Percona Distribution for PostgreSQL Documentation

11.16 (June 7, 2022)

Percona Technical Documentation Team

Table of contents

1. Percona Distribution for PostgreSQL 11 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:

- pg_repack rebuilds PostgreSQL database objects
- pgAudit provides detailed session or object audit logging via the standard PostgreSQL logging facility
- pgBackRest is a backup and restore solution for PostgreSQL
- Patroni is a HA (High Availability) solution for PostgreSQL.
- pg_stat_monitor collects and aggregates statistics for PostgreSQL and provides histogram information.
- wal2json - a PostgreSQL logical decoding JSON output plugin
- HAProxy - a high-availability and load-balancing solution
- A collection of additional PostgreSQL contrib extensions

See also

Blog Posts

- 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." ^1

1.1 Installation and Upgrade

- Installing Percona Distribution for PostgreSQL
- Minor Upgrade of Percona Distribution for PostgreSQL

1.2 Extensions

- pg_stat_monitor

1.3 Solutions

- High Availability in PostgreSQL with Patroni
  - Deploying high-availability on Debian and Ubuntu
  - Deploying high-availability on RHEL and CentOS
  - Testing the Patroni PostgreSQL Cluster
- Backup and disaster recovery with pgBackRest
  - Deploying backup and disaster recovery solution in Percona Distribution for PostgreSQL

1.4 Uninstall Percona Distribution for PostgreSQL

- Uninstalling Percona Distribution for PostgreSQL

1.5 Release Notes

- Release Notes

1.6 Reference

- Licensing

Last update: 2022-06-07

2. Installation and Upgrade

2.1 Installation and Upgrade

This section provides instructions on how to:

- install Percona Distribution for PostgreSQL 11;
- upgrade Percona Distribution for PostgreSQL to the next minor version.

For how to install one of the latest major versions of Percona Distribution for PostgreSQL, see the following guidelines:

- Installing Percona Distribution for PostgreSQL 12 or
- Installing Percona Distribution for PostgreSQL 13.

Last update: 2021-05-28

2.2 Installing Percona Distribution for PostgreSQL

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

Like many other Percona products, we recommend installing Percona Distribution for PostgreSQL from Percona repositories by using the **percona-release** utility. The **percona-release** utility automatically enables the required repository for you so you can easily install and update Percona Distribution for PostgreSQL packages and their dependencies through the package manager of your operating system.

The installation process includes the following steps:

1. Install **percona-release**
2. Enable the repository
3. Install the packages
4. Start the `postgresql` service
5. Connect to the server

2.2.1 Repositories overview

There are two repositories available for Percona Distribution for PostgreSQL. We recommend installing Percona Distribution for PostgreSQL from the *Major Release repository* (e.g. `ppg-11`) as 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.

The *Minor Release repository* includes a particular minor release of the database and all the packages that were tested and verified to work with that minor release (e.g. `ppg-11.8`). 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.

2.2.2 Install **percona-release**

Install **percona-release** utility. If you have installed it before, update it to the latest version.

2.2.3 Enable the repository

As soon as **percona-release** is installed or up-to-date, enable the repository for Percona Distribution for PostgreSQL (`ppg-11`). We recommend using the *setup* command as it enables the specified repository and updates the platform's package manager database.

To install the *latest* version of Percona Distribution for PostgreSQL, enable the Major release repository using the following command:

```
$ sudo percona-release setup ppg-11
```

To install a *specific minor version* of Percona Distribution for PostgreSQL, enable the Minor release repository. For example, to install Percona Distribution for PostgreSQL 11.10, enable the `ppg-11.10` repository using the following command:

```
$ sudo percona-release setup ppg-11.10
```

2.2.4 Install Percona Distribution for PostgreSQL packages

After you've installed percona-release and enabled the desired repository, install Percona Distribution for PostgreSQL using the commands of your package manager (the procedure differs depending on the package manager of your operating system).

Important

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

On Debian and Ubuntu using `apt`

Note

Debian and other systems that use the `apt` package manager include the upstream PostgreSQL server package (postgresql-11) by default. The components of Percona Distribution for PostgreSQL 11 can only be installed together with the PostgreSQL server shipped by Percona (percona-postgresql-11). If you wish to use Percona Distribution for PostgreSQL, uninstall the PostgreSQL package provided by your distribution (postgresql-11) and then install the chosen components from Percona Distribution for PostgreSQL.

Install the **percona-postgresql-11** package using the following command.

```
$ sudo apt install percona-postgresql-11
```

On Red Hat Enterprise Linux and derivatives using `yum`

PLATFORM SPECIFIC NOTES

If you intend to install Percona Distribution for PostgreSQL on Red Hat Enterprise Linux v8, disable the `postgresql` and `llvm-toolset` modules:

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

On CentOS 7, you should install the `epel-release` package:

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

Install the **percona-postgresql-11** package using **yum install**.

```
$ sudo yum install percona-postgresql11-server
```

Install the Percona Distribution for PostgreSQL components

Use the following commands to install components' packages:

On Debian and Ubuntu

Install `pg_repack`:

```
$ sudo apt install percona-postgresql-11-repack
```

Install `pgAudit`:

```
$ sudo apt install percona-postgresql-11-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-pgaudit11-set-user
```

Install `pgBadger`:

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

Install `wal2json`:

```
$ sudo apt install percona-postgresql-11-wal2json
```

Install PostgreSQL contrib extensions:

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

On Red Hat Enterprise Linux and derivatives

Install `pg_repack`:

```
$ sudo yum install percona-pg_repack11
```

Install `pgAudit`:

```
$ sudo yum install percona-pgaudit
```

Install `pgBackRest`:

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

Install `Patroni`:

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

Install `pg_stat_monitor`:

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

STARTING THE SERVICE

Debian and Ubuntu

The installation process automatically initializes and starts the default database. To check the status of Percona Distribution for PostgreSQL, use the following command:

```
$ sudo systemctl status postgresql
```

RHEL and derivatives

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-11/bin/postgresql-11-setup initdb
```

Start the PostgreSQL service:

```
$ sudo systemctl start postgresql-11
```

2.2.5 Enabling extensions

Some extensions require additional configuration before using them with Percona Distribution for PostgreSQL. This section 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 and Ubuntu 18.04, 20.04.

  For Debian 9 ("stretch"), a DEB package for ETCD is available within Percona Distribution for PostreSQL. You can install it using the following command:

  ```
  $ apt install etcd
  ```

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

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

- HAProxy.

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

Seealso

- 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
```

2.2.6 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
```

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

```
$ \q
```

Hint

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

```
$ sudo su postgres psql
```

Last update: 2022-04-26

2.3 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. Stopping the `postgresql` cluster;
2. Installing new version packages;
3. Restarting 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

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

On Red Hat Enterprise Linux and derivatives

```
$ sudo systemctl stop postgresql-11
```

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

On Debian / Ubuntu

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

On Red Hat Enterprise Linux and derivatives

```
$ sudo systemctl start postgresql-11
```

If you wish to upgrade Percona Distribution for PostgreSQL to the major version, refer to Upgrading Percona Distribution for PostgreSQL from 11 to 12.

Last update: 2022-04-26

3. Extensions

3.1 Extensions

pg_stat_monitor

Last update: 2021-05-28

3.2 pg_stat_monitor

Note

This document describes the functionality of pg_stat_monitor 1.0.0.

3.2.1 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` provides two views:

- `pg_stat_monitor` is the view where statistics data is presented.
- `pg_stat_monitor_settings` view shows available configuration options which you can change.

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`,
- `dbid`,
- `client_ip`,
- `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 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

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.

3.2.2 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.

Assumptions:

We assume that you have installed percona-release utility and enabled the Percona Distribution for PostgreSQL repository

To install `pg_stat_monitor`, run the following command:

On Debian and Ubuntu

```
$ sudo apt-get install percona-pg-stat-monitor11
```

On Red Hat Enterprise Linux and derivatives

```
$ sudo yum install percona-pg-stat-monitor11
```

3.2.3 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.

   Info

   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

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

   On Red Hat Enterprise Linux and derivatives

   ```
   $ sudo systemctl restart postgresql-11
   ```

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

   ```
   CREATE EXTENSION pg_stat_monitor;
   ```

Note

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();
```

3.2.4 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, client_ip
  FROM pg_stat_monitor, pg_database
  WHERE pg_database.oid = oid;

  userid  | datname  |                      query                       | calls | client_ip
----------+----------+--------------------------------------------------+-------+-----------
 postgres | postgres | select bucket, bucket_start_time, query,calls fro |     1 | 127.0.0.1
 postgres | postgres | SELECT c.relchecks, c.relkind, c.relhasindex, c.r |     1 | 127.0.0.1
 postgres | postgres | SELECT  userid,  total_time, min_time, max_time,  |     1 | 127.0.0.1
```

Find more usage examples in the pg_stat_monitor User Guide.

3.2.5 Changing the configuration

Run the following query to list available configuration parameters.

