pgagroal

Connection pool for PostgreSQL

Contents

1 Introduction

pgagroal is a high-performance protocol-native connection pool for PostgreSQL.

1.1 Features

- High performance
- Connection pool
- Limit connections for users and databases
- Prefill support
- Remove idle connections
- Perform connection validation
- Enable / disable database access
- Graceful / fast shutdown
- Prometheus support
- Grafana 12 dashboard
- Remote management
- Authentication query support
- Failover support
- Transport Layer Security (TLS) v1.2+ support
- Daemon mode
- User vault

1.2 Platforms

The supported platforms are

- Fedora 42+
- RHEL 10 / RockyLinux 10
- FreeBSD
- OpenBSD

1.3 How to Use This Manual

This manual is organized to guide you from initial setup to advanced usage and development. Use the table below to quickly find the section most relevant to your needs:

> **Navigation Note**: Each entry has two links separated by |: - **First link (Chapter)**: Use when reading the PDF manual (jumps to page) - **Second link (File)**: Use when browsing individual markdown files - File links will not work in PDF format

Chapter	File	Description
Installation	02-installation.md	Step-by-step setup for Rocky Linux, PostgreSQL 18, and pgagroal
Getting Started	03-gettingstarted.md	Quick introduction to basic pgagroal usage and initial configuration
Configuration	04-configuration.md	Comprehensive guide to all configuration files and options
Prefill	05-prefill.md	How to configure and use connection prefill for better performance
Remote Management	06-remote_management.md	Setting up and using remote management features for pgagroal
Split Security	07-split_security.md	Implementing split security models for authentication and access control
TLS	08-tls.md	Configuring Transport Layer Security (TLS) for secure connections
Database Alias	09-database_alias.md	Using database aliases for flexible client connections
Vault	10-vault.md	Managing user credentials and secrets with the pgagroal vault
Prometheus	11-prometheus.md	Integrating Prometheus metrics and monitoring
Docker	12-docker.md	Running pgagroal in Docker containers
Command Line Tools	13-cli-tools.md	Comprehensive CLI tools reference (pgagroal-cli, pgagroal-admin)

Chapter	File	Description
Performance	14-performance.md	Performance benchmarks, tuning, and optimization
Failover	15-failover.md	Failover configuration and scripting
Pipelines	16-pipelines.md	Pipeline types and configuration
Security	17-security.md	Comprehensive security hardening guide
Development	70-dev.md	Development environment setup and contribution guidelines
Git	71-git.md	Git workflow and version control practices for the project
Architecture	72-architecture.md	High-level architecture and design of pgagroal
RPM	73-rpm.md	Building and using RPM packages
Building	74-building.md	Compiling pgagroal from source
Code Coverage	75-codecoverage.md	Code coverage analysis and testing practices
Event Loop	76-eventloop.md	Understanding the event loop implementation
Core API	77-core_api.md	Reference for core API functions
Testing	78-test.md	Testing frameworks and procedures
Distribution Installation	79-distributions.md	Platform-specific installation notes
Acknowledgements	97-acknowledgement.md	Credits and contributors
Licenses	98-licenses.md	License information
References	99-references.md	Additional resources and references

2 Installation

2.1 Rocky Linux

We can download the Rocky Linux distruction from their web site

```
https://rockylinux.org/download
```

The installation and setup is beyond the scope of this guide.

Ideally, you would use dedicated user accounts to run **PostgreSQL** and **pgagroal**

```
useradd postgres
usermod -a -G wheel postgres
useradd pgagroal
usermod -a -G wheel pgagroal
```

Add a configuration directory for **pgagroal**

```
mkdir /etc/pgagroal
chown -R pgagroal:pgagroal /etc/pgagroal
```

and lets open the ports in the firewall that we will need

```
firewall-cmd --permanent --zone=public --add-port=2345/tcp
firewall-cmd --permanent --zone=public --add-port=2346/tcp
```

2.2 PostgreSQL

We will install PostgreSQL 18 from the official [YUM repository][yum] with the community binaries,

x86_64

```
dnf -qy module disable postgresql
dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL
    -10-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

aarch64

```
dnf -qy module disable postgresql
dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL
    -10-aarch64/pgdg-redhat-repo-latest.noarch.rpm
```

and do the install via

```
dnf install -y postgresql18 postgresql18-server postgresql18-contrib
```

First, we will update ~/.bashrc with

```
cat >> ~/.bashrc
export PGHOST=/tmp
export PATH=/usr/pgsql-18/bin/:$PATH
```

then Ctrl-d to save, and

```
source ~/.bashrc
```

to reload the Bash environment.

Then we can do the PostgreSQL initialization

```
mkdir DB
initdb -k DB
```

and update configuration - for a 8 GB memory machine.

postgresql.conf

```
listen_addresses = '*'
port = 5432
max_connections = 100
unix_socket_directories = '/tmp'
password_encryption = scram-sha-256
shared_buffers = 2GB
huge_pages = try
max_prepared_transactions = 100
work_mem = 16MB
dynamic_shared_memory_type = posix
wal_level = replica
wal_log_hints = on
max_wal_size = 16GB
min_wal_size = 2GB
log_destination = 'stderr'
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql.log'
log_rotation_age = 0
log_rotation_size = 0
log_truncate_on_rotation = on
log_line_prefix = '%p [%m] [%x] '
log_timezone = UTC
datestyle = 'iso, mdy'
timezone = UTC
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
summarize_wal = on
wal_summary_keep_time = '0'
```

Please, check with other sources in order to create a setup for your local setup.

Now, we are ready to start PostgreSQL

```
pg_ctl -D DB -l /tmp/ start
```

2.3 pgagroal

We will install **pgagroal** from the official [YUM repository][yum] as well,

```
dnf install -y pgagroal
```

First, we will need to create a master security key for the **pgagroal** installation, by

```
pgagroal-admin -g master-key
```

By default, this will ask for a key interactively. Alternatively, a key can be provided using either the `--password` command line argument, or the `PGAGROAL_PASSWORD` environment variable. Note that passing the key using the command line might not be secure.

Then we will create the configuration for **pgagroal**,

```
cat > /etc/pgagroal/pgagroal.conf
[pgagroal]
host = *
port = 2345

metrics = 2346

log_type = file
log_level = info
log_path = /tmp/pgagroal.log

max_connections = 100
idle_timeout = 600
validation = off
unix_socket_dir = /tmp/

[primary]
host = localhost
port = 5432
```

and end with a Ctrl-d to save the file.

Start **pgagroal** now, by

```
pgagroal -d
```

3 Getting started

First of all, make sure that **pgagroal** is installed and in your path by using `pgagroal -?`. You should see

```
pgagroal 2.0.0
  High-performance connection pool for PostgreSQL

Usage:
  pgagroal [ -c CONFIG_FILE ] [ -a HBA_FILE ] [ -d ]

Options:
  -c, --config CONFIG_FILE           Set the path to the pgagroal.conf
      file
  -a, --hba HBA_FILE                 Set the path to the pgagroal_hba.conf
       file
  -l, --limit LIMIT_FILE             Set the path to the
      pgagroal_databases.conf file
  -u, --users USERS_FILE             Set the path to the pgagroal_users.
      conf file
  -F, --frontend FRONTEND_USERS_FILE Set the path to the
      pgagroal_frontend_users.conf file
  -A, --admins ADMINS_FILE           Set the path to the pgagroal_admins.
      conf file
  -S, --superuser SUPERUSER_FILE     Set the path to the
      pgagroal_superuser.conf file
  -D, --directory DIRECTORY_PATH     Set the path to load configuration
      files
  -d, --daemon                       Run as a daemon
  -V, --version                      Display version information
  -?, --help                         Display help
```

If you don't have **pgagroal** in your path see README on how to compile and install **pgagroal** in your system.

3.1 Configuration

Lets create a simple configuration file called `pgagroal.conf` with the content

```
[pgagroal]
host = *
port = 2345

log_type = file
log_level = info
log_path = /tmp/pgagroal.log

max_connections = 100
```

```
idle_timeout = 600
validation = off
unix_socket_dir = /tmp/

[primary]
host = localhost
port = 5432
```

In our main section called `[pgagroal]` we setup **pgagroal** to listen on all network addresses on port 2345. Logging will be performed at `info` level and put in a file called `/tmp/pgagroal.log`. We want a maximum of 100 connections that are being closed if they have been idle for 10 minutes, and we also specify that we don't want any connection validation to be performed. Last we specify the location of the `unix_socket_dir` used for management operations.

Next we create a section called `[primary]` which has the information about our PostgreSQL instance. In this case it is running on `localhost` on port `5432`.

Now we need a host based authentication (HBA) file. Create one called `pgagroal_hba.conf` with the content

```
#
# TYPE  DATABASE USER  ADDRESS  METHOD
#
host    all      all   all      all
```

This tells **pgagroal** that it can accept connections from all network addresses for all databases and all user names.

We are now ready to run **pgagroal**.

See Configuration for all configuration options.

3.2 Running

We will run **pgagroal** using the command

```
pgagroal -c pgagroal.conf -a pgagroal_hba.conf
```

If this doesn't give an error, then we are ready to connect.

We will assume that we have a user called `test` with the password `test` in our PostgreSQL instance. See their documentation on how to setup PostgreSQL, add a user and add a database.

We will connect to **pgagroal** using the psql application.

```
psql -h localhost -p 2345 -U test test
```

That should give you a password prompt where `test` should be typed in. You are now connected to PostgreSQL through **pgagroal**.

Type `\q` to quit psql and **pgagroal** will now put the connection that you used into its pool.

If you type the above `psql` command again **pgagroal** will reuse the existing connection and thereby lower the overhead of getting a connection to PostgreSQL.

Now you are ready to point your applications to use **pgagroal** instead of going directly to PostgreSQL. **pgagroal** will work with any PostgreSQL compliant driver, for example pgjdbc, Npgsql and pq.

pgagroal is stopped by pressing Ctrl-C (`^C`) in the console where you started it, or by sending the `SIGTERM` signal to the process using `kill <pid>`.

3.3 Run-time administration

pgagroal has a run-time administration tool called `pgagroal-cli`.

You can see the commands it supports by using `pgagroal-cli -?` which will give

```
pgagroal-cli 2.0.0
  Command line utility for pgagroal

Usage:
  pgagroal-cli [ OPTIONS ] [ COMMAND ]

Options:
  -c, --config CONFIG_FILE Set the path to the pgagroal.conf file
                           Default: /etc/pgagroal/pgagroal.conf
  -h, --host HOST          Set the host name
  -p, --port PORT          Set the port number
  -U, --user USERNAME      Set the user name
  -P, --password PASSWORD  Set the password
  -L, --logfile FILE       Set the log file
  -F, --format text|json   Set the output format
  -v, --verbose            Output text string of result
  -V, --version            Display version information
  -?, --help               Display help

Commands:
  flush [mode] [database]  Flush connections according to <mode>.
                           Allowed modes are:
                           - 'gracefully' (default) to flush all
                             connections gracefully
                           - 'idle' to flush only idle connections
                           - 'all' to flush all connections. USE WITH
                             CAUTION!
                           If no <database> name is specified, applies to
                             all databases.
```

```
  ping                      Verifies if pgagroal is up and running
  enable   [database]       Enables the specified databases (or all
     databases)
  disable  [database]       Disables the specified databases (or all
     databases)
  shutdown [mode]           Stops pgagroal pooler. The <mode> can be:
                            - 'gracefully' (default) waits for active
                               connections to quit
                            - 'immediate' forces connections to close and
                               terminate
                            - 'cancel' avoid a previously issued 'shutdown
                               gracefully'
  status [details]          Status of pgagroal, with optional details
  switch-to <server>        Switches to the specified primary server
  conf <action>             Manages the configuration (e.g., reloads the
     configuration
                            The subcommand <action> can be:
                            - 'reload' to issue a configuration reload;
                            - 'get' to obtain information about a runtime
                               configuration value;
                                    conf get <parameter_name>
                            - 'set' to modify a configuration value;
                                    conf set <parameter_name> <
                                       parameter_value>;
                            - 'ls'  lists the configuration files used.
  clear <what>              Resets either the Prometheus statistics or the
     specified server.
                            <what> can be
                            - 'server' (default) followed by a server name
                            - a server name on its own
                            - 'prometheus' to reset the Prometheus metrics

pgagroal: <https://pgagroal.github.io/>
Report bugs: <https://github.com/pgagroal/pgagroal/issues>
```

This tool can be used on the machine running **pgagroal** to flush connections.

To flush all idle connections you would use

```
pgagroal-cli -c pgagroal.conf flush idle
```

To stop pgagroal you would use

```
pgagroal-cli -c pgagroal.conf stop
```

Check the outcome of the operations by verifying the exit code, like

```
echo $?
```

or by using the `-v` flag.

If pgagroal has both Transport Layer Security (TLS) and `management` enabled then `pgagroal-cli` can connect with TLS using the files `~/.pgagroal/pgagroal.key` (must be 0600 permission), `~/.pgagroal/pgagroal.crt` and `~/.pgagroal/root.crt`.

3.4 Administration

pgagroal has an administration tool called `pgagroal-admin`, which is used to control user registration with **pgagroal**.

You can see the commands it supports by using `pgagroal-admin -?` which will give

```
pgagroal-admin 2.0.0
 Administration utility for pgagroal

Usage:
  pgagroal-admin [ -f FILE ] [ COMMAND ]

Options:
  -f, --file FILE         Set the path to a user file
                          Defaults to /etc/pgagroal/pgagroal_users.conf
  -U, --user USER         Set the user name
  -P, --password PASSWORD Set the password for the user
  -g, --generate          Generate a password
  -l, --length            Password length
  -V, --version           Display version information
  -?, --help              Display help

Commands:
  master-key              Create or update the master key
  user <subcommand>       Manage a specific user, where <subcommand> can
     be
                          - add  to add a new user
                          - del  to remove an existing user
                          - edit to change the password for an existing
                             user
                          - ls   to list all available users

pgagroal: https://pgagroal.github.io/
Report bugs: https://github.com/pgagroal/pgagroal/issues
```

In order to set the master key for all users you can use

```
pgagroal-admin -g master-key
```

The master key must be at least 8 characters if provided interactively.

For scripted use, the master key can be provided using the `PGAGROAL_PASSWORD` environment variable.

Then use the other commands to add, update, remove or list the current user names, f.ex.

