0 |
| node2  | node2 | Leader  | running |  3 |           |
| node3  | node3 | Replica | stopped |    |   unknown |
+--------+-------+---------+---------+----+-----------+
Are you sure you want to switchover cluster stampede1, demoting current primary node2? [y/
N]: y
2021-09-23 14:56:40.54009 Successfully switched over to "node1"
+ Cluster: stampede1 (7011110722654005156) -----------+
| Member | Host  | Role    | State   | TL | Lag in MB |
+--------+-------+---------+---------+----+-----------+
| node1  | node1 | Leader  | running |  3 |           |
| node2  | node2 | Replica | stopped |    |   unknown |
| node3  | node3 | Replica | stopped |    |   unknown |
+--------+-------+---------+---------+----+-----------+
```

Restart the Patroni service in `node2` (after the "planned maintenance"). The node rejoins the cluster as a secondary.

Last update: December 5, 2022

Created: December 15, 2021

5.2 Backup and disaster recovery

5.2.1 Backup and disaster recovery in Percona Distribution for PostgreSQL

Summary

- Overview
- Architecture
- Deployment
- Testing

Overview

A Disaster Recovery (DR) solution ensures that a system can be quickly restored to a normal operational state if something unexpected happens. When operating a database, you would back up the data as frequently as possible and have a mechanism to restore that data when needed. Disaster Recovery is often mistaken for high availability (HA), but they are two different concepts altogether:

- High availability ensures guaranteed service levels at all times. This solution involves configuring one or more standby systems to an active database, and the ability to switch seamlessly to that standby when the primary database becomes unavailable, for example, during a power outage or a server crash. To learn more about high-availability solutions with Percona Distribution for PostgreSQL, refer to High Availability in PostgreSQL with Patroni.
- Disaster Recovery protects the database instance against accidental or malicious data loss or data corruption. Disaster recovery can be achieved by using either the options provided by PostgreSQL, or external extensions.

PostgreSQL disaster recovery options

PostgreSQL offers multiple options for setting up database disaster recovery.

- **pg_dump or the pg_dumpall utilities**

This is the basic backup approach. These tools can generate the backup of one or more PostgreSQL databases (either just the structure, or both the structure and data), then restore them through the pg_restore command.

Advantages	Disadvantages
Easy to use	1. Backup of only one database at a time. 2. No incremental backups. 3. No point-in-time recovery since the backup is a snapshot in time. 4. Performance degradation when the database size is large.

- **File-based backup and restore**

This method involves backing up the PostgreSQL data directory to a different location, and restoring it when needed.

Advantages	Disadvantages
Consistent snapshot of the data directory or the whole data disk volume	1. Requires stopping PostgreSQL in order to copy the files. This is not practical for most production setups. 2. No backup of individual databases or tables.

- **PostgreSQL pg_basebackup**

This backup tool is provided by PostgreSQL. It is used to back up data when the database instance is running. `pgasebackup` makes a binary copy of the database cluster files, while making sure the system is put in and out of backup mode automatically.

Advantages	Disadvantages
1. Supports backups when the database is running. 2. Supports point-in-time recovery	1. No incremental backups. 2. No backup of individual databases or tables.

To achieve a production grade PostgreSQL disaster recovery solution, you need something that can take full or incremental database backups from a running instance, and restore from those backups at any point in time. Percona Distribution for PostgreSQL is supplied with pgBackRest: a reliable, open-source backup and recovery solution for PostgreSQL.

This document focuses on the Disaster recovery solution in Percona Distribution for PostgreSQL. The Deploying backup and disaster recovery solution in Percona Distribution for PostgreSQL tutorial provides guidelines of how to set up and test this solution.

PGBACKREST

pgBackRest is an easy-to-use, open-source solution that can reliably back up even the largest of PostgreSQL databases. `pgBackRest` supports the following backup types:

- full backup - a complete copy of your entire data set.