```
SELECT name,description FROM pg_stat_monitor_settings;
```

Output

```
name                     |                            description
------------------------------------------------
+-------------------------------------------------------------------
 pg_stat_monitor.pgsm_max                     | Sets the maximum number of statements tracked
by pg_stat_monitor.
 pg_stat_monitor.pgsm_query_max_len           | Sets the maximum length of query.
 pg_stat_monitor.pgsm_enable                  | Enable/Disable statistics collector.
 pg_stat_monitor.pgsm_track_utility           | Selects whether utility commands are tracked.
 pg_stat_monitor.pgsm_normalized_query        | Selects whether save query in normalized
format.
 pg_stat_monitor.pgsm_max_buckets             | Sets the maximum number of buckets.
 pg_stat_monitor.pgsm_bucket_time             | Sets the time in seconds per bucket.
 pg_stat_monitor.pgsm_histogram_min           | Sets the time in millisecond.
 pg_stat_monitor.pgsm_histogram_max           | Sets the time in millisecond.
 pg_stat_monitor.pgsm_histogram_buckets       | Sets the maximum number of histogram buckets
 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_overflow_target         | Sets the overflow target for pg_stat_monitor
 pg_stat_monitor.pgsm_enable_query_plan       | Enable/Disable query plan monitoring
 pg_stat_monitor.pgsm_track_planning          | Selects whether planning statistics 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 section 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 100 seconds. Use the **ALTER SYSTEM** command:

```
ALTER SYSTEM set pg_stat_monitor.pgsm_bucket_time = 100;
```

Restart the server to apply the change:

On Debian and Ubuntu

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

On Red Hat Enterprise Linux and derivatives

```
$ sudo systemctl restart postgresql-11
```

Verify the updated parameter:

```
SELECT name, value
  FROM pg_stat_monitor_settings
  WHERE name = 'pg_stat_monitor.pgsm_bucket_time';

                name               | value
-----------------------------------+-------
 pg_stat_monitor.pgsm_bucket_time |   100
```

Seealso

`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

Last update: 2022-04-26

4. Solutions

4.1 High availability

4.1.1 High Availability in PostgreSQL with Patroni

Summary

- Solution overview
- Cluster deployment
- Testing the cluster

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.

This document provides instructions on how to set up and test a highly-available, single-primary, three-node cluster with Percona PostgreSQL and Patroni.

High availability overview

There are a few methods for achieving high availability with PostgreSQL:

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

In recent times, PostgreSQL high availability is most commonly achieved with 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.

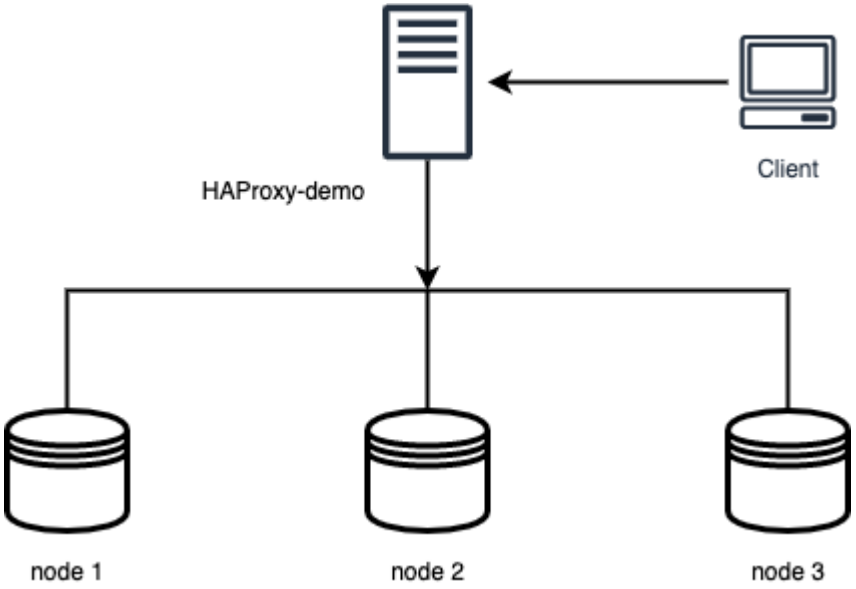
Percona Distribution for PostgreSQL solves this challenge by providing the Patroni extension for achieving PostgreSQL high availability.

Patroni

Patroni provides a template-based approach to create highly available PostgreSQL clusters. Running atop the PostgreSQL streaming replication process, it integrates with watchdog functionality to detect failed primary nodes and take corrective actions to prevent outages. Patroni also provides a pluggable configuration store to manage distributed, multi-node cluster configuration and comes with REST APIs to monitor and manage the cluster. There is also a command-line utility called *patronictl* that helps manage switchovers and failure scenarios.

Architecture layout

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



COMPONENTS

The following are the components:

- Three PosgreSQL nodes: `node1`, `node2` and `node3`
- A dedicated HAProxy node `HAProxy-demo`. HAProxy is an open-source load balancing software through which client connections to the cluster are routed.
- ETCD - a distributed configuration storage
- Softdog - a watchdog utility which is used to detect unhealthy nodes in an acceptable time frame.

Deployment

Use the links below to navigate to the setup instructions relevant to your operating system

- Deploy on Debian or Ubuntu
- Deploy on Red Hat Enterprise Linux or CentOS

Testing

See the Testing PostgreSQL cluster for the guidelines on how to test your PostgreSQL cluster for replication, failure, switchover.

Last update: 2022-04-26

4.1.2 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.

Preconditions

For this setup, we will use the nodes running on Ubuntu 20.04 as the base operating system and having the following IP addresses:

Node name	Public IP address	Internal IP address
node1	157.230.42.174	10.104.0.7
node2	68.183.177.183	10.104.0.2
node3	165.22.62.167	10.104.0.8
HAProxy-demo	134.209.111.138	10.104.0.6

Note

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 DigitalOcean VPS environment, and each node can access the other by its internal, private IP.

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

To make the nodes aware of each other and allow their seamless communication, resolve their hostnames to their public IP addresses. Modify the `/etc/hosts` file of each node as follows:

node 1

```
127.0.0.1 localhost node1
10.104.0.7 node1
10.104.0.2 node2
10.104.0.8 node3
```

node 2

```
127.0.0.1 localhost node2
10.104.0.7 node1
10.104.0.2 node2
10.104.0.8 node3
```

node 3

```
127.0.0.1 localhost node3
10.104.0.7 node1
10.104.0.2 node2
10.104.0.8 node3
```

The `/etc/hosts` file of the HAProxy-demo node looks like the following:

```
127.0.1.1 HAProxy-demo HAProxy-demo
127.0.0.1 localhost
10.104.0.6 HAProxy-demo
10.104.0.7 node1
10.104.0.2 node2
10.104.0.8 node3
```

INSTALL PERCONA DISTRIBUTION FOR POSTGRESQL

1. Follow the installation instructions to install Percona Distribution for PostgreSQL on `node1`, `node2` and `node3`.
2. Remove the data directory. Patroni requires a clean environment to initialize a new cluster. Use the following commands to stop the PostgreSQL service and then remove the data directory:

```
$ sudo systemctl stop postgresql
$ sudo rm -rf /var/lib/postgresql/11/main
```

Configure ETCD distributed store

The distributed configuration store helps establish a consensus among nodes during a failover and will manage the configuration for the three PostgreSQL instances. Although Patroni can work with other distributed consensus stores (i.e., Zookeeper, Consul, etc.), the most commonly used one is `etcd`.