```
pgagroal-admin -f pgagroal_users.conf user add
```

For scripted use, the user password can be provided using the `PGAGROAL_PASSWORD` environment variable.

3.5 Next Steps

Next steps in improving pgagroal's configuration could be

- Update `pgagroal.conf` with the required settings for your system
- Set the access rights in `pgagroal_hba.conf` for each user and database
- Add a `pgagroal_users.conf` file using `pgagroal-admin` with a list of known users
- Disable access for unknown users by setting `allow_unknown_users` to **`false`**
- Define a `pgagroal_databases.conf` file with the limits and prefill settings for each database
- Enable Transport Layer Security v1.2+ (TLS)
- Deploy Grafana dashboard

See Configuration for more information on these subjects.

Please, read the manual for a full description of all the features available.

3.6 Contribute

The pgagroal community hopes that you find the project interesting.

Feel free to

- Ask a question
- Raise an issue
- Submit a feature request
- Write a code submission

All contributions are most welcome !

Please, consult our Code of Conduct policies for interacting in our community.

Consider giving the project a star on GitHub if you find it useful. And, feel free to follow the project on X as well.

4 Configuration

The configuration is loaded from either the path specified by the `-c` flag or `/etc/pgagroal/pgagroal.conf`.

The configuration of **pgagroal** is split into sections using the `[` and `]` characters.

The main section, called `[pgagroal]`, is where you configure the overall properties of **pgagroal**.

Other sections doesn't have any requirements to their naming so you can give them meaningful names like `[primary]` for the primary PostgreSQL instance.

All properties are in the format `key = value`.

The characters # and `;` can be used for comments; must be the first character on the line.

The `Bool` data type supports the following values: `on`, `yes`, `1`, **`true`**, `off`, `no`, `0` and **`false`**.

See a sample configuration for running **pgagroal** on `localhost`.

4.1 pgagroal.conf

This section is mandatory and the pooler will refuse to start if the configuration file does not specify one and only one. Usually this section is place on top of the configuration file, but its position within the file does not really matter. The available keys and their accepted values are reported in the table below.

Property	Default	Unit	Required	Description
host		String	Yes	The bind address for pgagroal
port		Int	Yes	The bind port for pgagroal
unix_socket_dir		String	Yes	The Unix Domain Socket location
metrics	0	Int	No	The metrics port (disable = 0)

Property	Default	Unit	Required	Description
metrics_cache_max_age	0	String	No	The amount of time to keep a Prometheus (metrics) response in cache. If this value is specified without units, it is taken as seconds. It supports the following units as suffixes: 'S' for seconds (default), 'M' for minutes, 'H' for hours, 'D' for days, and 'W' for weeks. (disable = 0)
metrics_cache_max_	256k	String	No	The maximum amount of data to keep in cache when serving Prometheus responses. Changes require restart. This parameter determines the size of memory allocated for the cache even if `metrics_cache_max_age` or `metrics` are disabled. Its value, however, is taken into account only if `metrics_cache_max_age` is set to a non-zero value. Supports suffixes: 'B' (bytes), the default if omitted, 'K' or 'KB' (kilobytes), 'M' or 'MB' (megabytes), 'G' or 'GB' (gigabytes).

Property	Default	Unit	Required	Description
management	0	Int	No	The remote management port (disable = 0)
log_type	console	String	No	The logging type (console, file, syslog)
log_level	info	String	No	The logging level, any of the (case insensitive) strings `FATAL`, `ERROR`, `WARN`, `INFO` and `DEBUG` (that can be more specific as `DEBUG1` thru `DEBUG5`). Debug level greater than 5 will be set to `DEBUG5`. Not recognized values will make the log_level be `INFO`
log_path	pgagroal.log	String	No	The log file location. Can be a strftime(3) compatible string.
log_rotation_age	0	String	No	The amount of time after which log file rotation is triggered. If this value is specified without units, it is taken as seconds. It supports the following units as suffixes: ‘S’ for seconds (default), ‘M’ for minutes, ‘H’ for hours, ‘D’ for days, and ‘W’ for weeks. (disable = 0)

Property	Default	Unit	Required	Description
log_rotation_size	0	String	No	The size of the log file that will trigger a log rotation. Supports suffixes: ‘B’ (bytes), the default if omitted, ‘K’ or ‘KB’ (kilobytes), ‘M’ or ‘MB’ (megabytes), ‘G’ or ‘GB’ (gigabytes). A value of `0` (with or without suffix) disables.
log_line_prefix	%Y-%m-%d %H:%M:%S	String	No	A strftime(3) compatible string to use as prefix for every log line. Must be quoted if contains spaces.
log_mode	append	String	No	Append to or create the log file (append, create)
log_connections	`off`	Bool	No	Log connects
log_disconnections	`off`	Bool	No	Log disconnects
blocking_timeout	30	String	No	The amount of time the process will be blocking for a connection. If this value is specified without units, it is taken as seconds. It supports the following units as suffixes: ‘S’ for seconds (default), ‘M’ for minutes, ‘H’ for hours, ‘D’ for days, and ‘W’ for weeks. (disable = 0)

Property	Default	Unit	Required	Description
idle_timeout	0	String	No	The amount of time a connection is kept alive. If this value is specified without units, it is taken as seconds. It supports the following units as suffixes: ‘S’ for seconds (default), ‘M’ for minutes, ‘H’ for hours, ‘D’ for days, and ‘W’ for weeks. (disable = 0)
rotate_frontend_password_timeout	0	String	No	The amount of time after which the passwords of frontend users are updated periodically. If this value is specified without units, it is taken as seconds. It supports the following units as suffixes: ‘S’ for seconds (default), ‘M’ for minutes, ‘H’ for hours, ‘D’ for days, and ‘W’ for weeks. (disable = 0)
rotate_frontend_pas	8	Int	No	The length of the randomized frontend password

Property	Default	Unit	Required	Description
max_connection_age	0	String	No	The maximum amount of time that a connection will live. If this value is specified without units, it is taken as seconds. It supports the following units as suffixes: 'S' for seconds (default), 'M' for minutes, 'H' for hours, 'D' for days, and 'W' for weeks. (disable = 0)
validation	`off`	String	No	Should connection validation be performed. Valid options: `off`, `foreground` and `background`
background_interval	300	String	No	The interval between background validation scans. If this value is specified without units, it is taken as seconds. It supports the following units as suffixes: 'S' for seconds (default), 'M' for minutes, 'H' for hours, 'D' for days, and 'W' for weeks.
max_retries	5	Int	No	The maximum number of iterations to obtain a connection
max_connections	100	Int	No	The maximum number of connections to PostgreSQL (max 10000)

Property	Default	Unit	Required	Description
allow_unknown_use	`true`	Bool	No	Allow unknown users to connect
authentication_timeout	5	String	No	The amount of time the process will wait for valid credentials. If this value is specified without units, it is taken as seconds. It supports the following units as suffixes: 'S' for seconds (default), 'M' for minutes, 'H' for hours, 'D' for days, and 'W' for weeks.
pipeline	`auto`	String	No	The pipeline type (`auto`, `performance`, `session`, `transaction`)
auth_query	`off`	Bool	No	Enable authentication query
failover	`off`	Bool	No	Enable failover support
failover_script		String	No	The failover script to execute
tls	`off`	Bool	No	Enable Transport Layer Security (TLS)
tls_cert_file		String	No	Certificate file for TLS. This file must be owned by either the user running pgagroal or root.

Property	Default	Unit	Required	Description
tls_key_file		String	No	Private key file for TLS. This file must be owned by either the user running pgagroal or root. Additionally permissions must be at least `0640` when owned by root or `0600` otherwise.
tls_ca_file		String	No	Certificate Authority (CA) file for TLS. This file must be owned by either the user running pgagroal or root.
metrics_cert_file		String	No	Certificate file for TLS for Prometheus metrics. This file must be owned by either the user running pgagroal or root.
metrics_key_file		String	No	Private key file for TLS for Prometheus metrics. This file must be owned by either the user running pgagroal or root. Additionally permissions must be at least `0640` when owned by root or `0600` otherwise.

Property	Default	Unit	Required	Description
metrics_ca_file		String	No	Certificate Authority (CA) file for TLS for Prometheus metrics. This file must be owned by either the user running pgagroal or root.
libev	`auto`	String	No	Select the libev backend to use. Valid options: `auto`, `select`, `poll`, `epoll`, `iouring`, `devpoll` and `port`
keep_alive	on	Bool	No	Have `SO_KEEPALIVE` on sockets
nodelay	on	Bool	No	Have `TCP_NODELAY` on sockets
backlog	`max_connectic` / 4	Int	No	The backlog for `listen()`. Minimum 16
hugepage	**`try`**	String	No	Huge page support (`off`, **`try`**, `on`)
tracker	off	Bool	No	Track connection lifecycle
track_prepared_statements	off	Bool	No	Track prepared statements (transaction pooling)
pidfile		String	No	Path to the PID file. If omitted, automatically set to `unix_socket_dir` /pgagroal.`port`.pid

Property	Default	Unit	Required	Description
update_process_title	`verbose`	String	No	The behavior for updating the operating system process title, mainly related to connection processes. Allowed settings are: `never` (or `off`), does not update the process title; `strict` to set the process title without overriding the existing initial process title length; `minimal` to set the process title to `username`/`database`; `verbose` (or `full`) to set the process title to `user@host:port`/`database`. Please note that `strict` and `minimal` are honored only on those systems that do not provide a native way to set the process title (e.g., Linux). On other systems, there is no difference between `strict` and `minimal` and the assumed behaviour is `minimal` even if `strict` is used. `never` and `verbose` are always honored, on every system. On Linux systems the process title is always trimmed to 255 characters, while on system that provide a natve way to set the process title it can be longer.

Danger zone

Property	Default	Unit	Required	Description
disconnect_client	0	Int	No	Disconnect clients that have been idle for more than the specified seconds. This setting **DOES NOT** take long running transactions into account
disconnect_client_fc	off	Bool	No	Disconnect clients that have been active for more than the specified seconds. This setting __DOES NOT_** take long running transactions into account

Server section

Each section with a name different from **pgagroal** will be treated as an host section. There can be up to 64 host sections, each with an unique name and different combination of `host` and `port` settings, otherwise the pooler will complain about duplicated server configuration.

Property	Default	Unit	Required	Description
host		String	Yes	The address of the PostgreSQL instance
port		Int	Yes	The port of the PostgreSQL instance
primary		Bool	No	Identify the instance as primary (hint)

Property	Default	Unit	Required	Description
tls	`off`	Bool	No	Enable Transport Layer Security (TLS) support (Experimental - no pooling). Changes require restart.
tls_cert_file		String	No	Certificate file for TLS. This file must be owned by either the user running pgagroal or root. Changes require restart.
tls_key_file		String	No	Private key file for TLS. This file must be owned by either the user running pgagroal or root. Additionally permissions must be at least `0640` when owned by root or `0600` otherwise.Changes require restart.
tls_ca_file		String	No	Certificate Authority (CA) file for TLS. This file must be owned by either the user running pgagroal or root. Changes require restart.

Note, that if `host` starts with a `/` it represents a path and **pgagroal** will connect using a Unix Domain Socket.

4.2 pgagroal_hba.conf

The `pgagroal_hba` configuration controls access to **pgagroal** through host-based authentication.

The configuration is loaded from either the path specified by the `-a` flag or `/etc/pgagroal/pgagroal_hba.conf`.

The format of the file follows the similar PostgreSQL HBA configuration format, and as such looks like

```
#
# TYPE  DATABASE USER  ADDRESS  METHOD
#
host    all      all   all      all
```

Column	Required	Description
TYPE	Yes	Specifies the access method for clients. `host` and `hostssl` are supported
DATABASE	Yes	Specifies the database for the rule. Either specific name or `all` for all databases
USER	Yes	Specifies the user for the rule. Either specific name or `all` for all users
ADDRESS	Yes	Specifies the network for the rule. `all` for all networks, or IPv4 address with a mask (`0.0.0.0/0`) or IPv6 address with a mask (`::0/0`)
METHOD	Yes	Specifies the authentication mode for the user. `all` for all methods, otherwise `trust`, `reject`, `password`, `md5` or `scram-sha-256`

Remote management users needs to have their database set to `admin` in order for the entry to be considered.

There can be up to 64 HBA entries in the configuration file.

4.3 pgagroal_databases.conf

The `pgagroal_databases` configuration defines limits for a database or a user or both. The limits are the number of connections from **pgagroal** to PostgreSQL for each entry.

The file also defines the initial and minimum pool size for a database and user pair. Note, that this feature requires a user definition file, see below.

The configuration is loaded from either the path specified by the `-l` flag or `/etc/pgagroal/pgagroal_databases.conf`.