- differential backup - includes all data that has changed since the last full backup. While this means the backup time is slightly higher, it enables a faster restore.
- incremental backup - only backs up the files that have changed since the last full or differential backup, resulting in a quick backup time. To restore to a point in time, however, you will need to restore each incremental backup in the order they were taken.

When it comes to restoring, `pgBackRest` can do a full or a delta restore. A *full* restore needs an empty PostgreSQL target directory. A *delta* restore is intelligent enough to recognize already-existing files in the PostgreSQL data directory, and update only the ones the backup contains.

`pgBackRest` supports remote repository hosting and can even use cloud-based services like AWS S3, Google Cloud Services Cloud Storage, Azure Blob Storage for saving backup files. It supports parallel backup through multi-core processing and compression. By default, backup integrity is verified through checksums, and saved files can be encrypted for enhanced security.

`pgBackRest` can restore a database to a specific point in time in the past. This is the case where a database is not inaccessible but perhaps contains corrupted data. Using the point-in-time recovery, a database administrator can restore the database to the last known good state.

Finally, `pgBackRest` also supports restoring PostgreSQL databases to a different PostgreSQL instance or a separate data directory.

Setup overview

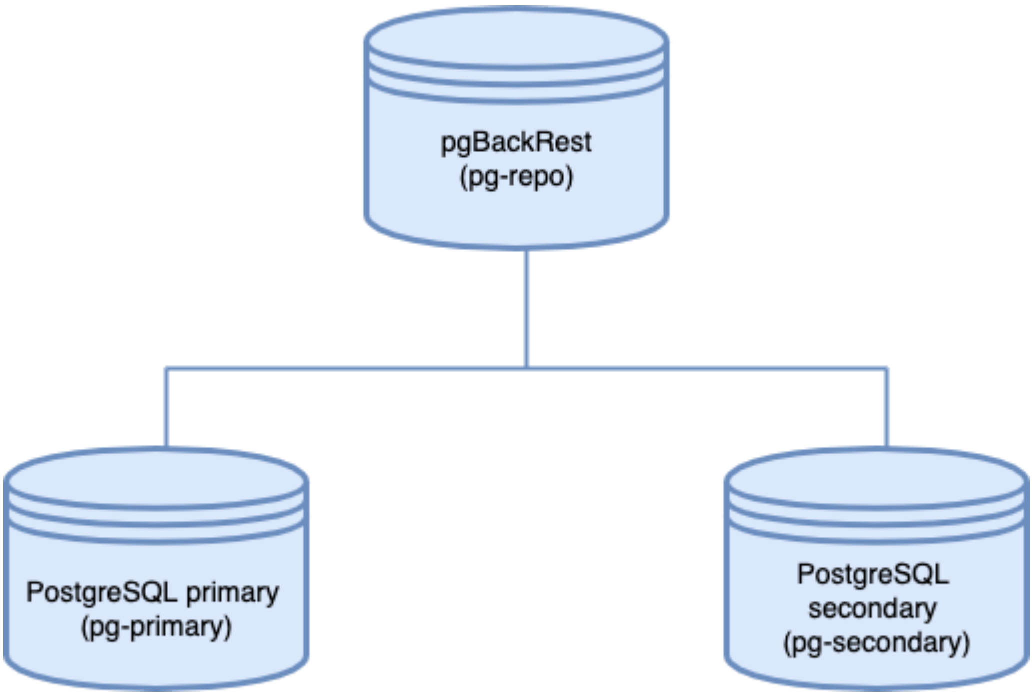
This section describes the architecture of the backup and disaster recovery solution. For the configuration steps, refer to the Deploying backup and disaster recovery solution in Percona Distribution for PostgreSQL.

SYSTEM ARCHITECTURE

As the configuration example, we will use a three server architecture where `pgBackRest` resides on a dedicated remote host. The servers communicate with each other via passwordless SSH.

> **Important**
>
> Passwordless SSH may not be an ideal solution for your environment. In this case, consider using other methods, for example, TLS with client certificates.

The following diagram illustrates the architecture layout:



Components:

The architecture consists of three server instances:

- `pg-primary` hosts the primary PostgreSQL server. Note that "primary" here means the main database instance and does not refer to the primary node of a PostgreSQL replication cluster or a HA setup.
- `pg-repo` is the remote backup repository and hosts `pgBackRest`. It's important to host the backup repository on a physically separate instance, to be accessed when the target goes down.
- `pg-secondary` is the *secondary* PostgreSQL node. Don't confuse it with a hot standby. "Secondary" in this context means a PostgreSQL instance that's idle. We will restore the database backup to this instance when the primary PostgreSQL instance goes down.

Note

For simplicity, we use a single-node PostgreSQL instance as the primary database server. In a production scenario, you will use some form of high-availability solution to protect the primary instance. When you are using a high-availability setup, we recommend configuring `pgBackRest` to back up the hot standby server so the primary node is not unnecessarily loaded.

DEPLOYMENT

Refer to the Deploying backup and disaster recovery solution in Percona Distribution for PostgreSQL tutorial.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: October 24, 2022

Created: January 21, 2022

5.2.2 Deploying backup and disaster recovery solution in Percona Distribution for PostgreSQL

This document provides instructions of how to set up and test the backup and disaster recovery solution in Percona Distribution for PostgreSQL with `pgBackRest`. For technical overview and architecture description of this solution, refer to Backup and disaster recovery in Percona Distribution for PostgreSQL.

Deployment

As the example configuration, we will use the nodes with the following IP addresses:

Node name	Internal IP address
pg-primary	10.104.0.3
pg-repo	10.104.0.5
pg-secondary	10.104.0.4

SET UP HOSTNAMES

In our architecture, the `pgBackRest` repository is located on a remote host. To allow communication among the nodes, passwordless SSH is required. To achieve this, properly setting up hostnames in the `/etc/hosts` files is very important.

1. Define the hostname for every server in the `/etc/hostname` file. The following are the examples of how the `/etc/hostname` file in three nodes looks like:

```
cat /etc/hostname
pg-primary
```

```
cat /etc/hostname
pg-repo
```

```
cat /etc/hostname
pg-secondary
```

2. For the nodes to communicate seamlessly across the network, resolve their hostnames to their IP addresses in the `/etc/hosts` file. (Alternatively, you can make appropriate entries in your internal DNS servers)

The `/etc/hosts` file for the `pg-primary node` looks like this:

```
```
127.0.1.1 pg-primary pg-primary
127.0.0.1 localhost
10.104.0.5 pg-repo
```
```

The `/etc/hosts` file in the `pg-repo` node looks like this:

```
```
127.0.1.1 pg-repo pg-repo
127.0.0.1 localhost
10.104.0.3 pg-primary
10.104.0.4 pg-secondary
```
```

The `/etc/hosts` file in the `pg-secondary` node is shown below:

~~~
```
127.0.1.1 pg-secondary pg-secondary
127.0.0.1 localhost
10.104.0.3 pg-primary
10.104.0.5 pg-repo
```
~~~

SET UP PASSWORDLESS SSH

Before setting up passwordless SSH, ensure that the *postgres* user in all three instances has a password.

1. To set or change the password, run the following command **as a root user**:

```
$ passwd postgres
```

2. Type the new password and confirm it.
3. After setting up the password, edit the `/etc/ssh/sshd_config` file and ensure the `PasswordAuthentication` variable is set as `yes`.

```
PasswordAuthentication yes
```

4. In the `pg-repo` node, restart the `sshd` service. Without the restart, the SSH server will not allow you to connect to it using a password while adding the keys.

```
$ sudo service sshd restart
```

5. In the `pg-primary` node, generate an SSH key pair and add the public key to the `pg-repo` node.

Important

Run the commands as the **postgres** user.

- Generate SSH keys:

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
...
```

- Copy the public key to the `pg-repo` node:

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub postgres@pg-repo
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is
to install the new keys
postgres@pg-repo's password:

Number of key(s) added: 1


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

6. To verify everything has worked as expected, run the following command from the `pg-primary` node.

```
$ ssh postgres@pg-repo
```

You should be able to connect to the `pg-repo` terminal without a password.

7. Repeat the SSH connection from `pg-repo` to `pg-primary` to ensure that passwordless SSH is working.
8. Set up bidirectional passwordless SSH between `pg-repo` and `pg-secondary` using the same method. This will allow `pg-repo` to recover the backups to `pg-secondary`.

INSTALL PERCONA DISTRIBUTION FOR POSTGRESQL

Install Percona Distribution for PostgreSQL in the primary and the secondary nodes from Percona repository.

1. Install `percona-release`.
2. Enable the repository:

```
$ sudo percona-release setup ppg14
```

3. Install Percona Distribution for PostgreSQL packages

On Debian and Ubuntu **On RedHat Enterprise Linux and derivatives**

```
$ sudo apt install percona-postgresql-14 -y

$ sudo yum install percona-postgresql14-server
```

CONFIGURE POSTGRESQL ON THE PRIMARY NODE FOR CONTINUOUS BACKUP

At this step, configure the PostgreSQL instance on the `pg-primary` node for continuous archiving of the WAL files.

Note

On Debian and Ubuntu, the path to the configuration file is `/etc/postgresql/14/main/postgresql.conf`.

On RHEL and CentOS, the path to the configuration file is `/var/lib/pgsql/14/data/`.

1. Edit the `postgresql.conf` configuration file to include the following changes:

```
archive_command = 'pgbackrest --stanza=prod_backup archive-push %p'
archive_mode = on
listen_addresses = '*'
log_line_prefix = ''
max_wal_senders = 3
wal_level = replica
```

2. Once the changes are saved, restart PostgreSQL.

```
$ sudo systemctl restart postgresql
```

INSTALL PGBACKREST

Install `pgBackRest` in all three instances from Percona repository. Use the following command:

On Debian / Ubuntu **On RHEL / derivatives**

```
$ sudo apt-get install percona-pgbackrest

$ sudo yum install percona-pgbackrest
```

CREATE THE `PGBACKREST` CONFIGURATION FILE

Run the following commands on all three nodes to set up the required configuration file for `pgBackRest`.

1. Configure a location and permissions for the `pgBackRest` log rotation:

```
$ sudo mkdir -p -m 770 /var/log/pgbackrest
$ sudo chown postgres:postgres /var/log/pgbackrest
```

2. Configure the location and permissions for the `pgBackRest` configuration file:

```
$ sudo mkdir -p /etc/pgbackrest
$ sudo mkdir -p /etc/pgbackrest/conf.d
$ sudo touch /etc/pgbackrest/pgbackrest.conf
$ sudo chmod 640 /etc/pgbackrest/pgbackrest.conf
$ sudo chown postgres:postgres /etc/pgbackrest/pgbackrest.conf
$ sudo mkdir -p /home/pgbackrest
$ sudo chmod postgres:postgres /home/pgbackrest
```

UPDATE `PGBACKREST` CONFIGURATION FILE IN THE PRIMARY NODE

Configure `pgBackRest` on the `pg-primary` node by setting up a stanza. A stanza is a set of configuration parameters that tells `pgBackRest` where to backup its files. Edit the `/etc/pgbackrest/pgbackrest.conf` file in the `pg-primary` node to include the following lines:

```
[global]
repo1-host=pg-repo
repo1-host-user=postgres
process-max=2
log-level-console=info
log-level-file=debug

[prod_backup]
pg1-path=/var/lib/postgresql/14/main
```

You can see the `pg1-path` attribute for the `prod_backup` stanza has been set to the PostgreSQL data folder.