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.

1. Install `etcd` on every PostgreSQL node using the following command:

```
$ sudo apt install etcd
```

2. Modify the `/etc/default/etcd` configuration file on each node.

   - On `node1`, add the IP address of `node1` to the `ETCD_INITIAL_CLUSTER` parameter. The configuration file looks as follows:

```
ETCD_NAME=node1
ETCD_INITIAL_CLUSTER="node1=http://10.104.0.7:2380"
ETCD_INITIAL_CLUSTER_TOKEN="devops_token"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://10.104.0.7:2380"
ETCD_DATA_DIR="/var/lib/etcd/postgresql"
ETCD_LISTEN_PEER_URLS="http://10.104.0.7:2380"
ETCD_LISTEN_CLIENT_URLS="http://10.104.0.7:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://10.104.0.7:2379"
…
```

   - On `node2`, add the IP addresses of both `node1` and `node2` to the `ETCD_INITIAL_CLUSTER` parameter:

```
ETCD_NAME=node2
ETCD_INITIAL_CLUSTER="node1=http://10.104.0.7:2380,node2=http://10.104.0.2:2380"
ETCD_INITIAL_CLUSTER_TOKEN="devops_token"
ETCD_INITIAL_CLUSTER_STATE="existing"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://10.104.0.2:2380"
ETCD_DATA_DIR="/var/lib/etcd/postgresql"
ETCD_LISTEN_PEER_URLS="http://10.104.0.2:2380"
ETCD_LISTEN_CLIENT_URLS="http://10.104.0.2:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://10.104.0.2:2379"
…
```

   - On `node3`, the `ETCD_INITIAL_CLUSTER` parameter includes the IP addresses of all three nodes:

```
ETCD_NAME=node3
ETCD_INITIAL_CLUSTER="node1=http://10.104.0.7:2380,node2=http://10.104.0.2:2380,node3=http://
10.104.0.8:2380"
ETCD_INITIAL_CLUSTER_TOKEN="devops_token"
ETCD_INITIAL_CLUSTER_STATE="existing"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://10.104.0.8:2380"
ETCD_DATA_DIR="/var/lib/etcd/postgresql"
ETCD_LISTEN_PEER_URLS="http://10.104.0.8:2380"
ETCD_LISTEN_CLIENT_URLS="http://10.104.0.8:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://10.104.0.8:2379"
…
```

3. On `node1`, add `node2` and `node3` to the cluster using the `add` command:

```
$ sudo etcdctl member add node2 http://10.104.0.2:2380
$ sudo etcdctl member add node3 http://10.104.0.8:2380
```

4. Restart the `etcd` service on `node2` and `node3`:

```
$ sudo systemctl restart etcd
```

5. Check the etcd cluster members.

```
$ sudo etcdctl member list
```

The output resembles the following:

```
21d50d7f768f153a: name=node1 peerURLs=http://10.104.0.7:2380 clientURLs=http://
10.104.0.7:2379 isLeader=true
af4661d829a39112: name=node2 peerURLs=http://10.104.0.2:2380 clientURLs=http://
10.104.0.2:2379 isLeader=false
e3f3c0c1d12e9097: name=node3 peerURLs=http://10.104.0.8:2380 clientURLs=http://
10.104.0.8:2379 isLeader=false
```

Set up the watchdog service

The Linux kernel uses the utility called a *watchdog* to protect against an unresponsive system. The watchdog monitors a system for unrecoverable application errors, depleted system resources, etc., and initiates a reboot to safely return the system to a working state. The watchdog functionality is useful for servers that are intended to run without human intervention for a long time. Instead of users finding a hung server, the watchdog functionality can help maintain the service.

In this example, we will configure *Softdog* - a standard software implementation for watchdog that is shipped with Ubuntu 20.04.

Complete the following steps on all three PostgreSQL nodes to load and configure Softdog.

1. Load Softdog:

```
$ sudo sh -c 'echo "softdog" >> /etc/modules'
```

2. Patroni will be interacting with the watchdog service. Since Patroni is run by the `postgres` user, this user must have access to Softdog. To make this happen, change the ownership of the `watchdog.rules` file to the `postgres` user:

```
$ sudo sh -c 'echo "KERNEL==\"watchdog\", OWNER=\"postgres\", GROUP=\"postgres\"" >> /etc/
udev/rules.d/61-watchdog.rules'
```

3. Remove Softdog from the blacklist.
   - Find out the files where Softdog is blacklisted:

```
$ grep blacklist /lib/modprobe.d/* /etc/modprobe.d/* |grep softdog
```

   In our case, `modprobe` is blacklisting the Softdog:

```
/lib/modprobe.d/blacklist_linux_5.4.0-73-generic.conf:blacklist softdog
```

   - Remove the `blacklist softdog` line from the `/lib/modprobe.d/blacklist_linux_5.4.0-73-generic.conf` file.
   - Restart the service

```
$ sudo modprobe softdog
```

   - Verify the `modprobe` is working correctly by running the `lsmod` command:

```
$ sudo lsmod | grep softdog
```

   The output will show a process identifier if it's running.

```
softdog                16384  0
```

4. Check that the Softdog files under the `/dev/` folder are owned by the `postgres` user:

```
$ ls -l /dev/watchdog*

crw-rw---- 1 postgres postgres  10, 130 Sep 11 12:53 /dev/watchdog
crw------- 1 root     root     245,   0 Sep 11 12:53 /dev/watchdog0
```

Tip

If the ownership has not been changed for any reason, run the following command to manually change it:

```
$ sudo chown postgres:postgres /dev/watchdog*
```

Configure Patroni

1. Install Patroni on every PostgreSQL node:

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

2. Create the `patroni.yml` configuration file under the `/etc/patroni` directory. The file holds the default configuration values for a PostgreSQL cluster and will reflect the current cluster setup.
3. Add the following configuration for `node1`:

```
scope: stampede1
name: node1

restapi:
  listen: 0.0.0.0:8008
  connect_address: node1:8008

etcd:
  host: node1:2379

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
#    primary_start_timeout: 300
#    synchronous_mode: false
    postgresql:
      use_pg_rewind: true
      use_slots: true
      parameters:
        wal_level: replica
        hot_standby: "on"
        logging_collector: 'on'
        max_wal_senders: 5
        max_replication_slots: 5
        wal_log_hints: "on"
        #archive_mode: "on"
        #archive_timeout: 600
        #archive_command: "cp -f %p /home/postgres/archived/%f"
        #recovery_conf:
        #restore_command: cp /home/postgres/archived/%f %p

  # 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 all all 10.104.0.7/32 md5
  - host replication replicator 127.0.0.1/32 trust
  - host all all 10.104.0.2/32 md5
  - host all all 10.104.0.8/32 md5
  - host all all 10.104.0.6/32 trust
#  - hostssl all all 0.0.0.0/0 md5

  # Additional script to be launched after initial cluster creation (will be passed the
connection URL as parameter)
# post_init: /usr/local/bin/setup_cluster.sh
```

```
  # Some additional users users which needs to be created after initializing new cluster
  users:
    admin:
      password: admin
      options:
        - createrole
        - createdb
    replicator:
      password: password
      options:
        - replication
postgresql:
  listen: 0.0.0.0:5432
  connect_address: node1:5432
  data_dir: "/var/lib/postgresql/11/main"
  bin_dir: "/usr/lib/postgresql/11/bin"
#  config_dir:
  pgpass: /tmp/pgpass0
  authentication:
    replication:
      username: replicator
      password: password
    superuser:
      username: postgres
      password: password
  parameters:
    unix_socket_directories: '/var/run/postgresql'

watchdog:
  mode: required # Allowed values: off, automatic, required
  device: /dev/watchdog
  safety_margin: 5

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 first node (`node1`) 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.

4. Create the configuration files for `node2` and `node3`. Replace the reference to `node1` with `node2` and `node3`, respectively.
5. Enable and restart the patroni service on every node. Use the following commands:

```
$ sudo systemctl enable 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.