```
#
# DATABASE USER    MAX_SIZE INITIAL_SIZE MIN_SIZE
#
mydb       myuser  all
anotherdb  userB   10           5        3
```

Column	Required	Description
DATABASE	Yes	Specifies the database for the rule. `all` for all databases
USER	Yes	Specifies the user for the rule. `all` for all users
MAX_SIZE	Yes	Specifies the maximum pool size for the entry. `all` for all remaining counts from `max_connections`
INITIAL_SIZE	No	Specifies the initial pool size for the entry. `all` for `MAX_SIZE` connections. Default is 0
MIN_SIZE	No	Specifies the minimum pool size for the entry. `all` for `MAX_SIZE` connections. Default is 0

Database Aliases

Database aliases allow clients to connect using alternative names for a configured database. This is useful for: - Application migrations where legacy database names need to be supported - Multi-tenancy scenarios where different clients use different logical names - Providing user-friendly names without exposing actual backend database names

Alias Rules and Behavior

- **Alias Resolution**: When a client connects using an alias, pgagroal automatically resolves it to the real database name before establishing or reusing backend connections
- **Connection Pooling**: Connections established with the real database name can be reused by clients connecting with any of its aliases
- **Transparent Mapping**: All authentication queries and backend communication use the real database name

- **Uniqueness**: Aliases must be unique across all database entries and cannot conflict with any real database name
- **Limit**: Maximum 8 aliases per database entry

Configuration Examples

```
# Database with aliases
production_db=prod,main,primary     myuser   10  5  2
development_db=dev,test,staging,qa  devuser    5  2  1
new_app_db=legacy_app,old_db        appuser  15  8  3
```

There can be up to 64 entries in the configuration file.

In the case a limit entry has incoherent values, for example `INITIAL_SIZE` smaller than `MIN_SIZE`, the system will try to automatically adjust the settings on the fly, reporting messages in the logs.

The system will find the best match limit entry for a given `DATABASE`-`USER` pair according to the following rules: 1. Use the first entry with an exact `DATABASE` and `USER` match. 2. If there is no exact match, use the entry with a `USER` match and `DATABASE` set to `all`. 3. If Rule 2 does not apply, use the entry with a `DATABASE` match and `USER` set to `all`.

Note: For alias matching in Rule 1, if a client connects using an alias name, pgagroal will find the entry where the alias is defined and treat it as an exact database match.

Alias Validation

The configuration system validates aliases to ensure: - No duplicate aliases within the same entry - No alias conflicts with main database names in other entries - No duplicate aliases across different entries - Aliases are not empty strings - Total alias count does not exceed the maximum limit

Changes to aliases can be reloaded without restarting pgagroal, making it easy to add or modify aliases for existing databases.

4.4 pgagroal_users.conf

The `pgagroal_users` configuration defines the users known to the system. This file is created and managed through the `pgagroal-admin` tool.

The configuration is loaded from either the path specified by the `-u` flag or `/etc/pgagroal/pgagroal_users.conf`.

4.5 pgagroal_frontend_users.conf

The `pgagroal_frontend_users` configuration defines the passwords for the users connecting to pgagroal. This allows the setup to use different passwords for the **pgagroal** to PostgreSQL authentica-

tion. This file is created and managed through the pgagroal-admin tool.

All users defined in the frontend authentication must be defined in the user vault (-u).

Frontend users (-F) requires a user vault (-u) to be defined.

The configuration is loaded from either the path specified by the -F flag or /etc/pgagroal/pgagroal_frontend_users.conf.

4.6 pgagroal_admins.conf

The pgagroal_admins configuration defines the administrators known to the system. This file is created and managed through the pgagroal-admin tool.

The configuration is loaded from either the path specified by the -A flag or /etc/pgagroal/pgagroal_admins.conf.

If pgagroal has both Transport Layer Security (TLS) and management enabled then pgagroal-cli can connect with TLS using the files ~/.pgagroal/pgagroal.key (must be 0600 permission), ~/.pgagroal/pgagroal.crt and ~/.pgagroal/root.crt.

4.7 pgagroal_superuser.conf

The pgagroal_superuser configuration defines the superuser known to the system. This file is created and managed through the pgagroal-admin tool. It may only have one user defined.

The configuration is loaded from either the path specified by the -S flag or /etc/pgagroal/pgagroal_superuser.conf.

4.8 Configuration directory

You can specify a directory for all configuration files using the -D flag (or --directory). Alternatively, you can set the PGAGROAL_CONFIG_DIR environment variable to define the configuration directory.

Behavior: - When the directory flag (-D) is set, pgagroal will look for all configuration files in the specified directory. - If a required file is not found in the specified directory, pgagroal will look for it in its default location (e.g., /etc/pgagroal/pgagroal.conf). - If the file is not found in either location: - If the file is mandatory, pgagroal will log an error and fail to start. - If the file is optional, pgagroal will log a warning and continue without it. - All file lookup attempts and missing files are logged for troubleshooting.

Precedence Rules: - Individual file flags (such as `-c`, `-a`, `-l`, etc.) always take precedence over the directory flag and environment variable for their respective files. - The directory flag (`-D`) takes precedence over the environment variable (`PGAGROAL_CONFIG_DIR`). - If neither the directory flag nor individual file flags are set, pgagroal uses the default locations for all configuration files.

Using the Environment Variable: 1. Set the environment variable before starting pgagroal: `export PGAGROAL_CONFIG_DIR=/path/to/config_dir pgagroal -d` 2. If both the environment variable and the `-D` flag are set, the flag takes precedence.

Example:

```
pgagroal -D /custom/config/dir -d
```

or

```
export PGAGROAL_CONFIG_DIR=/custom/config/dir
pgagroal -d
```

Refer to logs for details about which configuration files were loaded and from which locations.

5 Prefill

Create prefill configuration

Prefill is instrumented by the `pgagroal_databases.conf` configuration file, where you need to list databases, usernames, and limits. Every username/database pair has to be specified on a separated line.

The limits are assumed as:

- *max number of allowed connections* for that username/database
- *initial number of connections*, that is the effective prefill;
- *minimum number of connections* to always keep open for the pair username/database.

Assuming you want to configure the prefill for the `mydb` database with the `myuser` username, you have to edit the file `/etc/pgagroal/pgagroal_databases.conf` with your editor of choice or using `cat` from the command line, as follows:

```
cd /etc/pgagroal
cat > pgagroal_databases.conf
mydb   myuser   2   1   0
```

and press `Ctrl-d` to save the file.

This will create a configuration where `mydb` will have a maximum connection size of 2, an initial connection size of 1 and a minimum connection size of 0 for the `myuser` user.

The file must be owned by the operating system user **pgagroal**.

The `max_size` value is mandatory, while the `initial_size` and `min_size` are optional and if not explicitly set are assumed to be `0`. See the `pgagroal_databases.conf` file documentation for more details.

Restart pgagroal

In order to apply changes to the prefill configuration, you need to restart **pgagroal**. You can do so by stopping it and then re-launch the daemon, as **pgagroal** operating system user:

```
pgagroal-cli shutdown
pgagroal -d
```

Check the prefill

You can check the prefill by running, as the **pgagroal** operating system user, the `status` command:

```
pgagroal-cli status
Status:              Running
Active connections:  0
```

```
Total connections:   1
Max connections:     100
```

where the `Total connections` is set by the *initial* connection specified in the limit file.

6 Remote administration

Enable remote management

On the pooler machine, you need to enable the remote management. In order to do so, add the `management` setting to the main `pgagroal.conf` configuration file. The value of setting is the number of a free TCP/IP port to which the remote management will connect to.

With your editor of choice, edit the `/etc/pgagroal/pgagroal.conf` file and add the `management` option likely the following:

```
management = 2347
```

under the `[pgagroal]` section, so that the configuration file looks like:

```
[pgagroal]
...
management = 2347
```

See the pgagroal configuration settings for more details.

Add remote admin user

Remote management is done via a specific admin user, that has to be created within the pooler vault. As the **pgagroal** operating system user, run the following command:

```
cd /etc/pgagroal
pgagroal-admin -f /etc/pgagroal/pgagroal_admins.conf -U admin -P admin1234
    add-user
```

The above will create the `admin` username with the `admin1234` password.

We strongly encourage you to choose non trivial usernames and passwords!

Restart pgagroal

In order to make the changes available, and therefore activate the remote management, you have to restart **pgagroal**, for example by issuing the following commands from the **pgagroal** operatng system user:

```
pgagroal-cli shutdown
pgagroal -d
```

Connect via remote administration interface

In order to connect remotely, you need to specify at least the `-h` and `-p` flags on the `pgagroal-cli` command line. Such flags will tell `pgagroal-cli` to connect to a remote host. You can also specify the username you want to connect with by specifying the `-U` flag. So, to get the status of the pool remotely, you can issue:

```
pgagroal-cli -h localhost -p 2347 -U admin status
```

and type the password `admin1234` when asked for it.

If you don't specify the `-U` flag on the command line, you will be asked for a username too.

Please note that the above example uses `localhost` as the remote host, but clearly you can specify any *real* remote host you want to manage.

7 Security model

Create frontend users

Frontend users are stored into the `pgagroal_frontend_users.conf` file, that can be managed via the `pgagroal-admin` command line tool. See the documentation on frontend users for more details.

As an example, consider the user `myuser` that has the `mypassword` password defined on the PostgreSQL side. It is possible to *remap* the user password on the **pgagroal** side, so that an application can connect to the **pgagroal** using a different password, like `application_password`. In turn, **pgagroal** will connect to PostgreSQL using the `mypassword` password. Therefore, the application doesn't not know the *real* password used to connect to PostgreSQL.

To achieve this, as **pgagroal** operating system run the following command:

```
pgagroal-admin -f /etc/pgagroal/pgagroal_frontend_users.conf -U myuser -P
   application_password user add
```

You will need a password mapping for each user defined in the `pgagroal_users.conf` configuration file.

Restart pgagroal

In order to apply changes, you need to restart **pgagroal** so do:

```
pgagroal-cli shutdown
pgagroal -d
```

Connect to PostgreSQL

You can now use the "application password" to access the PostgreSQL instance. As an example, run the following as any operatng system user:

```
psql -h localhost -p 2345 -U myuser mydb
```

using `application_password` as the password. As already explained, **pgagroal** will then use the `mypassword` password against PostgreSQL.

This **split security model** allows you to avoid sharing password between applications and PostgreSQL, letting the **pgagroal** to be the secret-keeper. This not only improves security, but also allows you to change the PostgreSQL password without having the application change its configuration.

8 Transport Level Security (TLS)

Creating Certificates

This tutorial will show you how to create self-signed certificate for the server, valid for 365 days, use the following OpenSSL command, replacing `dbhost.yourdomain.com` with the server's host name, here `localhost`:

```
openssl req -new -x509 -days 365 -nodes -text -out server.crt \
  -keyout server.key -subj "/CN=dbhost.yourdomain.com"
```

then do -

```
chmod og-rwx server.key
```

because the server will reject the file if its permissions are more liberal than this. For more details on how to create your server private key and certificate, refer to the OpenSSL documentation.

For the purpose of this tutorial we will assume the client certificate and key same as the server certificate and server key and therefore, these equations always holds -

- </path/to/client.crt> = </path/to/server.crt>
- </path/to/client.key> = </path/to/server.key>
- </path/to/server_root_ca.crt> = </path/to/server.crt>
- </path/to/client_root_ca.crt> = </path/to/server_root_ca.crt>

TLS in `pgagroal`

Modify the `pgagroal` configuration

It is now time to modify the pgagroal section of configuration file `/etc/pgagroal/pgagroal_vault.conf`, with your editor of choice by adding the following lines in the pgagroal section.

```
tls = on
tls_cert_file = </path/to/server.crt>
tls_key_file = </path/to/server.key>
```

Only Server Authentication

If you wish to do only server authentication the aforementioned configuration suffice.

Client Request

```
PGSSLMODE=verify-full PGSSLROOTCERT=</path/to/server_root_ca.crt> psql -h
   localhost -p 2345 -U <postgres_user> <postgres_database>
```

Full Client and Server Authentication

To enable the server to request the client certificates add the following configuration lines

```
tls = on
tls_cert_file = </path/to/server.crt>
tls_key_file = </path/to/server.key>
tls_ca_file = </path/to/client_root_ca.crt>
```

Client Request

```
PGSSLMODE=verify-full PGSSLCERT=</path/to/client.crt> PGSSLKEY=</path/to/
   client.key> PGSSLROOTCERT=</path/to/server_root_ca.crt> psql -h
   localhost -p 2345 -U <postgres_user> <postgres_database>
```

9 Database Aliases

9.1 Overview

Database aliases are configured in the `pgagroal_databases.conf` file and allow multiple alternative names for a single database entry. When a client connects using an alias, **pgagroal** transparently maps it to the real database name for backend connections.

9.2 Configuration

Basic Configuration (Without Aliases)

A standard database configuration without aliases looks like this:

```
# Database configuration without aliases
mydb    myuser    10    5    2
```

Configuration with Aliases

To add aliases to a database, use the following syntax:

```
# Database configuration with aliases
mydb=alias1,alias2,alias3    myuser    10    5    2
```

In this example: - `mydb` is the real database name - `alias1`, `alias2`, and `alias3` are alternative names clients can use - Clients can connect using any of these four names: `mydb`, `alias1`, `alias2`, or `alias3`

Important Configuration Rules

Maximum Aliases Per Database

Each database can have a maximum of **8 aliases**. If you configure more than 8 aliases, only the first 8 will be accepted and the rest will be **truncated with a warning message logged**.