UPDATE `PGBACKREST` CONFIGURATION FILE IN THE REMOTE BACKUP REPOSITORY NODE

Add a stanza for the `pgBackRest` in the `pg-repo` node. Edit the `/etc/pgbackrest/pgbackrest.conf` configuration file to include the following lines:

```
[global]
repo1-path=/home/pgbackrest/pg_backup
repo1-retention-full=2
process-max=2
log-level-console=info
log-level-file=debug
start-fast=y
stop-auto=y

[prod_backup]
pg1-path=/var/lib/postgresql/14/main
pg1-host=pg-primary
pg1-host-user=postgres
pg1-port = 5432
```

INITIALIZE `PGBACKREST` STANZA IN THE REMOTE BACKUP REPOSITORY NODE

After the configuration files are set up, it's now time to initialize the `pgBackRest` stanza. Run the following command in the remote backup repository node (`pg-repo`).

```
$ sudo -u postgres pgbackrest --stanza=prod_backup stanza-create
2021-11-07 11:08:18.157 P00   INFO: stanza-create command begin 2.36: --exec-
id=155883-2277a3e7 --log-level-console=info --log-level-file=off --pg1-host=pg-primary --
pg1-host-user=postgres --pg1-path=/var/lib/postgresql/14/main --pg1-port=5432 --repo1-path=/
home/pgbackrest/pg_backup --stanza=prod_backup
2021-11-07 11:08:19.453 P00   INFO: stanza-create for stanza 'prod_backup' on repo1
2021-11-07 11:08:19.566 P00   INFO: stanza-create command end: completed successfully
(1412ms)
```

Once the stanza is created successfully, you can try out the different use cases for disaster recovery.

Testing Backup and Restore with `pgBackRest`

This section covers a few use cases where `pgBackRest` can back up and restore databases either in the same instance or a different node.

USE CASE 1: CREATE A BACKUP WITH `PGBACKREST`

1. To start our testing, let's create a table in the `postgres` database in the `pg-primary` node and add some data.

```
CREATE TABLE CUSTOMER (id integer, name text);
INSERT INTO CUSTOMER VALUES (1,'john');
INSERT INTO CUSTOMER VALUES (2,'martha');
INSERT INTO CUSTOMER VALUES (3,'mary');
```

2. Take a full backup of the database instance. Run the following commands from the `pg-repo` node:

```
$ pgbackrest -u postgres  --stanza=prod_backup backup --type=full
```

If you want an incremental backup, you can omit the `type` attribute. By default, `pgBackRest` always takes an incremental backup except the first backup of the cluster which is always a full backup.

If you need a differential backup, use *diff* for the `type` field:

```
$ pgbackrest -u postgres --stanza=prod_backup backup --type=diff
```

USE CASE 2: RESTORE A POSTGRESQL INSTANCE FROM A FULL BACKUP

For testing purposes, let's "damage" the PostgreSQL instance.

1. Run the following command in the `pg-primary` node to delete the main data directory.

```
$ rm -rf /var/lib/postgresql/14/main/*
```

2. To restore the backup, run the following commands.

- Stop the `postgresql` instance

```
$ sudo systemctl stop postgresql
```

- Restore the backup:

```
$ pgbackrest -u postgres --stanza=prod_backup restore
```

- Start the `postgresql` instance

```
$ sudo systemctl start postgresql
```

3. After the command executes successfully, you can access PostgreSQL from the `psql` command line tool and check if the table and data rows have been restored.

USE CASE 3: POINT-IN-TIME RECOVERY

If your target PostgreSQL instance has an already existing data directory, the full restore option will fail. You will get an error message stating there are existing data files. In this case, you can use the `--delta` option to restore only the corrupted files.

For example, let's say one of your developers mistakenly deleted a few rows from a table. You can use `pgBackRest` to revert your database to a previous point in time to recover the lost rows.

To test this use case, do the following:

1. Take a timestamp when the database is stable and error-free. Run the following command from the `psql` prompt.

   ```
   SELECT CURRENT_TIMESTAMP;
          current_timestamp
   -------------------------------
    2021-11-07 11:55:47.952405+00
   (1 row)
   ```

   Note down the above timestamp since we will use this time in the restore command. Note that in a real life scenario, finding the correct point in time when the database was error-free may require extensive investigation. It is also important to note that all changes after the selected point will be lost after the roll back.
2. Delete one of the customer records added before.

   ```
   DELETE FROM CUSTOMER WHERE ID=3;
   ```

3. To recover the data, run a command with the noted timestamp as an argument. Run the commands below to recover the database up to that time.