Troubleshooting Patroni

To ensure that Patroni has started properly, check the logs using the following command:

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

The output shouldn't show any errors:

```
…

Sep 23 12:50:21 node01 systemd[1]: Started PostgreSQL high-availability manager.
Sep 23 12:50:22 node01 patroni[10119]: 2021-09-23 12:50:22,022 INFO: Selected new etcd server
http://10.104.0.2:2379
Sep 23 12:50:22 node01 patroni[10119]: 2021-09-23 12:50:22,029 INFO: No PostgreSQL configuration
items changed, nothing to reload.
Sep 23 12:50:22 node01 patroni[10119]: 2021-09-23 12:50:22,168 INFO: Lock owner: None; I am node1
Sep 23 12:50:22 node01 patroni[10119]: 2021-09-23 12:50:22,177 INFO: trying to bootstrap a new
cluster
Sep 23 12:50:22 node01 patroni[10140]: The files belonging to this database system will be owned
by user "postgres".
Sep 23 12:50:22 node01 patroni[10140]: This user must also own the server process.
Sep 23 12:50:22 node01 patroni[10140]: The database cluster will be initialized with locale
"C.UTF-8".
Sep 23 12:50:22 node01 patroni[10140]: The default text search configuration will be set to
"english".
Sep 23 12:50:22 node01 patroni[10140]: Data page checksums are enabled.
Sep 23 12:50:22 node01 patroni[10140]: creating directory /var/lib/postgresql/12/main ... ok
Sep 23 12:50:22 node01 patroni[10140]: creating subdirectories ... ok
Sep 23 12:50:22 node01 patroni[10140]: selecting dynamic shared memory implementation ... posix
Sep 23 12:50:22 node01 patroni[10140]: selecting default max_connections ... 100
Sep 23 12:50:22 node01 patroni[10140]: selecting default shared_buffers ... 128MB
Sep 23 12:50:22 node01 patroni[10140]: selecting default time zone ... Etc/UTC
Sep 23 12:50:22 node01 patroni[10140]: creating configuration files ... ok
Sep 23 12:50:22 node01 patroni[10140]: running bootstrap script ... ok
Sep 23 12:50:23 node01 patroni[10140]: performing post-bootstrap initialization ... ok
Sep 23 12:50:23 node01 patroni[10140]: syncing data to disk ... ok
Sep 23 12:50:23 node01 patroni[10140]: initdb: warning: enabling "trust" authentication for
local connections
Sep 23 12:50:23 node01 patroni[10140]: You can change this by editing pg_hba.conf or using the
option -A, or
Sep 23 12:50:23 node01 patroni[10140]: --auth-local and --auth-host, the next time you run
initdb.
Sep 23 12:50:23 node01 patroni[10140]: Success. You can now start the database server using:
Sep 23 12:50:23 node01 patroni[10140]:     /usr/lib/postgresql/11/bin/pg_ctl -D /var/lib/
postgresql/11/main -l logfile start
Sep 23 12:50:23 node01 patroni[10156]: 2021-09-23 12:50:23.672 UTC [10156] LOG:  redirecting log
output to logging collector process
Sep 23 12:50:23 node01 patroni[10156]: 2021-09-23 12:50:23.672 UTC [10156] HINT:  Future log
output will appear in directory "log".
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,694 INFO: postprimary pid=10156
Sep 23 12:50:23 node01 patroni[10165]: localhost:5432 - accepting connections
Sep 23 12:50:23 node01 patroni[10167]: localhost:5432 - accepting connections
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,743 INFO: establishing a new patroni
connection to the postgres cluster
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,757 INFO: running post_bootstrap
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,767 INFO: Software Watchdog activated
with 25 second timeout, timing slack 15 seconds
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,793 INFO: initialized a new cluster
Sep 23 12:50:33 node01 patroni[10119]: 2021-09-23 12:50:33,810 INFO: no action. I am (node1) the
leader with the lock
Sep 23 12:50:33 node01 patroni[10119]: 2021-09-23 12:50:33,899 INFO: no action. I am (node1) the
leader with the lock
Sep 23 12:50:43 node01 patroni[10119]: 2021-09-23 12:50:43,898 INFO: no action. I am (node1) the
leader with the lock
Sep 23 12:50:53 node01 patroni[10119]: 2021-09-23 12:50:53,894 INFO: no action. I am (node1) the
leader with the
```

> 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
```

The command output is the following

```
psql (11.13)
Type "help" for help.

postgres=#
```

Configure HAProxy

HAProxy node will accept client connection requests and route those to the active node of the PostgreSQL cluster. This way, a client application doesn't have to know what node in the underlying cluster is the current primary. All it needs to do is to access a single HAProxy URL and send its read/write requests there. Behind-the-scene, HAProxy routes the connection to a healthy node (as long as there is at least one healthy node available) and ensures that client application requests are never rejected.

HAProxy is capable of routing 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.

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

```
$ sudo apt install 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
```

Testing

See the Testing PostgreSQL cluster for the guidelines on how to test your PostgreSQL cluster for replication, failure, switchover.

Last update: 2022-04-26

4.1.3 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.

Preconditions

For this setup, we will use the nodes running on CentOS 8 as the base operating system and having the following IP addresses:

Hostname	Public IP address	Internal IP address
node1	157.230.42.174	10.104.0.7
node2	68.183.177.183	10.104.0.2
node3	165.22.62.167	10.104.0.8
etcd	159.102.29.166	10.104.0.5
HAProxy-demo	134.209.111.138	10.104.0.6

Note

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 DigitalOcean VPS environment, and each node can access the other by its internal, private IP.

Setting up hostnames in the `/etc/hosts` file

To make the nodes aware of each other and allow their seamless communication, resolve their hostnames to their public IP addresses. Modify the `/etc/hosts` file of each PostgreSQL node to include the hostnames and IP addresses of the remaining nodes. The following is the `/etc/hosts` file for `node1`:

```
127.0.0.1 localhost node1
10.104.0.7 node1
10.104.0.2 node2
10.104.0.8 node3
```

The `/etc/hosts` file of the `HAProxy-demo` node hostnames and IP addresses of all PostgreSQL nodes:

```
127.0.1.1 HAProxy-demo HAProxy-demo
127.0.0.1 localhost
10.104.0.6 HAProxy-demo
10.104.0.7 node1
10.104.0.2 node2
10.104.0.8 node3
```

Keep the `/etc/hosts` file of the `etcd` node unchanged.

Configure ETCD distributed store

The distributed configuration store helps establish a consensus among nodes during a failover and will manage the configuration for the three PostgreSQL instances. Although Patroni can work with other distributed consensus stores (i.e., Zookeeper, Consul, etc.), the most commonly used one is `etcd`.

In this setup we will configure ETCD on a dedicated node.

1. Install `etcd` on the ETCD node. For CentOS 8, the etcd packages are available from Percona repository:
2. Install `percona-release`.
3. Enable the repository:

```
$ sudo percona-release setup ppg11
```

4. Install the etcd packages using the following command:

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

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

```
[Member]
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="http://10.104.0.5:2380,http://localhost:2380"
ETCD_LISTEN_CLIENT_URLS="http://10.104.0.5:2379,http://localhost:2379"


ETCD_NAME="default"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://10.104.0.5:2380"
ETCD_ADVERTISE_CLIENT_URLS="http://10.104.0.5:2379"
ETCD_INITIAL_CLUSTER="default=http://10.104.0.5:2380"
ETCD_INITIAL_CLUSTER_TOKEN="etcd-cluster"
ETCD_INITIAL_CLUSTER_STATE="new"
```

1. Start the `etcd` to apply the changes:

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

2. Check the etcd cluster members.

```
$ 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
```

Install Percona Distribution for PostgreSQL

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

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

```
$ sudo percona-release setup ppg11
```

3. 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.

Configure Patroni

1. Install Patroni on every PostgreSQL node:

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

2. Install the Python module that enables Patroni to communicate with ETCD.