```
# Only alias1 through alias8 will be used
# alias9 and alias10 will be truncated with a warning
mydb=alias1,alias2,alias3,alias4,alias5,alias6,alias7,alias8,alias9,
    alias10    myuser    10    5    2
```

When this occurs, you'll see a warning message in the logs similar to:

```
WARN: Database 'mydb' has 10 aliases, but only the first 8 will be used (
    max limit: 8)
```

Important: Clients attempting to connect using truncated aliases (alias9, alias10 in the example above) will receive connection errors since those aliases are not functional.

9.3 Validation rules

pgagroal enforces strict validation rules for database aliases to ensure configuration integrity:

Global Uniqueness

All aliases must be globally unique across the entire configuration:

```
# WRONG - 'shared_alias' appears twice
db1=shared_alias,alias1     user1    10    5    2
db2=shared_alias,alias2     user2    10    5    2

# CORRECT - All aliases are unique
db1=unique_alias1,alias1    user1    10    5    2
db2=unique_alias2,alias2    user2    10    5    2
```

No Conflicts with Database Names

Aliases cannot have the same name as any real database name in the configuration:

```
# WRONG - 'db2' is used as both database name and alias
db1=db2,alias1              user1    10    5    2
db2=alias2,alias3           user2    10    5    2

# CORRECT - No conflicts between database names and aliases
db1=unique_alias,alias1     user1    10    5    2
db2=alias2,alias3           user2    10    5    2
```

Special Database Restrictions

The special database `all` cannot have aliases:

```
# WRONG - 'all' database cannot have aliases
all=alias1,alias2    myuser    10    5    2

# CORRECT - 'all' database without aliases
all                  myuser    10    5    2
```

Empty Aliases Behavior

Empty aliases (consecutive commas or trailing commas) are automatically ignored:

```
# These configurations are  valid - empty aliases are skipped
mydb=alias1,,alias2       myuser    10    5    2  # Empty alias between
    alias1 and alias2
mydb=alias1, ,alias2      myuser    10    5    2  # Empty alias between
    alias1 and alias2 with space
mydb=,alias1,alias2       myuser    10    5    2  # Leading comma before
    alias1

# All result in the same aliases: alias1, alias2 (where applicable)
```

Note: While empty aliases are ignored, trailing commas may cause parsing issues in some edge cases. For best compatibility, avoid trailing commas:

```
# RECOMMENDED - No trailing comma
mydb=alias1,alias2,alias3    myuser    10    5    2

# NOT WORKS NOT RECOMMENDED - Trailing comma
mydb=alias1,alias2,alias3,   myuser    10    5    2
```

Invalid Configurations

Empty alias section: A database name followed by = with no aliases is invalid:

```
# INCORRECT - '=' with no aliases
mydb=                    myuser           10    5    2
mydb =                   myuser           10    5    2  # Spaces after
   '=' but no aliases

# CORRECT - Either use aliases or don't use '=' at all
mydb=alias1,alias2       myuser    10    5    2  # With aliases
mydb                     myuser    10    5    2  # Without aliases
```

Trailing commas: Any commas at the end of the alias list will cause parsing failures:

```
# WRONG - Trailing comma causes parsing issues
mydb=alias1,alias2,alias3,   myuser    10    5    2

# CORRECT - No trailing comma
mydb=alias1,alias2,alias3    myuser    10    5    2
```

9.4 Spaces

Allowed Spaces

Spaces are permitted in specific locations within alias configurations:

```
# CORRECT - Spaces around '=' and commas are allowed
mydb = alias1 , alias2 , alias3    myuser    10    5    2
mydb =alias1, alias2,alias3        myuser    10    5    2
mydb= alias1,alias2 ,alias3        myuser    10    5    2

# All result in the same aliases: alias1, alias2, alias3
```

Allowed space locations

- Between database name and =: `mydb =alias1`
- Between = and first alias: `mydb= alias1`
- Before commas: `alias1 ,alias2`

- After commas: `alias1, alias2`
- Multiple spaces in any of these locations: `mydb = alias1 , alias2`

Invalid Spaces

Spaces **within individual alias names** are not allowed and will cause parsing errors:

```
# WRONG - Spaces within alias names
mydb=my alias,another alias       myuser    10    5    2
mydb=alias 1,alias 2              myuser    10    5    2

# CORRECT - No spaces within alias names
mydb=my_alias,another_alias       myuser    10    5    2
mydb=alias1,alias2                myuser    10    5    2
```

Best Practices for Spaces

1. **Use underscores or hyphens** instead of spaces in alias names:

   ```
   # RECOMMENDED
   mydb=user_portal,admin_panel,api_gateway    myuser    10    5    2
   mydb=user-portal,admin-panel,api-gateway    myuser    10    5    2
   ```

2. **Consistent formatting** for readability:

   ```
   # Clean format (recommended)
   mydb=alias1,alias2,alias3        myuser    10    5    2

   # Spaced format (also valid)
   mydb = alias1, alias2, alias3    myuser    10    5    2
   ```

Summary

- **Allowed**: Spaces between database name, =, commas, and aliases
- **Not allowed**: Spaces within individual alias names
- **Best practice**: Use _ or − for multi-word aliases

9.5 Quoted aliases

pgagroal treats quoted and unquoted aliases as distinct entities. The following are considered different aliases:

- `alias`
- `'alias'`
- `"alias"`

Shell Quote Handling

When using command-line tools like `psql`, shells (Bash, Zsh) automatically strip quotes before passing arguments to programs. This means these commands are equivalent:

```
psql -h localhost -p 2345 -U myuser alias
psql -h localhost -p 2345 -U myuser 'alias'
psql -h localhost -p 2345 -U myuser "alias"
```

All result in `psql` receiving the argument `alias` (without quotes).

Using Quoted Aliases

To connect to a quoted alias, you must escape the quotes:

```
# Connect to alias named "alias" (with double quotes)
psql -h localhost -p 2345 -U myuser \"alias\"

# Connect to alias named 'alias' (with single quotes)
psql -h localhost -p 2345 -U myuser \'alias\'
```

Recommendation: Avoid using quotes in alias names to prevent confusion and simplify client connections.

9.6 Managing aliases

Viewing Configured Aliases

Use the `pgagroal-cli` command to view all configured databases and their aliases:

```
# Display all databases with their aliases
pgagroal-cli conf alias

# Get specific database alias information
pgagroal-cli conf get limit.mydb.aliases
pgagroal-cli conf get limit.mydb.number_of_aliases
```

Runtime Management

Database aliases are part of the limit configuration and can be managed through configuration reloads:

```
# Reload configuration to apply alias changes
pgagroal-cli conf reload
```

9.7 Connection pooling

When using aliases, **pgagroal** ensures efficient connection reuse:

- All connections are established using the real database name
- Connections can be reused regardless of whether clients connect via the real name or any alias
- Pool statistics and monitoring use the real database name
- Authentication and authorization use the real database name

9.8 Example configuration

Here's a complete example demonstrating various alias configurations:

```
# /etc/pgagroal/pgagroal_databases.conf

# Production database with environment-specific aliases
prod_db=production,live,main                myuser    20    10    5

# Development database with multiple aliases for teams
dev_db=development,staging,test,beta        devuser   10    5     2

# Legacy database with old naming convention
new_app_db=legacy_app,old_system            appuser   15    8     3

# Database without aliases
analytics                                   analyst   5     2     1
```

9.9 Best practices

1. **Use descriptive aliases** that make sense to your application teams
2. **Avoid spaces** in alias definitions
3. **Keep aliases simple** - avoid special characters and quotes
4. **Document your alias strategy** for team members
5. **Plan for the 8-alias limit** when designing your naming scheme
6. **Use aliases for gradual migrations** when renaming databases
7. **Test alias connectivity** after configuration changes
8. **Avoid trailing commas** in alias definitions to prevent parsing edge cases

9.10 Troubleshooting

Alias Not Working

- Check for spaces in the alias definition
- Verify the alias doesn't conflict with existing database names
- Ensure the alias is within the 8-alias limit
- Confirm configuration has been reloaded

Connection Failures

- Verify the alias exists in the configuration
- Check HBA rules allow connections for the target database
- Ensure proper escaping when using quoted aliases

10 Vault

10.1 Enable rotation of passwords

In the main configuration file of **pgagroal** add the following configuration for rotating frontend passwords and make sure management port is enabled in the **pgagroal** section of the configuration file.

```
management = 2347
rotate_frontend_password_timeout = 60
rotate_frontend_password_length = 12
```

10.2 Configuration

In order to run `pgagroal-vault`, you need to configure the vault `pgagroal_vault.conf` configuration file, that will tell the vault how to work, which address to listen, address of management service in **pgagroal** so on, and then `pgagroal_vault_users.conf` that will instrument the vault about the admin username and password of the remote management.

pgagroal-vault.conf

It is now time to create the main `/etc/pgagroal/pgagroal_vault.conf` configration file, with your editor of choice or using `cat` from the command line, create the following content:

```
cd /etc/pgagroal
cat > pgagroal_vault.conf
[pgagroal-vault]
host = localhost
port = 2500

metrics = 2501

ev_backend = auto

log_type = console
log_level = info
log_path = /tmp/pgagroal-vault.log

[main]
host = localhost
port = 2347
user = admin
```

and press `Ctrl-d` (if running `cat`) to save the file.

Add users file

As the **pgagroal** operating system user, run the following command:

```
pgagroal-admin -f /etc/pgagroal/pgagroal_vault_users.conf -U admin -P
   admin1234 user add
```

The above will create the `admin` username with the `admin1234` password. Alternately, `/etc/pgagroal/pgagroal_admins.conf` can be provided for vault users information.

See the documentation about `pgagroal_vault.conf` for more details.

10.3 Start the vault

It is now time to start `pgagroal-vault`, so as the **pgagroal** operating system user run:

```
pgagroal-vault -d
```

If both `pgagroal` and `pgagroal-vault` are on the same operating system they can use the same `pgagroal_admins.conf` file.

This command initializes an HTTP server on localhost port 2500, which is primed to exclusively handle GET requests from clients.

10.4 Connect to the vault

Since we have deployed an HTTP server we can simply use `curl` to send GET requests

Correct requests

If the requested URL is of form `http://<hostname>:<port>/users/<frontend_user>` such that `<frontend_user>` exists, the server will return a header response with a 200 status code and the frontend password corresponding to the `<frontend_user>` in the response body.

Example

```
curl -i http://localhost:2500/users/myuser
```

Output

```
HTTP/1.1 200 OK
Content-Type: text/plain


password
```

Incorrect requests

All the POST requests will be ignored and the server will send a `HTTP 404 ERROR` as a response.

Any URL other than the format: `http://<hostname>:<port>/users/*` will result in `HTTP 404 ERROR`.

Example

```
curl -i http://localhost:2500/user
```

Output

```
HTTP/1.1 404 Not Found
```

A URL of form `http://<hostname>:<port>/users/<frontend_user>` such that `<frontend_user>` does not exist will also give `HTTP 404 ERROR`.

Example

```
curl -i http://localhost:2500/users/randomuser
```

Output

```
HTTP/1.1 404 Not Found
```

10.5 Monitor the vault

Status endpoint

The vault provides a status endpoint for health monitoring and operational visibility:

```
curl http://localhost:2500/status
```

This endpoint: * Requires no authentication * Returns JSON with vault status information * Tests pgagroal connection in real-time * Shows configuration details * Available over both HTTP and HTTPS

Example response:

```
{
  "status": "ok",
  "timestamp": "2025-01-08T10:30:45Z",
  "vault": {
    "version": "2.0.0",
    "pid": 12345
  },
  "configuration": {
    "host": "localhost",
    "port": 2500,
    "tls_enabled": false,
    "metrics_port": 2501,
    "metrics_tls_enabled": false
  },
```

```
  "pgagroal_connection": {
    "status": "connected",
    "host": "localhost",
    "port": 2347
  }
}
```

10.6 Transport Level Security (TLS)

Enable TLS

It is now time to modify the `[pgagroal-vault]` section of configuration file `/etc/pgagroal/pgagroal_vault.conf` with your editor of choice by adding the following lines in the pgagroal-vault section.

```
tls = on
tls_cert_file = </path/to/server.crt>
tls_key_file = </path/to/server.key>
```

This will add TLS support to the server alongside the standard `http` endpoint, allowing clients to make requests to either the `https` or `http` endpoint.

Only Server Authentication

If you wish to do only server authentication the aforementioned configuration suffice.

Client Request

```
curl --cacert </path/to/server_root_ca.crt> -i https://localhost:2500/
   users/<frontend_user>
```

Full Client and Server Authentication

To enable the server to request the client certificates add the following configuration lines

```
tls = on
tls_cert_file = </path/to/server.crt>
tls_key_file = </path/to/server.key>
tls_ca_file = </path/to/client_root_ca.crt>
```

Client Request

```
curl --cert </path/to/client.crt> --key </path/to/client.key> --cacert </
   path/to/server_root_ca.crt> -i https://localhost:2500/users/<
   frontend_user>
```

Certificate Authentication Modes

When `tls_ca_file` is configured, the vault supports two certificate authentication modes controlled by the `tls_cert_auth_mode` setting:

verify-ca (default): The vault verifies that the client certificate is signed by a trusted CA. This is the default mode and provides a balance between security and ease of use.

```
tls = on
tls_cert_file = </path/to/server.crt>
tls_key_file = </path/to/server.key>
tls_ca_file = </path/to/client_root_ca.crt>
tls_cert_auth_mode = verify-ca
```

verify-full: In addition to CA verification, the vault also verifies that the certificate's Subject Alternative Name (SAN) or Common Name (CN) matches the username being accessed. This provides the highest level of security by ensuring certificate ownership.