- Stop the `postgresql` instance

  ```
  $ sudo systemctl stop postgresql
  ```

- Restore the backup

  ```
  $ pgbackrest -u postgres --stanza=prod_backup --delta \
  --type=time "--target= 2021-11-07 11:55:47.952405+00" \
  --target-action=promote restore
  ```

- Start the `postgresql` instance

  ```
  $ sudo systemctl start postgresql
  ```

4. Check the database table to see if the record has been restored.

   ```
   SELECT * FROM customer;
    id |   name
   ----+--------
     1 | john
     2 | martha
     3 | mary
   (3 rows)
   ```

USE CASE 4: RESTORING TO A SEPARATE POSTGRESQL INSTANCE

Sometimes a PostgreSQL server may encounter hardware issues and become completely inaccessible. In such cases, we will need to recover the database to a separate instance where `pgBackRest` is not initially configured. To restore the instance to a separate host, you have to first install both PostgreSQL and `pgBackRest` in this host.

In our test setup, we already have PostgreSQL and `pgBackRest` installed in the third node, `pg-secondary`. Change the `pgBackRest` configuration file in the `pg-secondary` node as shown below.

```
[global]
repo1-host=pg-repo
repo1-host-user=postgres
process-max=2
log-level-console=info
log-level-file=debug

[prod_backup]
pg1-path=/var/lib/postgresql/14/main
```

There should be bidirectional passwordless SSH communication between `pg-repo` and `pg-secondary`. Refer to the Set up passwordless SSH section for the steps, if you haven't configured it.

Stop the PostgreSQL instance

```
$ sudo systemctl stop postgresql
```

Restore the database backup from `pg-repo` to `pg-secondary`.

```
$ pgbackrest -u postgres --stanza=prod_backup --delta restore

2021-11-07 13:34:08.897 P00   INFO: restore command begin 2.36: --delta --exec-id=109728-
d81c7b0b --log-level-console=info --log-level-file=debug --pg1-path=/var/lib/postgresql/14/
main --process-max=2 --repo1-host=pg-repo --repo1-host-user=postgres --stanza=prod_backup
2021-11-07 13:34:09.784 P00   INFO: repo1: restore backup set
20211107-111534F_20211107-131807I, recovery will start at 2021-11-07 13:18:07
2021-11-07 13:34:09.786 P00   INFO: remove invalid files/links/paths from '/var/lib/
postgresql/14/main'
2021-11-07 13:34:11.803 P00   INFO: write updated /var/lib/postgresql/14/main/
postgresql.auto.conf
2021-11-07 13:34:11.819 P00   INFO: restore global/pg_control (performed last to ensure
aborted restores cannot be started)
2021-11-07 13:34:11.819 P00   INFO: restore size = 23.2MB, file total = 937
2021-11-07 13:34:11.820 P00   INFO: restore command end: completed successfully (2924ms)
```

After the restore completes successfully, restart PostgreSQL:

```
$ sudo systemctl start postgresql
```

Check the database contents from the local `psql` shell.

```
SELECT * FROM customer;
 id |  name
----+--------
  1 | john
  2 | martha
  3 | mary
(3 rows)
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: December 5, 2022

Created: January 21, 2022

5.3 Spatial data handling

5.3.1 Spatial data manipulation

Version added: 15.3

Organizations dealing with spatial data need to store it somewhere and manipulate it. PostGIS is the open source extension for PostgreSQL that allows doing just that. It adds support for storing the spatial data types such as:

- Geographical data like points, lines, polygons, GPS coordinates that can be mapped on a sphere.
- Geometrical data. This is also points, lines and polygons but they apply to a 2D surface.

To operate with spatial data inside SQL queries, PostGIS supports spatial functions like distance, area, union, intersection. It uses the spatial indexes like R-Tree and Quadtree for efficient processing of database operations. Read more about supported spatial functions and indexes in PostGIS documentation.

By deploying PostGIS with Percona Distribution for PostgreSQL, you receive the open-source spatial database that you can use in various areas without vendor lock-in.

When to use PostGIS

You can use PostGIS in the following cases:

- To store and manage spatial data, create and store spatial shapes, calculate areas and distances
- To build the software that visualizes spatial data on a map,
- To work with raster data, such as satellite imagery or digital elevation models.
- To integrate spatial and non-spatial data such as demographic or economic data in a database

When not to use PostGIS

Despite its power and flexibility, PostGIS may not suit your needs if:

- You need to store only a couple of map locations. Consider using the built-in geometric functions and operations of PostgreSQL
- You need real-time data analysis. While PostGIS can handle real-time spatial data, it may not be the best option for real-time data analysis on large volumes of data.
- You need complex 3D analysis or visualization.
- You need to acquire spatial data. Use other tools for this purpose and import spatial data into PostGIS to manipulate it.

Next steps:

Deployment

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: June 30, 2023

Created: June 28, 2023

5.3.2 Deploy spatial data with PostgreSQL

The following document provides guidelines how to install PostGIS and how to run the basic queries.

Considerations

1. We assume that you have the basic knowledge of spatial data, GIS (Geographical Information System) and of shapefiles.
2. For uploading the spatial data and querying the database, we use the same data set as is used in PostGIS tutorial.

Install PostGIS

On Debian and Ubuntu **On RHEL and derivatives**

1. Enable Percona repository

   As other components of Percona Distribution for PostgreSQL, PostGIS is available from Percona repositories. Use the `percona-release` repository management tool to enable the repository.

   ```
   $ sudo percona-release setup ppg15
   ```

2. Install PostGIS packages

   ```
   $ sudo apt install percona-postgis
   ```

3. The command in the previous step installs the set of PostGIS extensions. To check what extensions are available, run the following query from the `psql` terminal:

   ```
   SELECT name, default_version,installed_version
   FROM pg_available_extensions WHERE name LIKE 'postgis%' or name LIKE address%';
   ```

Note

To enable the `postgis_sfcgal-3` extension on Ubuntu 18.04, you need to manually install the required dependency:

```
$ sudo apt-get install libsfcgal1
```

1. Check the Platform specific notes and enable required repositories and modules for the dependencies relevant to your operating system.
2. Enable Percona repository

   As other components of Percona Distribution for PostgreSQL, PostGIS is available from Percona repositories. Use the `percona-release` repository management tool to enable the repository.

   ```
   $ sudo percona-release setup ppg15
   ```

3. Install the extension

   ```
   $ sudo yum install percona-postgis33_15 percona-postgis33_15-client
   ```

   This installs the set of PostGIS extensions. To check what extensions are available, run the following query from the `psql` terminal:

   ```
   SELECT name, default_version,installed_version
   FROM pg_available_extensions WHERE name LIKE 'postgis%' or name LIKE 'address%';
   ```

Enable PostGIS extension

1. Create a database and a schema for this database to store your data. A schema is a container that logically segments objects (tables, functions, views, and so on) for better management. Run the following commands from the `psql` terminal:

   ```
   CREATE database nyc;
   \c nyc;
   CREATE SCHEMA gis;
   ```

2. To make PostGIS functions and operations work, you need to enable the `postgis` extension. Make sure you are connected to the database you created earlier and run the following command:

   ```
   CREATE EXTENSION postgis;
   ```

3. Check that the extension is enabled:

   ```
   SELECT postgis_full_version();
   ```

   The output should be similar to the following:

   ```
   postgis_full_version
   -----------------------------------------------------------------------------------------------
    POSTGIS="3.3.3" [EXTENSION] PGSQL="140" GEOS="3.10.2-CAPI-1.16.0" PROJ="8.2.1"
   LIBXML="2.9.13" LIBJSON="0.15" LIBPROTOBUF="1.3.3" WAGYU="0.5.0 (Internal)"
   ```

Upload spatial data to PostgreSQL

PostGIS provides the `shp2pgsql` command line utility that converts the binary data from shapefiles into the series of SQL commands and loads them into the database.

1. For testing purposes, download the sample data set:

```
$ curl -LO https://s3.amazonaws.com/s3.cleverelephant.ca/postgis-workshop-2020.zip
```

2. Unzip the archive and from the folder where the `.shp` files are located, execute the following command and replace the `dbname` value with the name of your database:

```
shp2pgsql \
  -D \
  -I \
  -s 26918 \
  nyc_streets.shp \
  nyc_streets \
  | psql -U postgres dbname=nyc
```

The command does the following:

- `-D` flag instructs the command to generate the dump format
- `-I` flag instructs to create the spatial index on the table upon the data load
- `-s` indicates the spatial reference identifier of the data. The data we load is in the Projected coordinate system for North America and has the value 26918.
- `nyc_streets.shp` is the source shapefile
- `nyc_streets` is the table name to create in the database
- `dbname=nyc` is the database name

3. Check the uploaded data

```
\d nyc_streets;
                                         Table "public.nyc_streets"
 Column |               Type                | Collation | Nullable |                 Default
--------+---------------------------------+-----------+----------
+-------------------------------------------
 gid    | integer                         |           | not null |
nextval('nyc_streets_gid_seq'::regclass)
 id     | double precision                |           |          |
 name   | character varying(200)          |           |          |
 oneway | character varying(10)           |           |          |
 type   | character varying(50)           |           |          |
 geom   | geometry(MultiLineString,26918) |           |          |
Indexes:
    "nyc_streets_pkey" PRIMARY KEY, btree (gid)
    "nyc_streets_geom_idx" gist (geom)
```

1. Repeat the command to upload other shapefiles in the data set: `nyc_census_blocks`, `nyc_neighborhoods`, `nyc_subway_stations`

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 30, 2023

Created: June 28, 2023

5.3.3 Query spatial data

After you installed and set up PostGIS, let's find answers to the following questions by querying the database:

What is the population of the New York City?

```
SELECT Sum(popn_total) AS population
  FROM nyc_census_blocks;
```

Output:

```
population
------------
    8175032
(1 row)
```

What is the area of Central Park?

To get the answer we will use the `ST_Area` function that returns the areas of polygons.