```
$ sudo python3 -m pip install patroni[etcd]
```

3. 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 for Patroni. Change its ownership to the `postgres` user and restrict the access to it

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

4. Create the `patroni.yml` configuration file.

```
$ su postgres
$ vim /etc/patroni/patroni.yml
```

5. Specify the following configuration:

```
scope: postgres
namespace: /pg_cluster/
name: node1

restapi:
  listen: 10.104.0.7:8008            # PostgreSQL node IP address
  connect_address: 10.104.0.7:8008   # PostgreSQL node IP address
```

```
etcd:
  host: 10.104.0.5:2379  # ETCD node IP address

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
    postgresql:
      use_pg_rewind: true
      use_slots: true
      parameters:
        wal_level: replica
        hot_standby: "on"
        logging_collector: 'on'
        max_wal_senders: 5
        max_replication_slots: 5
        wal_log_hints: "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 md5
  - host replication replicator 10.104.0.2/32 md5
  - host replication replicator 10.104.0.8/32 md5
  - host replication replicator 10.104.0.7/32 md5
  - host all all 0.0.0.0/0 md5
#  - hostssl all all 0.0.0.0/0 md5

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

postgresql:
  listen: 10.104.0.7:5432            # PostgreSQL node IP address
  connect_address: 10.104.0.7:5432   # PostgreSQL node IP address
  data_dir: /data/patroni            # The datadir you created
  bin_dir: /usr/pgsql-11/bin
  pgpass: /tmp/pgpass0
  authentication:
    replication:
      username: replicator
      password: replicator
    superuser:
      username: postgres
      password: postgres
  parameters:
    unix_socket_directories: '.'

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

6. Create the configuration files for `node2` and `node3`. Replace the node and IP address of `node1` to those of `node2` and `node3`, respectively.
7. Create the systemd unit file `patroni.service` in `/etc/systemd/system`.

```
$ sudo vim /etc/systemd/system/patroni.service
```

Add the following contents in the file:

```
[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
```

8. Make systemd aware of the new service:

```
$ sudo systemctl daemon-reload
$ sudo systemctl enable patroni
$ sudo systemctl start patroni
```

Troubleshooting Patroni

To ensure that Patroni has started properly, check the logs using the following command:

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

The output shouldn't show any errors:

```
…

Sep 23 12:50:21 node01 systemd[1]: Started PostgreSQL high-availability manager.
Sep 23 12:50:22 node01 patroni[10119]: 2021-09-23 12:50:22,022 INFO: Selected new etcd server
http://10.104.0.2:2379
Sep 23 12:50:22 node01 patroni[10119]: 2021-09-23 12:50:22,029 INFO: No PostgreSQL
configuration items changed, nothing to reload.
Sep 23 12:50:22 node01 patroni[10119]: 2021-09-23 12:50:22,168 INFO: Lock owner: None; I am
node1
Sep 23 12:50:22 node01 patroni[10119]: 2021-09-23 12:50:22,177 INFO: trying to bootstrap a new
cluster
Sep 23 12:50:22 node01 patroni[10140]: The files belonging to this database system will be
owned by user "postgres".
Sep 23 12:50:22 node01 patroni[10140]: This user must also own the server process.
Sep 23 12:50:22 node01 patroni[10140]: The database cluster will be initialized with locale
"C.UTF-8".
Sep 23 12:50:22 node01 patroni[10140]: The default text search configuration will be set to
"english".
Sep 23 12:50:22 node01 patroni[10140]: Data page checksums are enabled.
Sep 23 12:50:22 node01 patroni[10140]: creating directory /var/lib/postgresql/12/main ... ok
Sep 23 12:50:22 node01 patroni[10140]: creating subdirectories ... ok
Sep 23 12:50:22 node01 patroni[10140]: selecting dynamic shared memory implementation ... posix
Sep 23 12:50:22 node01 patroni[10140]: selecting default max_connections ... 100
Sep 23 12:50:22 node01 patroni[10140]: selecting default shared_buffers ... 128MB
Sep 23 12:50:22 node01 patroni[10140]: selecting default time zone ... Etc/UTC
Sep 23 12:50:22 node01 patroni[10140]: creating configuration files ... ok
Sep 23 12:50:22 node01 patroni[10140]: running bootstrap script ... ok
Sep 23 12:50:23 node01 patroni[10140]: performing post-bootstrap initialization ... ok
Sep 23 12:50:23 node01 patroni[10140]: syncing data to disk ... ok
Sep 23 12:50:23 node01 patroni[10140]: initdb: warning: enabling "trust" authentication for
local connections
Sep 23 12:50:23 node01 patroni[10140]: You can change this by editing pg_hba.conf or using the
option -A, or
Sep 23 12:50:23 node01 patroni[10140]: --auth-local and --auth-host, the next time you run
initdb.
Sep 23 12:50:23 node01 patroni[10140]: Success. You can now start the database server using:
Sep 23 12:50:23 node01 patroni[10140]:     /usr/lib/postgresql/11/bin/pg_ctl -D /var/lib/
postgresql/11/main -l logfile start
Sep 23 12:50:23 node01 patroni[10156]: 2021-09-23 12:50:23.672 UTC [10156] LOG:  redirecting
log output to logging collector process
Sep 23 12:50:23 node01 patroni[10156]: 2021-09-23 12:50:23.672 UTC [10156] HINT:  Future log
output will appear in directory "log".
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,694 INFO: postprimary pid=10156
Sep 23 12:50:23 node01 patroni[10165]: localhost:5432 - accepting connections
Sep 23 12:50:23 node01 patroni[10167]: localhost:5432 - accepting connections
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,743 INFO: establishing a new patroni
connection to the postgres cluster
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,757 INFO: running post_bootstrap
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,767 INFO: Software Watchdog
activated with 25 second timeout, timing slack 15 seconds
Sep 23 12:50:23 node01 patroni[10119]: 2021-09-23 12:50:23,793 INFO: initialized a new cluster
Sep 23 12:50:33 node01 patroni[10119]: 2021-09-23 12:50:33,810 INFO: no action. I am (node1)
the leader with the lock
Sep 23 12:50:33 node01 patroni[10119]: 2021-09-23 12:50:33,899 INFO: no action. I am (node1)
the leader with the lock
Sep 23 12:50:43 node01 patroni[10119]: 2021-09-23 12:50:43,898 INFO: no action. I am (node1)
the leader with the lock
```

```
Sep 23 12:50:53 node01 patroni[10119]: 2021-09-23 12:50:53,894 INFO: no action. I am (node1)
the leader with the
```

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 (11.13)
Type "help" for help.

postgres=#
```

9. Configure, enable and start Patroni on the remaining nodes.
10. 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


+ Cluster: postgres (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 |
+--------+-------+---------+---------+----+-----------+
```

Configure HAProxy

HAProxy node will accept client connection requests and route those to the active node of the PostgreSQL cluster. This way, a client application doesn't have to know what node in the underlying cluster is the current primary. All it needs to do is to access a single HAProxy URL and send its read/write requests there. Behind-the-scene, HAProxy routes the connection to a healthy node (as long as there is at least one healthy node available) and ensures that client application requests are never rejected.

HAProxy is capable of routing 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.