```
tls = on
tls_cert_file = </path/to/server.crt>
tls_key_file = </path/to/server.key>
tls_ca_file = </path/to/client_root_ca.crt>
tls_cert_auth_mode = verify-full
```

With `verify-full` mode enabled, if a client attempts to access `/users/alice`, the client certificate must contain "alice" in the SAN extension or in the CN field (if no SAN exists).

Certificate Identity Priority

When using `verify-full` mode, the vault extracts the username from the client certificate using this priority:

1. **Subject Alternative Name (SAN)** - Checked first. The vault examines DNS, Email, then URI types in order and uses the first valid value found (non-empty, no null bytes).
2. **Common Name (CN)** - Only checked if no SAN extension exists, or no valid SAN value is found.

Important: CN is **completely ignored** if any valid SAN value exists in the certificate, even if the SAN value doesn't match the requested username.

How identity extraction works: 1. Check SAN extension: If present, examine DNS type first, then Email, then URI 2. Use the first valid SAN value found and stop (remaining SANs and CN are ignored) 3. If no SAN extension exists or no valid SAN values, check CN as fallback 4. The extracted identity is then compared (case-sensitive) against the requested username

Recommendation: For simplicity, use `CN=username` with no SAN entries, or a SAN entry with a single DNS value.

Example certificate with CN only:

```
openssl req -new -key user.key -out user.csr -subj "/CN=alice"
```

Examples:

```
Certificate: CN=alice (no SAN extension)
Access: /users/alice => Success (CN used)

Certificate: SAN DNS=alice
Access: /users/alice => Success (SAN DNS used, CN ignored)

Certificate: SAN DNS=alice, CN=bob
Access: /users/alice => Success (SAN used, CN completely ignored)
Access: /users/bob => Fails (CN ignored when SAN present)
```

11 Prometheus

11.1 pgagroal

Once **pgagroal** is running you can access the metrics with a browser at the pooler address, specifying the `metrics` port number and routing to the `/metrics` page. For example, point your web browser at:

```
http://localhost:2346/metrics
```

It is also possible to get an explaination of what is the meaning of each metric by pointing your web browser at:

```
http://localhost:2346/
```

11.1.1 Metrics

pgagroal_state

The state of pgagroal

pgagroal_pipeline_mode

The mode of pipeline

pgagroal_server_error

The number of errors for servers

pgagroal_logging_info

The number of INFO logging statements

pgagroal_logging_warn

The number of WARN logging statements

pgagroal_logging_error

The number of ERROR logging statements

pgagroal_logging_fatal

The number of FATAL logging statements

pgagroal_failed_servers

The number of failed servers

pgagroal_wait_time

The waiting time of clients

pgagroal_query_count

The number of queries

pgagroal_connection_query_count

The number of queries per connection

pgagroal_tx_count

The number of transactions

pgagroal_active_connections

The number of active connections

pgagroal_total_connections

The total number of connections

pgagroal_max_connections

The maximum number of connections

pgagroal_connection

The connection information

pgagroal_session_time_seconds

The session times

pgagroal_connection_error

Number of connection errors

pgagroal_connection_kill

Number of connection kills

pgagroal_connection_remove

Number of connection removes

pgagroal_connection_timeout

Number of connection time outs

pgagroal_connection_return

Number of connection returns

pgagroal_connection_invalid

Number of connection invalids

pgagroal_connection_get

Number of connection gets

pgagroal_connection_idletimeout

Number of connection idle timeouts

pgagroal_connection_max_connection_age

Number of connection max age timeouts

pgagroal_connection_flush

Number of connection flushes

pgagroal_connection_success

Number of connection successes

pgagroal_auth_user_success

Number of successful user authentications

pgagroal_auth_user_bad_password

Number of bad passwords during user authentication

pgagroal_auth_user_error

Number of errors during user authentication

pgagroal_client_wait

Number of waiting clients

pgagroal_client_active

Number of active clients

pgagroal_network_sent

Bytes sent by clients

pgagroal_network_received

Bytes received from servers

pgagroal_client_sockets

Number of sockets the client used

pgagroal_self_sockets

Number of sockets used by pgagroal itself

pgagroal_connection_awaiting

Number of connection on-hold (awaiting)

pgagroal_os_info

Operating system version information

pgagroal_certificates_total

Total number of TLS certificates configured

pgagroal_certificates_accessible

Number of accessible TLS certificates

pgagroal_certificates_valid

Number of valid TLS certificates

pgagroal_certificates_expired

Number of expired TLS certificates

pgagroal_certificates_expiring_soon

Number of TLS certificates expiring within 30 days

pgagroal_certificates_inaccessible

Number of inaccessible TLS certificate files

pgagroal_certificates_parse_errors

Number of TLS certificates with parsing errors

pgagroal_tls_certificate_status

Certificate status (1=valid, 0=invalid/inaccessible)

pgagroal_tls_certificate_expiration_seconds

TLS certificate expiration time

pgagroal_tls_certificate_key_size_bits

TLS certificate key size in bits

pgagroal_tls_certificate_is_ca

Whether certificate is a CA certificate

pgagroal_tls_certificate_key_type

TLS certificate key type

pgagroal_tls_certificate_signature_algorithm

TLS certificate signature algorithm

pgagroal_tls_certificate_info

TLS certificate metadata

11.2 pgagroal-vault

Once **pgagroal-vault** is running you can access the metrics with a browser at the pooler address, specifying the `metrics` port number and routing to the `/metrics` page. For example, point your web browser at:

```
http://localhost:2501/metrics
```

It is also possible to get an explaination of what is the meaning of each metric by pointing your web browser at:

```
http://localhost:2501/
```

12 Docker

You can run **pgagroal** using Docker instead of compiling it manually.

Prerequisites

- **Docker** or **Podman** must be installed on the server where PostgreSQL is running.
- Ensure PostgreSQL is configured to allow external connections.

Update the configuration file if needed:

```
[pgagroal]
host = *
port = 2345
metrics = 2346
log_type = file
log_level = debug
log_path = /tmp/pgagroal.log
ev_backend = auto

max_connections = 100
idle_timeout = 600
validation = off
unix_socket_dir = /tmp/

[primary]
host = host.docker.internal
port = 5432
```

pgagroal_hba.conf

```
#
# TYPE  DATABASE USER  ADDRESS  METHOD
#
host    all      all   all      all
```

Step 1: Enable External PostgreSQL Access

Modify the local PostgreSQL server's `postgresql.conf` file to allow connections from outside:

```
listen_addresses = '*'
```

Update `pg_hba.conf` to allow remote connections:

```
host    all    all    0.0.0.0/0    scram-sha-256
```

Follow GETTING STARTED for further server setup

Then, restart PostgreSQL for the changes to take effect:

```
sudo systemctl restart postgresql
```

Step 2: Clone the Repository

```
git clone https://github.com/pgagroal/pgagroal.git
cd pgagroal
```

Step 3: Build the Docker Image

There are two Dockerfiles available: 1. **Alpine-based image**

Using Docker

```
docker build -t pgagroal:latest -f ./contrib/docker/Dockerfile.alpine .
```

Using Podman

```
podman build -t pgagroal:latest -f ./contrib/docker/Dockerfile.alpine .
```

2. **Rocky Linux 9-based image**

Using Docker

```
docker build -t pgagroal:latest -f ./contrib/docker/Dockerfile.rocky9 .
```

Using Podman

```
podman build -t pgagroal:latest -f ./contrib/docker/Dockerfile.rocky9 .
```

Step 4: Run pgagroal as a Docker Container

Once the image is built, run the container using:

- **Using Docker**

```
docker run -d --name pgagroal \
  -p 2345:2345 \
  -p 2346:2346 \
  --add-host=host.docker.internal:host-gateway \
  pgagroal:latest
```

- **Using Podman**

```
podman run -d --name pgagroal \
  -p 2345:2345 \
  -p 2346:2346 \
  --add-host=host.docker.internal:host-gateway \
  pgagroal:latest
```

Step 5: Verify the Container

Check if the container is running:

- **Using Docker**

```
docker ps | grep pgagroal -->
```

- **Using Podman**

```
podman ps | grep pgagroal
```

Check logs for any errors:

- **Using Docker**

```
docker logs pgagroal
```

- **Using Podman**

```
podman logs pgagroal
```

You can also inspect the exposed metrics at:

```
http://localhost:5001/metrics
```

You can stop the container using

- **Using Docker**

```
docker stop ppgagroal
```

- **Using Podman**

```
podman stop ppgagroal
```

We will assume that we have a user called `test` with the password `test` in our PostgreSQL instance. See their documentation on how to setup PostgreSQL, add a user and add a database.

We will connect to **pgagroal** using the psql application.

```
psql -h localhost -p 2345 -U test test
```

You can exec into the container and run the cli commands as

```
docker exec -it pgagroal /bin/bash
#or using podman
podman exec -it pgagroal /bin/bash

cd /etc/pgagroal
/usr/local/bin/pgagroal-cli -c pgagroal.conf shutdown
```

See this for more cli commands.

You can access the three binaries at `/usr/local/bin`

13 Command Line Tools

This chapter provides comprehensive reference for pgagroal's command-line utilities.

13.1 pgagroal-cli

`pgagroal-cli` is a command line interface to interact with **pgagroal**. The executable accepts a set of options, as well as a command to execute. If no command is provided, the program will show the help screen.

The `pgagroal-cli` utility has the following synopsis:

```
pgagroal-cli [ OPTIONS ] [ COMMAND ]
```

13.1.1 Options

Available options are the following ones:

```
-c, --config CONFIG_FILE Set the path to the pgagroal.conf file
-h, --host HOST          Set the host name
-p, --port PORT          Set the port number
-U, --user USERNAME      Set the user name
-P, --password PASSWORD  Set the password
-L, --logfile FILE       Set the log file
-F, --format text|json   Set the output format
-v, --verbose            Output text string of result
-V, --version            Display version information
-?, --help               Display help
```

Options can be specified either in short or long form, in any position of the command line.

By default the command output, if any, is reported as text. It is possible to specify JSON as the output format, and this is the suggested format if there is the need to automatically parse the command output, since the text format could be subject to changes in future releases.

13.1.2 Commands

13.1.2.1 flush The `flush` command performs a connection flushing. It accepts a *mode* to operate the actual flushing: - `gracefully` (the default if not specified), flush connections when possible; - `idle` to flush only connections in state *idle*; - `all` to flush all the connections (**use with caution!**).

The command accepts a database name, that if provided, restricts the scope of `flush` only to connections related to such database. If no database is provided, the `flush` command is operated against all databases.

Command:

```
pgagroal-cli flush [gracefully|idle|all] [*|<database>]
```

Examples:

```
pgagroal-cli flush           # pgagroal-cli flush gracefully '*'
pgagroal-cli flush idle      # pgagroal-cli flush idle '*'
pgagroal-cli flush all       # pgagroal-cli flush all '*'
pgagroal-cli flush pgbench   # pgagroal-cli flush gracefully pgbench
```

13.1.2.2 ping The `ping` command checks if **pgagroal** is running. In case of success, the command does not print anything on the standard output unless the `--verbose` flag is used.

Command:

```
pgagroal-cli ping
```

Example:

```
pgagroal-cli ping --verbose  # pgagroal-cli: Success (0)
pgagroal-cli ping            # $? = 0
```

13.1.2.3 enable Enables a database (or all databases).

Command:

```
pgagroal-cli enable [<database>|*]
```

Example:

```
pgagroal-cli enable
```

13.1.2.4 disable Disables a database (or all databases).

Command:

```
pgagroal-cli disable [<database>|*]
```

Example:

```
pgagroal-cli disable
```

13.1.2.5 status The `status` command reports the current status of the **pgagroal** pooler. Without any subcommand, `status` reports back a short set of information about the pooler.

Command:

```
pgagroal-cli status [details]
```

With the `details` subcommand, a more verbose output is printed with a detail about every connection.

Example:

```
pgagroal-cli status details
```

13.1.2.6 switch-to Switch to another primary server.

Command:

```
pgagroal-cli switch-to <server>
```

Example:

```
pgagroal-cli switch-to replica
```

13.1.2.7 shutdown The `shutdown` command is used to stop the connection pooler. It supports the following operating modes: - `gracefully` (the default) closes the pooler as soon as no active connections are running; - `immediate` forces an immediate stop.

If the `gracefully` mode is requested, chances are the system will take some time to perform the effective shutdown, and therefore it is possible to abort the request issuing another `shutdown` command with the mode `cancel`.

Command:

```
pgagroal-cli shutdown [gracefully|immediate|cancel]
```

Examples:

```
pgagroal-cli shutdown   # pgagroal-cli shutdown gracefully
...
pgagroal-cli shutdown cancel  # stops the above command
```

13.1.2.8 conf Manages the configuration of the running instance. This command requires one subcommand, that can be: - `reload` issue a reload of the configuration, applying at runtime any

changes from the configuration files; - `get` provides a configuration parameter value; - `set` modifies a configuration parameter at runtime; - `ls` prints where the configuration files are located; - `alias` shows all databases with their aliases and usernames.

Command:

```
pgagroal-cli conf <what>
```

Examples:

```
pgagroal-cli conf reload
pgagroal-cli conf get max_connections
pgagroal-cli conf set max_connections 25
```

13.1.2.8.1 conf get The `conf get` command retrieves the current value of a configuration parameter from the running instance.

Syntax:

```
pgagroal-cli conf get <parameter_name> [--verbose] [--format json]
```

Parameter Name Format: The parameter name can be specified in several forms using dot notation: - `key`: Refers to a global configuration parameter (e.g., `log_level`) - `section.key`: Refers to a parameter within a named section (e.g., `server.venkman.port`) - `section.context.key`: Refers to a parameter within a context, such as a limit or HBA entry (e.g., `limit.pgbench.max_size`, `hba.myuser.method`)

Supported Namespaces: - `pgagroal` (optional): Main configuration namespace, e.g., `pgagroal.log_level` or simply `log_level` - `server`: Refers to a specific server, e.g., `server.venkman.port` - `limit`: Refers to a specific limit entry, e.g., `limit.pgbench.max_size` - `hba`: Refers to a specific HBA entry, e.g., `hba.myuser.method`

Context Rules: - For `limit`, the context is the database name as found in `pgagroal_databases.conf`. - For `hba`, the context is the username as found in `pgagroal_hba.conf`. - For `server`, the context is the server name as found in `pgagroal.conf`. - For `pgagroal`, the context must be empty.

> **Important:**
> If there are multiple entries with the same database name in the `limit` section or the same username in the `hba` section, **the last matching entry in the configuration file is used** for `conf get`.

Examples:

```
pgagroal-cli conf get pipeline
performance

pgagroal-cli conf get limit.pgbench.max_size
2

pgagroal-cli conf get server.venkman.primary
off

pgagroal-cli conf get hba.myuser.method
scram-sha-256
```

Verbose Output: If the `--verbose` option is specified, the output is more descriptive:

```
pgagroal-cli conf get max_connections --verbose
max_connections = 4
Success (0)
```

Full Configuration Output:

If you run the `conf get` command without specifying any parameter name, the command will return the complete configuration, including all main settings, servers, limits, and HBA entries.