```
SELECT ST_Area(geom) / 1000000
  FROM nyc_neighborhoods
  WHERE name = 'Central Park';
```

Output:

```
      st_area
--------------------
 3.5198365965413293
(1 row)
```

By default, the output is given in square meters. To get the value in square kilometers, divide it by 1 000 000.

How long is Columbus Circle?

```
SELECT ST_Length(geom)
  FROM nyc_streets
  WHERE name = 'Columbus Cir';
```

Output:

```
     st_length
-------------------
 308.3419936909855
(1 row)
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: July 3, 2023

Created: June 28, 2023

5.3.4 Spatial database upgrade

When using PostgreSQL and PostGIS for some time, you may eventually come to the decision to upgrade your spatial database. There can be different reasons for that: to receive improvements and/or bug fixes that come with a minor version of the database/extension, reaching the end of life of the currently used software and others.

The spatial database upgrade consists of two steps:

- upgrade of PostgreSQL, and
- upgrade of the PostGIS extension.

Important

Before the upgrade, backup your data.

Upgrade PostGIS

Each version of PostGIS is compatible with several versions of PostgreSQL and vise versa. The best practice is to first upgrade the PostGIS extension on the source cluster to match the compatible version on the target cluster and then upgrade PostgreSQL. Please see the PostGIS Support matrix for version compatibility.

PostGIS is enabled on the database level. This means that the upgrade is also done on the database level.

PostGIS 3 and above **PostGIS 2.5**

Connect to the database where it is enabled and run the `PostGIS_Extensions_Upgrade()` function:

```
SELECT postgis_extensions_upgrade();
```

Repeat these steps to upgrade PostGIS on every database where it is enabled.

Connect to the database with the enabled extension and run the following commands:

```
ALTER EXTENSION postgis UPDATE;
SELECT postgis_extensions_upgrade();
```

Starting with version 3, vector and raster functionalities have been separated in two individual extensions. Thus, to upgrade those, you need to run the `postgis_extensions_upgrade();` twice.

```
SELECT postgis_extensions_upgrade();
```

TIP: If you don't need the raster functionality, you can drop the `postgis_raster` extension after the upgrade.

Repeat these steps to upgrade PostGIS on every database where it is enabled.

Upgrade PostgreSQL

Upgrade PostgreSQL either to the latest minor or to the major version.

If you are using long deprecated views and functions and / or need the expertise in upgrading your spatial database, contact Percona Managed Services for an individual upgrade scenario development.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: June 28, 2023

Created: June 28, 2023

5.4 LDAP Authentication

When a client application or a user that runs the client application connects to the database, it must identify themselves. The process of validating the client's identity and determining whether this client is permitted to access the database it has requested is called **authentication**.

Percona Distribution for PortgreSQL supports several authentication methods, including the LDAP authentication. The use of LDAP is to provide a central place for authentication - meaning the LDAP server stores usernames and passwords and their resource permissions.

The LDAP authentication in Percona Distribution for PortgreSQL is implemented the same way as in upstream PostgreSQL.

Last update: October 24, 2022

Created: May 30, 2022

6. Telemetry on Percona Distribution for PostgreSQL

Percona telemetry fills in the gaps in our understanding of how you use Percona Distribution for PostgreSQL to improve our products. Participation in this anonymous program is optional. You can opt-out if you prefer to not share this information.

6.1 What information is collected

Currently, telemetry is added only to the Percona packages and Docker images. It collects only information about the installation environment. Future releases may add additional telemetry metrics.

Be assured that access to this raw data is rigorously controlled. Percona does not collect personal data. All data is anonymous and cannot be traced to a specific user. To learn more about our privacy practices, read the Percona Privacy statement.

The following is an example of the collected data:

```
[{"id" : "c416c3ee-48cd-471c-9733-37c2886f8231",
"product_family" : "PRODUCT_FAMILY_POSTGRESQL",
"instanceId" : "6aef422e-56a7-4530-af9d-94cc02198343",
"createTime" : "2023-11-01T10:46:23Z",
"metrics":
[{"key" : "deployment","value" : "PACKAGE"},
{"key" : "pillar_version","value" : "15.5"},
{"key" : "OS","value" : "Oracle Linux Server 8.8"},
{"key" : "hardware_arch","value" : "x86_64 x86_64"}]}]
```

6.2 Disable telemetry

Starting with Percona Distribution for PostgreSQL 16.1, telemetry is enabled by default. If you decide not to send usage data to Percona, you can set the `PERCONA_TELEMETRY_DISABLE=1` environment variable for either the root user or in the operating system prior to the installation process.