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

```
$ sudo yum install 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
```

Last update: 2022-04-26

4.1.4 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

+ 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
```

```
+ 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: 2022-04-26

4.2 Backup and disaster recovery

4.2.1 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**"

```
<br>
PostgreSQL offers multiple options for setting up database disaster recovery.

- **[pg_dump](https://www.postgresql.org/docs/11/app-pgdump.html) or the [pg_dumpall](https://
www.postgresql.org/docs/11/app-pg-dumpall.html) 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](https://www.postgresql.org/docs/11/app-pgrestore.html) command.

| Advantages   | Disadvantages   |
| ------------ | --------------- |
| Easy to use  | 1. Backup of only one database at a time.<br>2. No incremental backups.<br>3.
No point-in-time recovery since the backup is a snapshot in time.<br>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.<br> 2.
```

```
No backup of individual databases or tables.|

- **PostgreSQL [pg_basebackup](https://www.postgresql.org/docs/11/app-pgbasebackup.html)**

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.<br>2. Supports point-in-time recovery | 1.
No incremental backups.<br>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.

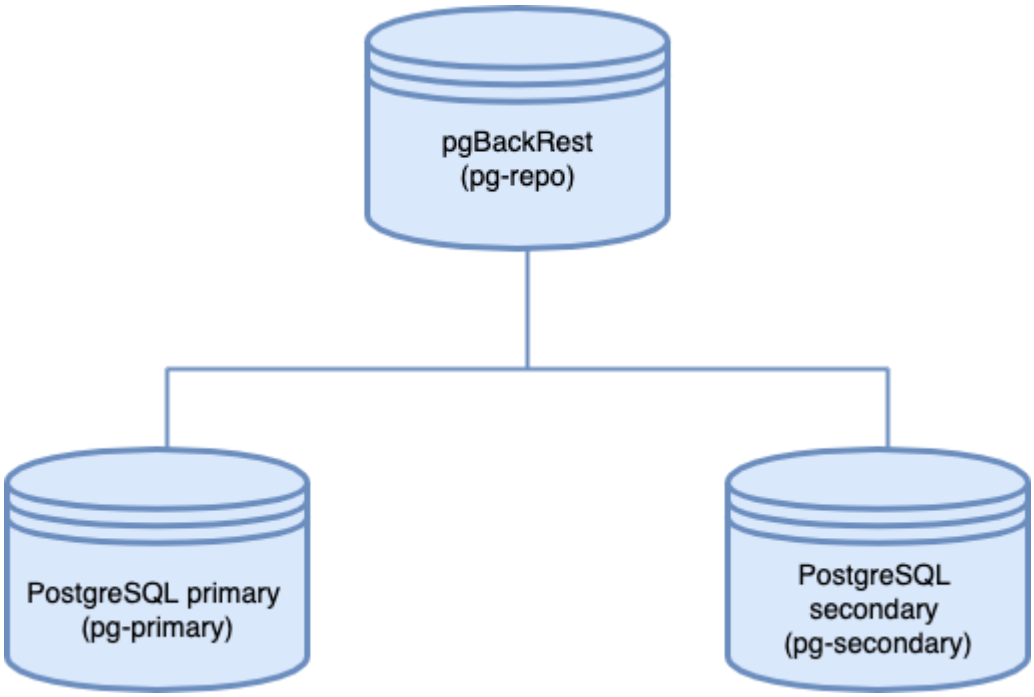
SYSTEM ARCHITECTURE

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.

Warning

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.

Last update: 2022-04-26

4.2.2 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 ppg11
```

3. Install Percona Distribution for PostgreSQL packages

On Debian and Ubuntu

```
$ sudo apt install percona-postgresql-11 -y
```

On RedHat Enterprise Linux and derivatives

```
$ sudo yum install percona-postgresql11-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/11/main/postgresql.conf`.

On RHEL and derivatives, the path to the configuration file is `/var/lib/pgsql/11/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

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

On RHEL and derivatives

```
$ 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/11/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/11/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/11/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/11/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/11/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/11/
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/
11/main'
2021-11-07 13:34:11.803 P00   INFO: write updated /var/lib/postgresql/11/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.

```
postgres=# select * from customer;
 id |  name
----+--------
  1 | john
  2 | martha
  3 | mary
(3 rows)
```

Last update: 2022-04-26

4.3 LDAP authentication

4.3.1 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: 2022-06-07

5. Uninstall

5.1 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`

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 PostgreSQL service.

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

2. Remove the **percona-postgresql** packages.

```
$ sudo apt remove percona-postgresql-11* percona-pgbackrest percona-patroni percona-pgbadger percona-pgbouncer
```

3. Remove configuration and data files.

```
$ rm -rf /etc/postgresql/11/main
```

On Red Hat Enterprise Linux and derivatives using `yum`

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 PostgreSQL service.

```
$ sudo systemctl stop postgresql-11
```

2. Remove the **percona-postgresql** packages

```
$ sudo yum remove percona-postgresql11* percona-pgbadger
```

3. Remove configuration and data files

```
$ rm -rf /var/lib/pgsql/11/data
```

Last update: 2022-04-26

6. Release Notes

6.1 Release Notes

- Percona Distribution for PostgreSQL 11.16
- Percona Distribution for PostgreSQL 11.15 Second Update
- Percona Distribution for PostgreSQL 11.15 Update
- Percona Distribution for PostgreSQL 11.15
- Percona Distribution for PostgreSQL 11.14
- Percona Distribution for PostgreSQL 11.13 Update
- Percona Distribution for PostgreSQL 11.13
- Percona Distribution for PostgreSQL 11.12 Third Update
- Percona Distribution for PostgreSQL 11.12 Second Update
- Percona Distribution for PostgreSQL 11.12 Update
- Percona Distribution for PostgreSQL 11.12
- Percona Distribution for PostgreSQL 11.11 Third Update
- Percona Distribution for PostgreSQL 11.11 Second Update
- Percona Distribution for PostgreSQL 11.11 Update
- Percona Distribution for PostgreSQL 11.11
- Percona Distribution for PostgreSQL 11.10 Update
- Percona Distribution for PostgreSQL 11.10
- Percona Distribution for PostgreSQL 11.9
- Percona Distribution for PostgreSQL 11.8
- Percona Distribution for PostgreSQL 11.7
- Percona Distribution for PostgreSQL 11.6
- Percona Distribution for PostgreSQL 11
- Percona Distribution for PostgreSQL 11 (Beta)

Last update: 2022-06-07

6.2 Percona Distribution for PostgreSQL 11.16 (2022-06-07)

Date: June 7, 2022

Installation: Installing Percona Distribution for PostgreSQL

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

This release is based on PostgreSQL 11.16. The set of extensions supplied with Percona Distribution for PostgreSQL now includes the HAProxy - a high-availability and load-balancing solution.

Extension	Version	Description
Patroni	2.1.3	a HA (High Availability) solution for PostgreSQL
PgAudit	1.3.4	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 superusers or object owner roles in order to perform needed maintenance tasks.
pgBackRest	2.38	a backup and restore solution for PostgreSQL
pgBadger	11.8	a fast PostgreSQL Log Analyzer.
PgBouncer	1.17.0	a lightweight connection pooler for PostgreSQL
pg_repack	1.4.7	rebuilds PostgreSQL database objects
pg_stat_monitor	1.0.1	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	2.4	a PostgreSQL logical decoding JSON output plugin
HAProxy	2.5.6	a high-availability and load-balancing solution

Percona Distribution for PostgreSQL also includes the following packages:

- `llvm` 12.0.1 packages for Red Hat Enterprise Linux 8 and 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 7	`python3-python-etcd`	0.4.3	A Python client for ETCD
RHEL 8	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-python-etcd`	0.4.3	A Python client for ETCD
Debian 9 ('stretch')	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-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."

Last update: 2022-06-07

6.3 Percona Distribution for PostgreSQL 11.15 Second Update (2022-05-05)

Date: May 5, 2022

Installation: Installing Percona Distribution for PostgreSQL

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

This update of Percona Distribution for PostgreSQL includes the general availability release of pg_stat_monitor 1.0.0 - the statistics collection tool for PostgreSQL.

We welcome your feedback on your experience with `pg_stat_monitor` on our Forum and in the public JIRA project.

Last update: 2022-06-07

6.4 Percona Distribution for PostgreSQL 11.15 Update (2022-04-14)

Date: April 14, 2022

Installation: Installing Percona Distribution for PostgreSQL

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

This update of Percona Distribution for PostgreSQL includes pg_stat_monitor 1.0.0-rc.2 - the new version of the statistics collection tool for PostgreSQL.

We welcome your feedback on your experience with `pg_stat_monitor` on our Forum and in the public JIRA project.

Last update: 2022-06-07

6.5 Percona Distribution for PostgreSQL 11.15 (2022-04-08)

Date: April 8, 2022

Installation: Installing Percona Distribution for PostgreSQL

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

This release is based on PostgreSQL 11.15 and includes the extended set of extensions supplied with Percona Distribution for PostgreSQL.