Debian-derived distribution **Red Hat-derived distribution** **DOCKER**

Add the environment variable before the install process.

```
$ sudo PERCONA_TELEMETRY_DISABLE=1 apt install percona-postgresql-15
```

Add the environment variable before the install process.

```
$ sudo PERCONA_TELEMETRY_DISABLE=1 yum install percona-postgresql15-server
```

Add the environment variable when running a command in a new container.

```
$ docker run --name container-name -e POSTGRES_PASSWORD=secret -e
PERCONA_TELEMETRY_DISABLE=1 -d percona/percona-distribution-postgresql:tag
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: November 30, 2023

Created: November 30, 2023

7. Uninstalling Percona Distribution for PostgreSQL

To uninstall Percona Distribution for PostgreSQL, remove all the installed packages and data / configuration files.

NOTE: Should you need the data files later, back up your data before uninstalling Percona Distribution for PostgreSQL.

On Debian and Ubuntu using `apt` **On Red Hat Enterprise Linux and derivatives using `yum`**

To uninstall Percona Distribution for PostgreSQL on platforms that use **apt** package manager such as Debian or Ubuntu, complete the following steps.

Run all commands as root or via **sudo**.

1. Stop the Percona Distribution for PostgreSQL service.

```
$ sudo systemctl stop postgresql.service
```

2. Remove the **percona-postgresql** packages.

```
$ sudo apt remove percona-postgresql-15* percona-patroni percona-pgbackrest  percona-
pgbadger percona-pgbouncer
```

3. Remove configuration and data files.

```
$ rm -rf /etc/postgresql/15/main
```

To uninstall Percona Distribution for PostgreSQL on platforms that use **yum** package manager such as Red Hat Enterprise Linux or CentOS, complete the following steps.

Run all commands as root or via **sudo**.

1. Stop the Percona Distribution for PostgreSQL service.

```
$ sudo systemctl stop postgresql-15
```

2. Remove the **percona-postgresql** packages

```
$ sudo yum remove percona-postgresql15* percona-pgbadger
```

3. Remove configuration and data files

```
$ rm -rf /var/lib/pgsql/15/data
```

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: March 20, 2023
Created: June 4, 2021

8. Copyright and licensing information

Percona Distribution for PostgreSQL is licensed under the PostgreSQL license and licenses of all components included in the Distribution.

8.1 Documentation licensing

Percona Distribution for PostgreSQL documentation is (C)2009-2023 Percona LLC and/or its affiliates and is distributed under the Creative Commons Attribution 4.0 International License.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: June 28, 2023

Created: June 4, 2021

9. Trademark Policy

This Trademark Policy is to ensure that users of Percona-branded products or services know that what they receive has really been developed, approved, tested and maintained by Percona. Trademarks help to prevent confusion in the marketplace, by distinguishing one company's or person's products and services from another's.

Percona owns a number of marks, including but not limited to Percona, XtraDB, Percona XtraDB, XtraBackup, Percona XtraBackup, Percona Server, and Percona Live, plus the distinctive visual icons and logos associated with these marks. Both the unregistered and registered marks of Percona are protected.

Use of any Percona trademark in the name, URL, or other identifying characteristic of any product, service, website, or other use is not permitted without Percona's written permission with the following three limited exceptions.

First, you may use the appropriate Percona mark when making a nominative fair use reference to a bona fide Percona product.

Second, when Percona has released a product under a version of the GNU General Public License ("GPL"), you may use the appropriate Percona mark when distributing a verbatim copy of that product in accordance with the terms and conditions of the GPL.

Third, you may use the appropriate Percona mark to refer to a distribution of GPL-released Percona software that has been modified with minor changes for the sole purpose of allowing the software to operate on an operating system or hardware platform for which Percona has not yet released the software, provided that those third party changes do not affect the behavior, functionality, features, design or performance of the software. Users who acquire this Percona-branded software receive substantially exact implementations of the Percona software.

Percona reserves the right to revoke this authorization at any time in its sole discretion. For example, if Percona believes that your modification is beyond the scope of the limited license granted in this Policy or that your use of the Percona mark is detrimental to Percona, Percona will revoke this authorization. Upon revocation, you must immediately cease using the applicable Percona mark. If you do not immediately cease using the Percona mark upon revocation, Percona may take action to protect its rights and interests in the Percona mark. Percona does not grant any license to use any Percona mark for any other modified versions of Percona software; such use will require our prior written permission.

Neither trademark law nor any of the exceptions set forth in this Trademark Policy permit you to truncate, modify or otherwise use any Percona mark as part of your own brand. For example, if XYZ creates a modified version of the Percona Server, XYZ may not brand that modification as "XYZ Percona Server" or "Percona XYZ Server", even if that modification otherwise complies with the third exception noted above.

In all cases, you must comply with applicable law, the underlying license, and this Trademark Policy, as amended from time to time. For instance, any mention of Percona trademarks should include the full trademarked name, with proper spelling and capitalization, along with attribution of ownership to Percona Inc. For example, the full proper name for XtraBackup is Percona XtraBackup. However, it is acceptable to omit the word "Percona" for brevity on the second and subsequent uses, where such omission does not cause confusion.

In the event of doubt as to any of the conditions or exceptions outlined in this Trademark Policy, please contact trademarks@percona.com for assistance and we will do our very best to be helpful.

Contact Us

For free technical help, visit the Percona Community Forum.

To report bugs or submit feature requests, open a JIRA ticket.

For paid support and managed or consulting services , contact Percona Sales.

Last update: June 28, 2023
Created: June 28, 2023