Extension	Version	Description
Patroni	2.1.2	a HA (High Availability) solution for PostgreSQL
PgAudit	1.3.3	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 superusers or object owner roles in order to perform needed maintenance tasks.
pgBackRest	2.37	a backup and restore solution for PostgreSQL
pgBadger	11.7	a fast PostgreSQL Log Analyzer.
PgBouncer	1.16.1	a lightweight connection pooler for PostgreSQL
pg_repack	1.4.7	rebuilds PostgreSQL database objects
pg_stat_monitor	1.0.0-rc.1	collects and aggregates statistics for PostgreSQL and provides histogram information.
wal2json	2.4	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 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 7	`python3-python-etcd`	0.4.3	A Python client for ETCD
RHEL 8	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-python-etcd`	0.4.3	A Python client for ETCD
Debian 9 ('stretch')	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-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."

Last update: 2022-06-07

6.6 Percona Distribution for PostgreSQL 11.14 (2021-12-20)

Date: December 20, 2021

Installation: Installing Percona Distribution for PostgreSQL

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

This release is based on PostgreSQL 11.14 and includes the extended set of extensions supplied with Percona Distribution for PostgreSQL.

Extension	Version	Description
Patroni	2.1.1	a HA (High Availability) solution for PostgreSQL
PgAudit	1.3.2	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 superusers or object owner roles in order to perform needed maintenance tasks.
pgBackRest	2.36	a backup and restore solution for PostgreSQL
pgBadger	11.6	a fast PostgreSQL Log Analyzer.
PgBouncer	1.16.1	a lightweight connection pooler for PostgreSQL
pg_repack	1.4.7	rebuilds PostgreSQL database objects
pg_stat_monitor	1.0.0-rc.1	collects and aggregates statistics for PostgreSQL and provides histogram information.
wal2json	2.4	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 7	`python3-python-etcd`	0.4.3	A Python client for ETCD
CentOS 8	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-python-etcd`	0.4.3	A Python client for ETCD
Debian 9 ('stretch')	`etcd`	3.3.11	A consistent, distributed key-value store
	`python3-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."

Last update: 2022-06-07

6.7 Percona Distribution for PostgreSQL 11.13 Update (2021-12-07)

Date: December 7, 2021

Installation: Installing Percona Distribution for PostgreSQL

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

This update of Percona Distribution for PostgreSQL includes the new version of pg_stat_monitor 1.0.0-RC - the statistics collection tool for PostgreSQL.

We welcome your feedback on your experience with `pg_stat_monitor` in the public JIRA project.

Last update: 2022-06-07

6.8 Percona Distribution for PostgreSQL 11.13 (2021-09-09)

Date: September 9, 2021

Installation: Installing Percona Distribution for PostgreSQL

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

This release is based on PostgreSQL 11.13 and includes the extended set of extensions supplied with Percona Distribution for PostgreSQL.

Extension	Version	Description
Patroni	2.1.0	a HA (High Availability) solution for PostgreSQL
PgAudit	1.3.2	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
pgAudit set_user	2.0.1	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	2.34	a backup and restore solution for PostgreSQL
pgBadger	11.5	a fast PostgreSQL Log Analyzer.
PgBouncer	1.16.0	a lightweight connection pooler for PostgreSQL
pg_repack	1.4.6	rebuilds PostgreSQL database objects
pg_stat_monitor	0.9.2 - Beta1	collects and aggregates statistics for PostgreSQL and provides histogram information.
wal2json	2.3	a PostgreSQL logical decoding JSON output plugin

Percona Distribution for PostgreSQL also includes the ETCD packages which are used for Patroni cluster setup. These packages are available for the following operating systems:

Operating System	Package	Description
CentOS 7	`python3-python-etcd`	A Python client for ETCD
CentOS 8	`etcd`	A consistent, distributed key-value store
	`python3-python-etcd`	A Python client for ETCD
Debian 9 ('stretch')	`etcd`	A consistent, distributed key-value store
	`python3-etcd`	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."^1

Last update: 2022-06-07

6.9 Percona Distribution for PostgreSQL 11.12 Third Update (2021-07-15)

Date: July 15, 2021

Installation: Installing Percona Distribution for PostgreSQL

This update of Percona Distribution for PostgreSQL, includes the RPM package for `python3-python-etcd` for CentOS 7. This package is a Python client for ETCD and is used by Patroni to communicate with ETCD storage. For how to set up Patroni clusters, see Patroni documentation.

Last update: 2022-06-07

6.10 Percona Distribution for PostgreSQL 11.12 Second Update (2021-07-01)

Date: July 1, 2021

Installation: Installing Percona Distribution for PostgreSQL

With this update of Percona Distribution for PostgreSQL, `etcd` package is added as a `DEB` package to Percona Distribution for PostgreSQL for Debian 9 ("stretch"). This package is used to set up High Availability clusters with Patroni. For how to set up Patroni clusters, see Patroni documentation.

Last update: 2022-06-07

6.11 Percona Distribution for PostgreSQL 11.12 Update (2021-06-10)

Date: June 10, 2021

Installation: Installing Percona Distribution for PostgreSQL

This update of Percona Distribution for PostgreSQL includes the following fixes for Red Hat Enterprise Linux 8 / CentOS 8:

- `llvm` packages are added to the repository. This fixes compatibility issues with LLVM from upstream. To use `llvm` packages supplied by us, disable the upstream `llvm-toolset` module before the installation:

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

- systemd unit file includes the correct path to Patroni configuration file.
- `etcd` and `python3-python-etcd` packages are added as `RPM` packages to Percona Distribution for PostgreSQL for Red Hat Enterprise Linux / CentOS 8. These packages are used to set up High Availability clusters with Patroni. For how to set up Patroni clusters, see Patroni documentation

Last update: 2022-06-07

6.12 Percona Distribution for PostgreSQL 11.12 (2021-05-24)

Date: May 24, 2021

Installation: Installing Percona Distribution for PostgreSQL

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

This release is based on PostgreSQL 11.12 and includes the extended set of extensions supplied with Percona Distribution for PostgreSQL.

Extension	Version	Description
pg_repack	1.4.6	rebuilds PostgreSQL database objects
PgAudit	1.3.2	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
pgAudit set_user	2.0.0	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	2.33	a backup and restore solution for PostgreSQL
Patroni	2.0.2	a HA (High Availability) solution for PostgreSQL
pg_stat_monitor (Tech Preview Feature [^1])		0.9.1
pgBadger	11.5	a fast PostgreSQL Log Analyzer.
PgBouncer	1.15.0	a lightweight connection pooler for PostgreSQL
wal2json	2.3	a PostgreSQL logical decoding JSON output plugin
PostgreSQL contrib extensions	11.11	a collection of additional extensions for PostgreSQL

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."^2

[^1]: Tech Preview Features are not yet ready for enterprise use and are not included in support via . They are included in this release, so that users can provide feedback prior to the full release of the feature in a future release (or removal of the feature if it is deemed not useful). This functionality can change (APIs, CLIs, etc.) from tech preview to GA.

Last update: 2022-06-07

6.13 Percona Distribution for PostgreSQL 11.11 Third Update (2021-06-10)

Date: June 10, 2021

Installation: Installing Percona Distribution for PostgreSQL

This update of Percona Distribution for PostgreSQL includes `llvm` packages for Red Hat Enterprise Linux 8 / CentOS 8. This fixes compatibility issues with LLVM from upstream. To use `llvm` packages supplied by us, disable the upstream `llvm-toolset` module before the installation:

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

Last update: 2022-06-07

6.14 Percona Distribution for PostgreSQL 11.11 Second Update (2021-05-10)

Date: May 10, 2021

Installation: Installing Percona Distribution for PostgreSQL

This update of Percona Distribution for PostgreSQL includes the latest version of pg_stat_monitor 0.9.0 - the statistics collection tool for PostgreSQL. `pg_stat_monitor` is available as the Tech Preview Feature [^1] and is supplied in the set of extensions within Percona Distribution for PostgreSQL.

We welcome your feedback on your experience with `pg_stat_monitor` in the public JIRA project.

[^1]: Tech Preview Features are not yet ready for enterprise use and are not included in support via SLA. They are included in this release, so that users can provide feedback prior to the full release of the feature in a future GA release (or removal of the feature if it is deemed not useful). This functionality can change (APIs, CLIs, etc.) from tech preview to GA.

Last update: 2022-06-07

6.15 Percona Distribution for PostgreSQL 11.11 Update (2021-04-12)

Date: April 12, 2021

Installation: Installing Percona Distribution for PostgreSQL

This update of Percona Distribution for PostgreSQL includes the latest version of pg_stat_monitor 0.8.1 - the statistics collection tool for PostgreSQL. `pg_stat_monitor` is available as the Tech Preview Feature [^1] and is supplied in the set of extensions within Percona Distribution for PostgreSQL.

We welcome your feedback on your experience with `pg_stat_monitor` in the public JIRA project.

[^1]: Tech Preview Features are not yet ready for enterprise use and are not included in support via SLA. They are included in this release, so that users can provide feedback prior to the full release of the feature in a future GA release (or removal of the feature if it is deemed not useful). This functionality can change (APIs, CLIs, etc.) from tech preview to GA.

Last update: 2022-06-07

6.16 Percona Distribution for PostgreSQL 11.11 (2021-03-08)

Date: March 8, 2021

Installation: Installing Percona Distribution for PostgreSQL

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

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

Extension	Version	Description
pg_repack	1.4.6	rebuilds PostgreSQL database objects
Pgaudit	1.3.2	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
pgBackRest	2.32	a backup and restore solution for PostgreSQL
Patroni	2.0.1	a HA (High Availability) solution for PostgreSQL
pg_stat_monitor (Tech Preview Feature [^1])		0.6.0
PostgreSQL contrib extensions	11.11	a collection of additional extensions for PostgreSQL

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." ^2

This release is based on PostgreSQL 11.11.

[^1]: Tech Preview Features are not yet ready for enterprise use and are not included in support via SLA. They are included in this release, so that users can provide feedback prior to the full release of the feature in a future GA release (or removal of the feature if it is deemed not useful). This functionality can change (APIs, CLIs, etc.) from tech preview to GA.

Last update: 2022-06-07

6.17 Percona Distribution for PostgreSQL 11.10 Update (2021-06-10)

Date: June 10, 2021

Installation: Installing Percona Distribution for PostgreSQL

This update of Percona Distribution for PostgreSQL includes `llvm` packages for Red Hat Enterprise Linux 8 / CentOS 8. This fixes compatibility issues with LLVM from upstream. To use `llvm` packages supplied by us, disable the upstream `llvm-toolset` module before the installation:

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

Last update: 2022-06-07

6.18 Percona Distribution for PostgreSQL 11.10 (2020-12-15)

Date: December 15, 2020

Installation: Installing Percona Distribution for PostgreSQL

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

This release is based on PostgreSQL 11.10 and includes pg_stat_monitor (Tech Preview Feature) - a new statistics collection extension for PostgreSQL.

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

Extension	Version	Description
pg_repack	1.4.6	rebuilds PostgreSQL database objects
Pgaudit	1.3.2	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
pgBackRest	2.30	a backup and restore solution for PostgreSQL
Patroni	2.0.1	a HA (High Availability) solution for PostgreSQL
PostgreSQL contrib extensions	11.10	a collection of additional extensions for PostgreSQL

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." ^1

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 11.10.

Last update: 2022-06-07

6.19 Percona Distribution for PostgreSQL 11.9

Date: September 8, 2020

Installation: Installing Percona Distribution for PostgreSQL

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

Extension	Version	Description
pg_repack	1.4.5	rebuilds PostgreSQL database objects
Pgaudit	1.4.0	provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
pgBackRest	2.29	a backup and restore solution for PostgreSQL
Patroni	2.0.0	a HA (High Availability) solution for PostgreSQL
PostgreSQL contrib extensions	11.9	a collection of additional extensions for PostgreSQL

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." ^1

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 11.9.

Last update: 2021-05-28

6.20 Percona Distribution for PostgreSQL 11.8

Date: June 11, 2020

Installation: Installing Percona Distribution for PostgreSQL

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

- Pg_repack rebuilds PostgreSQL database objects.
- Pgaudit provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
- pgBackRest - a backup and restore solution for PostgreSQL
- Patroni - an HA (High Availability) solution for PostgreSQL
- A collection of additional PostgreSQL contrib extensions

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." ^1

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 11.8.

Last update: 2021-05-28

6.21 Percona Distribution for PostgreSQL 11.7

Date: April 9, 2020

Installation: Installing Percona Distribution for PostgreSQL

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

- Pg_repack rebuilds PostgreSQL database objects.
- Pgaudit provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
- pgBackRest - a backup and restore solution for PostgreSQL
- Patroni - an HA (High Availability) solution for PostgreSQL
- A collection of additional PostgreSQL contrib extensions

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." ^1

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 11.7.

Starting from May 15, 2020, this release of Percona Distribution for PostgreSQL includes improvements to simplify the upgrade to the major version of Percona Distribution for PostgreSQL.

Last update: 2021-05-28

6.22 Percona Distribution for PostgreSQL 11.6

Percona is happy to announce the release of Percona Distribution for PostgreSQL 11.6 on January 23, 2020. Percona Distribution for PostgreSQL is a collection of tools to assist you in managing PostgreSQL. Percona Distribution for PostgreSQL installs PostgreSQL and complements it by a selection of extensions that enable solving essential practical tasks efficiently:

- Pg_repack rebuilds PostgreSQL database objects.
- Pgaudit provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
- pgBackRest - a backup and restore solution for PostgreSQL
- Patroni - an HA (High Availability) solution for PostgreSQL
- A collection of additional PostgreSQL contrib extensions

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." ^1

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 11.6.

> **Important**
>
> On Debian and other systems that use the `apt` package manager, such as Ubuntu, components of Percona Distribution for PostgreSQL 11 can only be installed together with the server shipped by Percona (`percona-postgresql-11`). If you wish to use Percona Distribution for PostgreSQL, uninstall the PostgreSQL package provided by your distribution (postgresql-11) and then install the chosen components from Percona Distribution for PostgreSQL.

6.22.1 Bugs Fixed

- DISTPG-34: Components shipped with Percona Distribution for PostgreSQL could not be installed alongside PostgreSQL.

Last update: 2021-05-28

6.23 Percona Distribution for PostgreSQL 11

Percona is excited to announce the GA release of Percona Distribution for PostgreSQL 11 on September 17, 2019. Percona Distribution for PostgreSQL is a collection of tools to assist you in managing PostgreSQL. Percona Distribution for PostgreSQL installs PostgreSQL and complements it by a selection of extensions that enable solving essential practical tasks efficiently:

- Pg_repack rebuilds PostgreSQL database objects.
- Pgaudit provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
- pgBackRest - a backup and restore solution for PostgreSQL
- Patroni - an HA solution for PostgreSQL
- A collection of additional PostgreSQL contrib extensions

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." ^1

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 11.

Last update: 2021-05-28

6.24 Percona Distribution for PostgreSQL 11 (Beta) (2019-05-15)

Percona is excited to announce a beta release of Percona Distribution for PostgreSQL 11 (Beta) on August 15, 2019. Percona Distribution for PostgreSQL is a collection of tools to assist you in managing PostgreSQL. Percona Distribution for PostgreSQL installs PostgreSQL and complements it by a selection of extensions that enable solving essential practical tasks efficiently:

- Pg_repack rebuilds PostgreSQL database objects.
- Pgaudit provides detailed session or object audit logging via the standard logging facility provided by PostgreSQL
- pgBackRest - a backup and restore solution for PostgreSQL
- Patroni - an HA (High Availability) solution for PostgreSQL
- A collection of additional PostgreSQL contrib extensions

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." ^1

This release of Percona Distribution for PostgreSQL is based on PostgreSQL 11.

Last update: 2022-06-07

7. Licensing

7.1 Licensing

Percona Distribution for PostgreSQL is licensed under the PostgreSQL license and licenses of all components included in the Distribution.

7.1.1 Documentation Licensing

Percona Distribution for PostgreSQL documentation is licensed under the PostgreSQL license.

Last update: 2021-05-28