```
pgagroal-cli conf get
```

You can also retrieve a specific section by specifying only the section name:

- To get all limit entries:

  ```
  pgagroal-cli conf get limit
  ```

- To get all HBA entries:

  ```
  pgagroal-cli conf get hba
  ```

- To get all Server entries:

  ```
  pgagroal-cli conf get server
  ```

> **Important:**
> When viewing the full configuration or a section (such as `limit` or `hba`), **only a single entry will be present for each database name in `limit` and for each username in `hba`**.
> If your configuration file contains multiple entries with the same database name (in `limit`) or the same username (in `hba`), **only the last entry in the configuration file will be shown**.

13.1.2.8.2 conf set The `conf set` command allows you to change a configuration parameter at run-time, if possible.

Syntax:

```
pgagroal-cli conf set <parameter_name> <parameter_value>
```

Examples:

```
pgagroal-cli conf set log_level debug
pgagroal-cli conf set server.venkman.port 6432
pgagroal-cli conf set limit.pgbench.max_size 2
```

The syntax for setting parameters is the same as for the `conf get` command. Parameters are organized into namespaces: - `pgagroal` (optional): the main configuration namespace, e.g., `pgagroal.log_level` or simply `log_level` - `server`: refers to a specific server, e.g., `server.venkman.port` - `limit`: refers to a specific limit entry, e.g., `limit.pgbench.max_size` - `hba`: refers to a specific HBA entry, e.g., `hba.myuser.method`

Output:

When executed, the `conf set` command returns a detailed result describing the outcome of the operation.

Case 1: Successful Application

If the parameter is set and applied to the running instance:

```
Configuration change applied successfully
   Parameter: log_level
   Old value: info
   New value: debug
   Status: Active (applied to running instance)
```

Case 2: Restart Required

If the parameter change requires a full service restart to take effect:

```
Configuration change requires manual restart
   Parameter: max_connections
   Current value: 40 (unchanged in running instance)
   Requested value: 100
   Status: Requires full service restart
```

Case 3: Invalid Key or Syntax

If the parameter name is invalid or the syntax is incorrect:

```
Configuration change failed
   Invalid key format: 'max_connectionz'
   Valid formats: 'key', 'section.key', or 'section.context.key'
```

> **Warning:**
> When changing critical parameters such as the **main port**, **metrics port**, **management port**, or **unix_socket_dir**, you must choose values carefully.
>
> If you set a port or socket to a value that is already in use or unavailable, the reload will fail. In this case, the CLI may still report "success" because the configuration was accepted, but the server will **not** be listening on the new port or socket.
>
> **How to recover:**
> If you accidentally set a port or socket to an unavailable value, simply use the `conf set` command again to set it to a valid, available value.
>
> **Multiple Entries in `limit` or `hba`:**
> If your configuration contains multiple entries with the **same database name** in the `limit` section or the **same username** in the `hba` section, **the `conf set` command will apply the change to the first matching entry** (topmost in the configuration file).

13.1.2.8.3 conf ls The command `conf ls` provides information about the location of the configuration files.

Example:

```
pgagroal-cli conf ls

Main Configuration file:   /etc/pgagroal/pgagroal.conf
HBA file:                  /etc/pgagroal/pgagroal_hba.conf
Limit file:                /etc/pgagroal/pgagroal_databases.conf
Frontend users file:       /etc/pgagroal/pgagroal_frontend_users.conf
Admins file:               /etc/pgagroal/pgagroal_admins.conf
Superuser file:
Users file:                /etc/pgagroal/pgagroal_users.conf
```

13.1.2.8.4 conf alias The command `conf alias` shows all the databases in `pgagroal_databases.conf` along with their aliases and usernames.

```
pgagroal-cli conf alias

# DATABASE=ALIASES                          USER           MAX   INIT
   MIN
#------------------------------------------------------------------------

production_db=prod,main,primary             myuser          10    5    2
```

13.1.2.9 clear Resets different parts of the pooler. It accepts an operational mode: - `prometheus` resets the metrics provided without altering the pooler status; - `server` resets the specified server status.

Command:

```
pgagroal-cli clear [prometheus|server <server>]
```

Examples:

```
pgagroal-cli clear spengler            # pgagroal-cli clear server
   spengler
pgagroal-cli clear prometheus
```

13.1.3 Shell Completions

pgagroal provides shell completion support for both `pgagroal-cli` and `pgagroal-admin` commands in bash and zsh shells.

13.1.3.1 Installation The shell completion scripts are located in the `contrib/shell_comp/` directory: - `pgagroal_comp.bash` - Bash completion script - `pgagroal_comp.zsh` - Zsh completion script

13.1.3.1.1 Bash Completion For current session only:

```
source /path/to/pgagroal/contrib/shell_comp/pgagroal_comp.bash
```

For permanent installation, add to your `~/.bashrc`:

```
echo "source /path/to/pgagroal/contrib/shell_comp/pgagroal_comp.bash" >>
   ~/.bashrc
```

If pgagroal is installed via package manager:

```
source /usr/share/doc/pgagroal/shell_comp/pgagroal_comp.bash
```

13.1.3.1.2 Zsh Completion For current session only:

```
source /path/to/pgagroal/contrib/shell_comp/pgagroal_comp.zsh
```

For permanent installation, add to your `~/.zshrc`:

```
echo "source /path/to/pgagroal/contrib/shell_comp/pgagroal_comp.zsh" >>
   ~/.zshrc
```

13.1.3.2 Usage

Once enabled, you can use tab completion with pgagroal commands:

pgagroal-cli commands:

```
pgagroal-cli <TAB>
```

Shows: `flush ping enable disable shutdown status switch-to conf clear`

pgagroal-cli subcommands:

```
pgagroal-cli flush <TAB>
```

Shows: `gracefully idle all`

pgagroal-admin commands:

```
pgagroal-admin <TAB>
```

Shows available admin commands

The completion scripts provide intelligent suggestions for: - Available commands and subcommands - Command options and flags - Database names (where applicable)

13.1.4 JSON Output Format

It is possible to obtain the output of a command in a JSON format by specifying the `-F` (`--format`) option on the command line. Supported output formats are: - `text` (the default) - `json`

As an example, the following are invocations of commands with different output formats:

```
pgagroal-cli status     # defaults to text output format

pgagroal-cli status --format text  # same as above
pgagroal-cli status -F text        # same as above

pgagroal-cli status --format json  # outputs as JSON text
pgagroal-cli status -F json        # same as above
```

Whenever a command produces output, the latter can be obtained in a JSON format. Every command output consists of an object that contains two other objects: - a `command` object, with all the details about the command and its output; - an `application` object, with all the details about the executable that launched the command (e.g., `pgagroal-cli`).

13.1.4.1 The `application` object

The `application` object is made by the following attributes: - `name` a string representing the name of the executable that launched the command; - `version` a string representing the version of the executable; - `major`, `minor`, `patch` are integers representing every single part of the version of the application.

As an example, when `pgagroal-cli` launches a command, the output includes an `application` object like the following:

```
"application": {
    "name": "pgagroal-cli",
    "major": 1,
    "minor": 6,
    "patch": 0,
    "version": "1.6.0"
}
```

13.1.4.2 The `command` object

The `command` object represents the launched command and contains also the answer from **pgagroal**. The object is made by the following attributes: - `name` a string representing the command launched (e.g., `status`); - `status` a string that contains either “OK” or an error string if the command failed; - `error` an integer value used as a flag to indicate if the command was in error or not, where `0` means success and `1` means error; - `exit-status` an integer that contains zero if the command run successfully, another value depending on the specific command in case of failure; - `output` an object that contains the details of the executed command.

The `output` object is *the variable part* in the JSON command output, that means its effective content depends on the launched command.

Whenever the command output includes an array of stuff, for example a connection list, such array is wrapped into a `list` JSON array with a sibling named `count` that contains the integer size of the array (number of elements).

13.1.4.3 JSON Examples

The following are a few examples of commands that provide output in JSON:

```
pgagroal-cli ping --format json
{
    "command": {
        "name": "ping",
        "status": "OK",
        "error": 0,
        "exit-status": 0,
        "output": {
            "status": 1,
            "message": "running"
```

```
        }
    },
    "application": {
        "name": "pgagroal-cli",
        "major": 1,
        "minor": 6,
        "patch": 0,
        "version": "1.6.0"
    }
}
```

```
pgagroal-cli status --format json
{
    "command": {
        "name": "status",
        "status": "OK",
        "error": 0,
        "exit-status": 0,
        "output": {
            "status": {
                "message": "Running",
                "status": 1
            },
            "connections": {
                "active": 0,
                "total": 2,
                "max": 15
            },
            "databases": {
                "disabled": {
                    "count": 0,
                    "state": "disabled",
                    "list": []
                }
            }
        }
    },
    "application": {
        "name": "pgagroal-cli",
        "major": 1,
        "minor": 6,
        "patch": 0,
        "version": "1.6.0"
    }
}
```

As an example, the following is the output of a faulty `conf set` command (note the `status`, `error` and `exit-status` values):

```
pgagroal-cli conf set max_connections 1000 --format json
```

```
{
    "command": {
        "name": "conf set",
        "status": "Current and expected values are different",
        "error": true,
        "exit-status": 2,
        "output": {
            "key": "max_connections",
            "value": "15",
            "expected": "1000"
        }
    },
    "application": {
        "name": "pgagroal-cli",
        "major": 1,
        "minor": 6,
        "patch": 0,
        "version": "1.6.0"
    }
}
```

The `conf ls` command returns an array named `files` where each entry is made by a couple `description` and `path`, where the former is the mnemonic name of the configuration file, and the latter is the value of the configuration file used:

```
pgagroal-cli conf ls --format json
{
    "command": {
        "name": "conf ls",
        "status": "OK",
        "error": 0,
        "exit-status": 0,
        "output": {
            "files": {
                "list": [{
                    "description": "Main Configuration file",
                    "path": "/etc/pgagroal/pgagroal.conf"
                }, {
                    "description": "HBA File",
                    "path": "/etc/pgagroal/pgagroal_hba.conf"
                }, {
                    "description": "Limit file",
                    "path": "/etc/pgagroal/pgagroal_databases.conf"
                }, {
                    "description": "Frontend users file",
                    "path": "/etc/pgagroal/pgagroal_frontend_users.conf"
                }, {
                    "description": "Admins file",
                    "path": "/etc/pgagroal/pgagroal_admins.conf"
                }, {
```

```
                    "description": "Superuser file",
                    "path": ""
                }, {
                    "description": "Users file",
                    "path": "/etc/pgagroal/pgagroal_users.conf"
                }]
            }
        }
    },
    "application": {
        "name": "pgagroal-cli",
        "major": 1,
        "minor": 6,
        "patch": 0,
        "version": "1.6.0"
    }
}
```

13.2 pgagroal-admin

`pgagroal-admin` is a command line interface to manage users known to the **pgagroal** connection pooler. The executable accepts a set of options, as well as a command to execute. If no command is provided, the program will show the help screen.

The `pgagroal-admin` utility has the following synopsis:

```
pgagroal-admin [ OPTIONS ] [ COMMAND ]
```

13.2.1 Options

Available options are the following ones:

```
  -f, --file FILE         Set the path to a user file
  -U, --user USER         Set the user name
  -P, --password PASSWORD Set the password for the user
  -g, --generate          Generate a password
  -l, --length            Password length
  -V, --version           Display version information
  -?, --help              Display help
```

Options can be specified either in short or long form, in any position of the command line.

The `-f` option is mandatory for every operation that involves user management. If no user file is specified, `pgagroal-admin` will silently use the default one (`pgagroal_users.conf`).

The password can be passed using the environment variable `PGAGROAL_PASSWORD` instead of `-P`, however the command line argument will have precedence.

13.2.2 Commands

13.2.2.1 master-key Create or update the master key for all users.

Command:

```
pgagroal-admin master-key
```

13.2.2.2 user The `user` command allows the management of the users known to the connection pooler. The command accepts the following subcommands: - `add` to add a new user to the system; - `del` to remove an existing user from the system; - `edit` to change the credentials of an existing user; - `ls` to list all known users within the system.

The command will edit the `pgagroal_users.conf` file or any file specified by means of the `-f` option flag.

Unless the command is run with the `-U` and/or `-P` flags, the execution will be interactive.

Command:

```
pgagroal-admin user <subcommand>
```

Examples:

```
pgagroal-admin user add -U simon -P secret
pgagroal-admin user del -U simon
pgagroal-admin -f pgagroal_users.conf user add
pgagroal-admin -f pgagroal_users.conf user ls
pgagroal-admin -f pgagroal_users.conf user del -U myuser
```

13.2.3 Deprecated Commands

The following commands have been deprecated and will be removed in later releases of **pgagroal**. For each command, this is the corresponding current mapping to the working command:

- `add-user` is now `user add`;
- `remove-user` is now `user del`;
- `update-user` is now `user edit`;
- `list-users` is now `user ls`.

Whenever you use a deprecated command, the pgagroal-admin will print on standard error a warning message. If you don't want to get any warning about deprecated commands, you can redirect the stderr to /dev/**null** or any other location with:

```
pgagroal-admin user-add -U luca -P strongPassword 2>/dev/null
```

14 Performance

Performance is an important goal for **pgagroal** and effort have been made to make **pgagroal** scale and use a limited number of resources.

This chapter describes **pgagroal** performance characteristics and provides benchmarking results compared to other PostgreSQL connection pool implementations.

14.1 Benchmarking Methodology

The [pgbench][pgbench] program was used in the performance runs. All pool configurations were made with performance in mind.

The runs were performed on RHEL 7.7 / EPEL / DevTools 8 based machines on 10G network. All connection pools were the latest versions as of January 14, 2020. **pgagroal** was using the `epoll` mode of libev.

14.2 Performance Results

14.2.1 Simple Protocol

This run uses:

```
pgbench -M simple
```

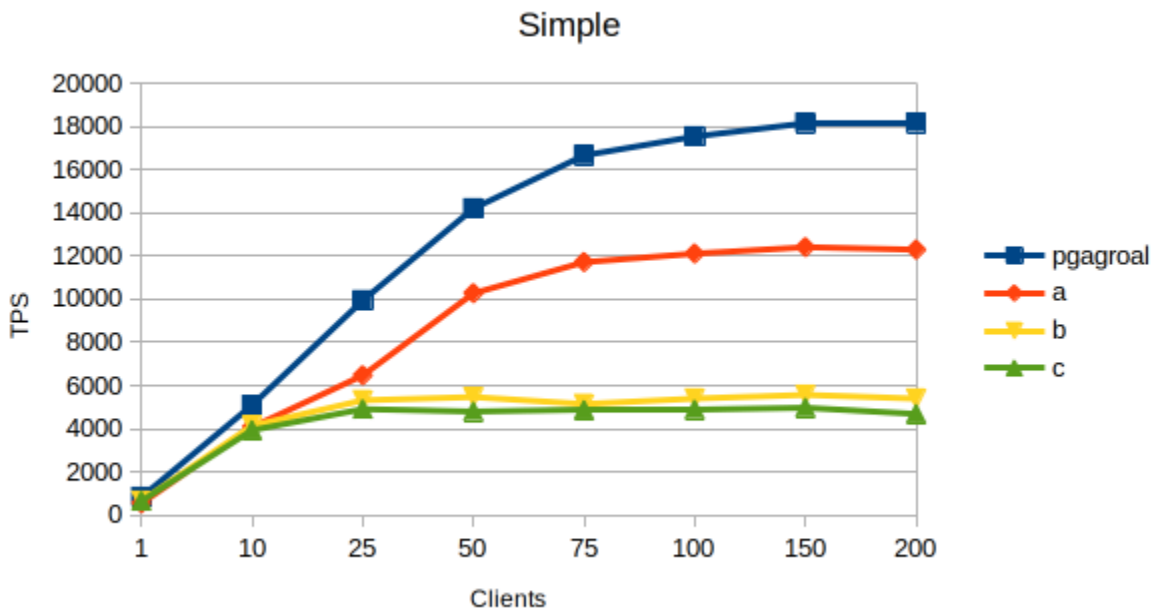

Figure 1: pgbench simple

14.2.2 Extended Protocol

This run uses:

```
pgbench -M extended
```

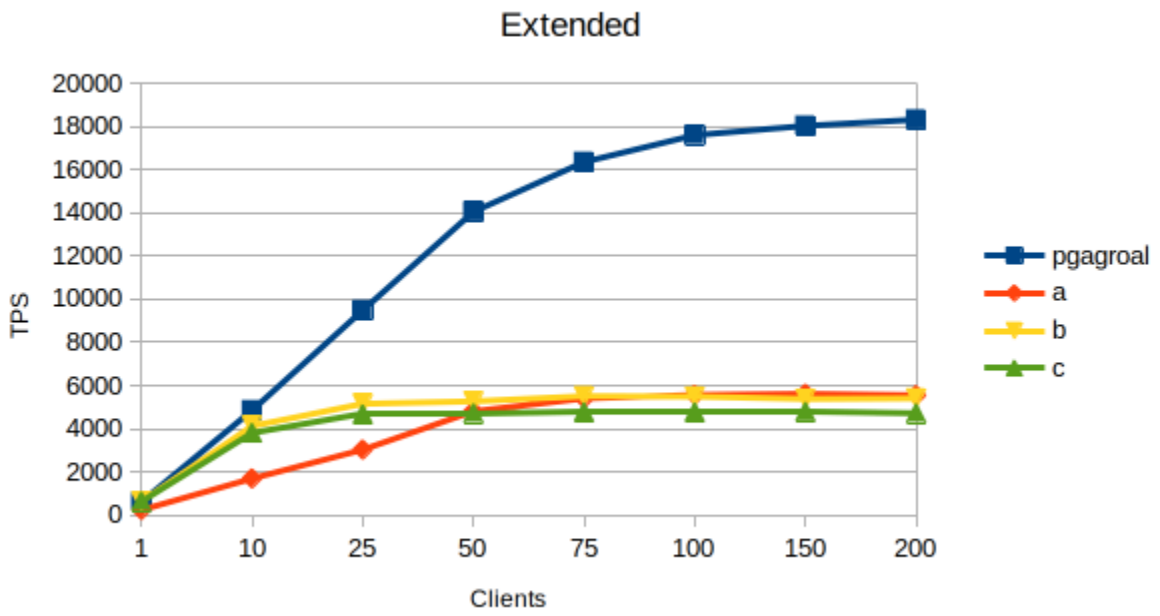

Figure 2: pgbench extended

14.2.3 Prepared Statements

This run uses:

```
pgbench -M prepared
```

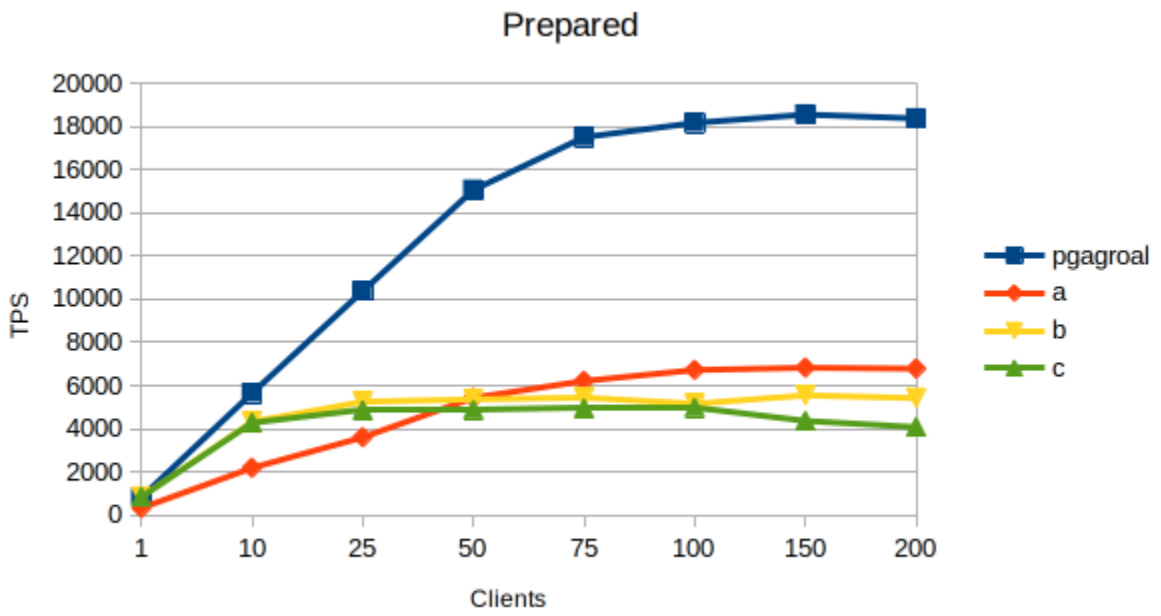

Figure 3: pgbench prepared

14.2.4 Read-Only Workload

This run uses:

```
pgbench -S
```

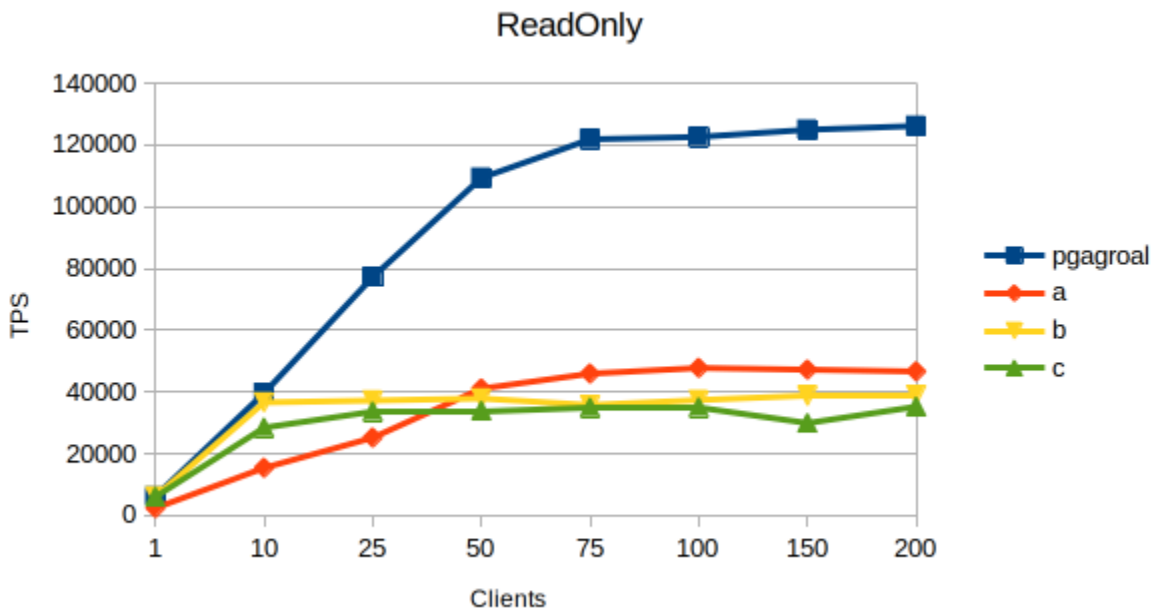

Figure 4: pgbench readonly

14.3 Performance Tuning

14.3.1 Pipeline Selection

Choose the appropriate pipeline for your workload:

- **Performance pipeline**: Fastest option for high-throughput scenarios
- **Session pipeline**: Balanced performance with full feature support
- **Transaction pipeline**: Best for applications with many short transactions

See Pipelines for detailed configuration.

14.3.2 Connection Pool Sizing

Optimal pool sizing depends on your workload:

- **CPU-bound workloads**: Pool size approximately equals number of CPU cores
- **I/O-bound workloads**: Pool size can be higher than CPU cores
- **Mixed workloads**: Start with 2x CPU cores and adjust based on monitoring

14.3.3 System-Level Optimizations

14.3.3.1 Network Configuration

- Use dedicated network interfaces for database traffic
- Configure appropriate TCP buffer sizes
- Consider using 10G or higher network speeds for high-throughput scenarios

14.3.3.2 Memory Configuration

- Enable huge pages for better memory management
- Configure appropriate shared memory settings
- Monitor memory usage patterns

14.3.3.3 CPU Configuration

- Pin pgagroal processes to specific CPU cores if needed
- Configure CPU governor for performance
- Monitor CPU utilization patterns

14.3.4 Monitoring Performance

Use the following metrics to monitor pgagroal performance:

- **Connection utilization**: Active vs. total connections
- **Response times**: Average and percentile response times
- **Throughput**: Transactions per second
- **Resource usage**: CPU, memory, and network utilization

See Prometheus for detailed monitoring setup.

15 Failover

pgagroal can failover a PostgreSQL instance if clients can't write to it.

15.1 Configuration

In `pgagroal.conf` define:

```
failover = on
failover_script = /path/to/myscript.sh
```

The script will be run as the same user as the pgagroal process so proper permissions (access and execution) must be in place.

15.2 Failover Script

The following information is passed to the script as parameters:

1. Old primary host
2. Old primary port
3. New primary host
4. New primary port

15.2.1 Example Script

A basic failover script could look like:

```
#!/bin/bash

OLD_PRIMARY_HOST=$1
OLD_PRIMARY_PORT=$2
NEW_PRIMARY_HOST=$3
NEW_PRIMARY_PORT=$4

# Promote the new primary
ssh -tt -o StrictHostKeyChecking=no postgres@${NEW_PRIMARY_HOST} pg_ctl
   promote -D /mnt/pgdata

if [ $? -ne 0 ]; then
  exit 1
fi

exit 0
```

15.2.2 Script Requirements

- The script is assumed successful if it has an exit code of 0
- Otherwise both servers will be recorded as failed
- The script should handle promotion of the new primary server
- Consider implementing proper error handling and logging

15.3 Advanced Failover Scenarios

15.3.1 Multiple Replica Configuration

When multiple replicas are available, the failover script can implement logic to:

1. Check replica lag to select the best candidate
2. Ensure proper promotion sequence
3. Update DNS or load balancer configuration
4. Notify monitoring systems

15.3.2 Automatic Failback

Consider implementing automatic failback when the original primary becomes available:

```
#!/bin/bash

# Check if original primary is healthy
if pg_isready -h $OLD_PRIMARY_HOST -p $OLD_PRIMARY_PORT; then
    # Implement failback logic
    echo "Original primary is healthy, considering failback"
fi
```

15.4 Monitoring Failover

Monitor failover events through:

- **Log files**: Check pgagroal logs for failover events
- **Prometheus metrics**: Monitor server status changes
- **External monitoring**: Implement alerts for failover events

15.5 Best Practices

1. **Test failover scripts** regularly in non-production environments
2. **Monitor replica lag** to ensure replicas are suitable for promotion
3. **Implement proper logging** in failover scripts for troubleshooting
4. **Consider network partitions** and split-brain scenarios
5. **Document failover procedures** for operational teams
6. **Use configuration management** to ensure consistent failover scripts across environments

16 Pipelines

pgagroal supports 3 different pipelines that determine how connections are managed and what features are available.

The pipeline is defined in `pgagroal.conf` under the setting:

```
pipeline = auto
```

pgagroal will choose either the performance or the session pipeline based on the configuration settings by default.

16.1 Performance Pipeline

The performance pipeline is the fastest pipeline as it is a minimal implementation of the pipeline architecture.

However, it doesn't support Transport Layer Security (TLS), failover support and the `disconnect_client` setting.

A `DISCARD ALL` query is run after each client session.

16.1.1 Configuration

Select the performance pipeline by:

```
pipeline = performance
```

16.1.2 Use Cases

The performance pipeline is ideal for: - High-throughput applications - Scenarios where maximum performance is critical - Environments where TLS is not required - Simple connection pooling needs

16.1.3 Limitations

- No Transport Layer Security (TLS) support
- No failover support
- No `disconnect_client` setting support
- Minimal feature set

16.2 Session Pipeline

The session pipeline supports all features of **pgagroal**.

A `DISCARD ALL` query is run after each client session.

16.2.1 Configuration

Select the session pipeline by:

```
pipeline = session
```

16.2.2 Features

The session pipeline supports: - Transport Layer Security (TLS) - Failover functionality - All configuration options - Complete feature set

16.2.3 Use Cases

The session pipeline is ideal for: - Production environments requiring full features - Applications needing TLS encryption - Environments with failover requirements - Complex connection pooling scenarios

16.3 Transaction Pipeline

The transaction pipeline will release the connection back to the pool after each transaction completes. This feature will support many more clients than there are database connections.

16.3.1 Configuration

Select the transaction pipeline by:

```
pipeline = transaction
```

16.3.2 Features

- Connection released after each transaction
- Supports many more clients than database connections

- Automatic transaction boundary detection
- Rollback handling for failed transactions

16.3.3 Use Cases

The transaction pipeline is ideal for: - Applications with many short transactions - Microservices architectures - High-concurrency scenarios with brief database interactions - Applications that can handle connection state loss between transactions

16.3.4 Considerations

- Application must handle loss of connection state between transactions
- Prepared statements are not preserved across transactions
- Temporary tables and other session-specific objects are not available
- May require application code changes

16.4 Pipeline Comparison

Feature	Performance	Session	Transaction
Speed	Fastest	Fast	Moderate
TLS Support	No	Yes	Yes
Failover Support	No	Yes	Yes
Connection Reuse	Session-based	Session-based	Transaction-based
Client Capacity	Limited by pool size	Limited by pool size	High
State Preservation	Session	Session	None
Complexity	Low	Medium	High

16.5 Choosing the Right Pipeline

16.5.1 Performance Pipeline

Choose when: - Maximum performance is required - TLS is not needed - Simple connection pooling is sufficient - Failover is handled externally

16.5.2 Session Pipeline

Choose when: - Full feature set is required - TLS encryption is needed - Failover support is required - Standard connection pooling behavior is desired

16.5.3 Transaction Pipeline

Choose when: - High client concurrency is needed - Connections are used for short transactions - Application can handle stateless connections - Database connection limits are a constraint

16.6 Configuration Examples

16.6.1 High-Performance Setup

```
[pgagroal]
pipeline = performance
max_connections = 100
validation = off
```

16.6.2 Production Setup with TLS

```
[pgagroal]
pipeline = session
max_connections = 50
tls = on
tls_cert_file = /path/to/cert.pem
tls_key_file = /path/to/key.pem
failover = on
failover_script = /path/to/failover.sh
```

16.6.3 High-Concurrency Setup

```
[pgagroal]
pipeline = transaction
max_connections = 20
# Support many more clients than connections
```

17 Security

This chapter provides comprehensive security guidance for **pgagroal** deployments.

17.1 Security Models

pgagroal supports multiple security models to meet different deployment requirements.

17.1.1 Pass-through Security

pgagroal uses pass-through security by default.

This means that **pgagroal** delegates to PostgreSQL to determine if the credentials used are valid.

Once a connection is obtained **pgagroal** will replay the previous communication sequence to verify the new client. This only works for connections using `trust`, `password` or `md5` authentication methods, so `scram-sha-256` based connections are not cached.

Security Considerations: - This can lead to replay attacks against `md5` based connections since the hash doesn't change - Make sure that **pgagroal** is deployed on a private trusted network - Consider using either a user vault or authentication query instead

17.1.2 User Vault

A user vault is a vault which defines the known users and their password.

The vault is static, and is managed through the `pgagroal-admin` tool.

The user vault is specified using the `-u` or `--users` command line parameter.

17.1.2.1 Frontend Users

The `-F` or `--frontend` command line parameter allows users to be defined for the client to **pgagroal** authentication. This allows the setup to use different passwords for the **pgagroal** to PostgreSQL authentication.

All users defined in the frontend authentication must be defined in the user vault (`-u`).

Frontend users (`-F`) requires a user vault (`-u`) to be defined.

17.1.3 Authentication Query

Authentication query will use the below defined function to query the database for the user password:

```
CREATE FUNCTION public.pgagroal_get_password(
  IN  p_user     name,
  OUT p_password text
) RETURNS text
LANGUAGE sql SECURITY DEFINER SET search_path = pg_catalog AS
$SELECT passwd FROM pg_shadow WHERE usename = p_user$;
```

This function needs to be installed in each database.

17.2 Network Security

17.2.1 Host-Based Authentication

Configure `pgagroal_hba.conf` to restrict access:

```
# TYPE  DATABASE USER  ADDRESS  METHOD
host    mydb     myuser 192.168.1.0/24  scram-sha-256
host    mydb     myuser 10.0.0.0/8      scram-sha-256
```

17.2.2 TLS Configuration

For complete TLS setup, see Transport Level Security (TLS).

Key security considerations: - Use strong cipher suites - Regularly update certificates - Implement proper certificate validation - Consider mutual TLS authentication

17.3 Access Control

17.3.1 User Management

Use `pgagroal-admin` to manage users securely:

```
# Create master key
pgagroal-admin -g master-key

# Add users with strong passwords
pgagroal-admin -f pgagroal_users.conf user add
```

17.3.2 Database Access Control

Configure database-specific access in `pgagroal_databases.conf`:

```
# DATABASE USER    MAX_SIZE INITIAL_SIZE MIN_SIZE
production myuser  10       5            2
test       testuser 5       2            1
```

17.3.3 Administrative Access

Secure administrative access:

```
# Create admin users with strong credentials
pgagroal-admin -f pgagroal_admins.conf -U admin user add
```

17.4 Hardening Guidelines

17.4.1 System-Level Security

1. **Run as dedicated user**: Never run pgagroal as root
2. **File permissions**: Ensure configuration files have appropriate permissions
3. **Network isolation**: Deploy on private networks when possible
4. **Firewall rules**: Restrict access to pgagroal ports
5. **Log monitoring**: Monitor logs for suspicious activity

17.4.2 Configuration Security

1. **Strong passwords**: Use complex passwords for all users
2. **Regular rotation**: Implement password rotation policies
3. **Minimal privileges**: Grant only necessary database permissions
4. **Connection limits**: Set appropriate connection limits per user/database

17.4.3 Monitoring and Auditing

1. **Enable connection logging**:

   ```
   log_connections = on
   log_disconnections = on
   ```

2. **Monitor failed authentication attempts**

3. **Set up alerts for unusual connection patterns**
4. **Regular security audits of user accounts and permissions**

17.5 Security Best Practices

17.5.1 Production Deployment

1. **Use TLS encryption** for all connections
2. **Implement proper certificate management**
3. **Regular security updates** of pgagroal and dependencies
4. **Network segmentation** to isolate database traffic
5. **Backup and disaster recovery** procedures

17.5.2 Development and Testing

1. **Separate environments** for development, testing, and production
2. **Test security configurations** before production deployment
3. **Use different credentials** for each environment
4. **Regular penetration testing** of the complete stack

17.5.3 Compliance Considerations

For environments requiring compliance (PCI DSS, HIPAA, etc.):

1. **Encryption at rest and in transit**
2. **Audit logging** of all database access
3. **Access control documentation**
4. **Regular security assessments**
5. **Incident response procedures**

17.6 Security Troubleshooting

17.6.1 Common Security Issues

Authentication failures: - Check user vault configuration - Verify password hashes - Review HBA configuration

TLS connection issues: - Verify certificate validity - Check cipher suite compatibility - Review TLS configuration

Access denied errors: - Check HBA rules - Verify user permissions - Review database access configuration

17.6.2 Security Monitoring

Monitor these security-related metrics: - Failed authentication attempts - Unusual connection patterns - Certificate expiration dates - User account activity - Administrative actions

18 Developers

This chapter provides comprehensive guidance for developers who want to contribute to pgagroal, understand its architecture, or extend its functionality.

18.1 Documentation Guide

pgagroal documentation is organized to serve different audiences and use cases. Use this guide to quickly find the information you need.

> **Note**: This manual contains the core documentation chapters. Additional standalone documentation files are located in the `doc/` directory, project root, and `contrib/` directory of the source repository. File paths shown below are relative to the project root directory.

18.1.1 Quick Reference

> **Navigation Note**: Each entry has two links separated by |: - **First link (Chapter)**: Use when reading the PDF manual (jumps to page) - **Second link (File)**: Use when browsing individual markdown files - File links will not work in PDF format

What you want to do	Where to look
Get started quickly	Getting started | 03-gettingstarted.md
Install pgagroal	Installation | 02-installation.md
Configure pgagroal	Configuration | 04-configuration.md
Set up development environment	Building pgagroal | 74-building.md
Understand the architecture	Architecture | 72-architecture.md
Write tests	Test Suite | 78-test.md
Use Git workflow	Git guide | 71-git.md
Build RPM packages	RPM | 73-rpm.md
Analyze code coverage	Code Coverage | 75-codecoverage.md
Work with core APIs	Core API | 77-core_api.md

What you want to do	Where to look
Understand event loop	Event Loop | 76-eventloop.md
Configure security/TLS	Transport Level Security (TLS) | 08-tls.md
Use command-line tools	Command Line Tools | 13-cli-tools.md
Set up monitoring	Prometheus | 11-prometheus.md
Optimize performance	Performance | 14-performance.md
Configure failover	Failover | 15-failover.md
Choose pipeline type	Pipelines | 16-pipelines.md
Harden security	Security | 17-security.md
Deploy with Docker	Docker | 12-docker.md
Configure database aliases	Database Aliases | 09-database_alias.md
Manage user credentials	Vault | 10-vault.md
Contribute to project	Git guide | 71-git.md, see also `CONTRIBUTING.md` in project root
Report issues or get help	GitHub Issues: https://github.com/pgagroal/pgagroal/issues

18.1.2 User Documentation

18.1.2.1 Manual Chapters (Comprehensive Guide)

> **Navigation Note**: Each table entry has two links - a **Chapter** link and a **File** link: - **If reading the PDF manual**: Use the **Chapter** links (first column) to navigate within the PDF - **If reading individual markdown files**: Use the **File** links (second column) to open specific files - The File links will not work in PDF format, and Chapter links may not work when browsing individual files

User-Focused Chapters (01-17):

Chapter	File	Description
Introduction	01-introduction.md	Overview of pgagroal features and manual structure
Installation	02-installation.md	Step-by-step setup for Rocky Linux, PostgreSQL 17, and pgagroal

Chapter	File	Description
Getting Started	03-gettingstarted.md	Quick introduction to basic pgagroal usage and configuration
Configuration	04-configuration.md	Comprehensive guide to all configuration files and options
Prefill	05-prefill.md	How to configure and use connection prefill for performance
Remote Management	06-remote_management.md	Setting up and using remote management features
Security Model	07-split_security.md	Implementing split security models for authentication
Transport Level Security	08-tls.md	Configuring TLS for secure connections
Database Aliases	09-database_alias.md	Using database aliases for flexible client connections
Vault	10-vault.md	Managing user credentials and secrets with pgagroal vault
Prometheus	11-prometheus.md	Integrating Prometheus metrics and monitoring
Docker	12-docker.md	Running pgagroal in Docker containers
Command Line Tools	13-cli-tools.md	Comprehensive CLI tools reference (pgagroal-cli, pgagroal-admin)
Performance	14-performance.md	Performance benchmarks, tuning, and optimization
Failover	15-failover.md	Failover configuration and scripting
Pipelines	16-pipelines.md	Pipeline types and configuration
Security	17-security.md	Comprehensive security hardening guide

Developer-Focused Chapters (70-79):

Chapter	File	Description
Developers	70-dev.md	Development environment setup and contribution guidelines (this chapter)
Git Guide	71-git.md	Git workflow and version control practices for the project
Architecture	72-architecture.md	High-level architecture and design of pgagroal
RPM	73-rpm.md	Building and using RPM packages
Building pgagroal	74-building.md	Compiling pgagroal from source
Code Coverage	75-codecoverage.md	Code coverage analysis and testing practices
Event Loop	76-eventloop.md	Understanding the event loop implementation
Core API	77-core_api.md	Reference for core API functions
Test Suite	78-test.md	Testing frameworks and procedures
Distribution Installation	79-distributions.md	Platform-specific installation notes

Reference Chapters (97-99):

Chapter	File	Description
Acknowledgements	97-acknowledgement.md	Credits and contributors
Licenses	98-licenses.md	License information
References	99-references.md	Additional resources and references

18.1.2.2 Additional User Resources The manual chapters above provide comprehensive coverage. Additional standalone files in `doc/` directory provide supplementary information:

- **doc/GETTING_STARTED.md** - Alternative quick start guide (supplements Getting Started | 03-gettingstarted.md)
- **doc/VAULT.md** - Additional vault examples (supplements Vault | 10-vault.md)

18.1.3 Administrator Documentation

All administration topics are covered in this manual:

> **Navigation Note**: Use **Chapter** links when reading the PDF manual, **File** links when browsing individual markdown files.

Chapter	File	Description
Configuration	04-configuration.md	Complete configuration reference
Remote Management	06-remote_management.md	Remote management setup
Transport Level Security	08-tls.md	TLS configuration
Vault	10-vault.md	User credential management
Command Line Tools	13-cli-tools.md	Complete CLI reference (pgagroal-cli, pgagroal-admin)
Performance	14-performance.md	Performance tuning and benchmarks
Failover	15-failover.md	Failover configuration and procedures
Pipelines	16-pipelines.md	Pipeline configuration and usage
Security	17-security.md	Security hardening and best practices
Prometheus	11-prometheus.md	Monitoring and metrics
Docker	12-docker.md	Container deployment

Legacy standalone documentation files (now superseded by manual chapters):

Legacy File	Superseded By	Chapter
doc/CLI.md	Command Line Tools	13-cli-tools.md
doc/ADMIN.md	Command Line Tools	13-cli-tools.md
doc/PERFORMANCE.md	Performance	14-performance.md
doc/FAILOVER.md	Failover	15-failover.md

Legacy File	Superseded By	Chapter
doc/PIPELINES.md	Pipelines	16-pipelines.md
doc/SECURITY.md	Security	17-security.md
doc/DISTRIBUTIONS.md	Distribution Installation	79-distributions.md

18.1.4 Developer Documentation

Essential reading for contributors and developers:

> **Navigation Note**: Use **Chapter** links when reading the PDF manual, **File** links when browsing individual markdown files.

Chapter	File	Description
Architecture	72-architecture.md	High-level architecture and design of pgagroal
Building pgagroal	74-building.md	Compiling pgagroal from source with development options
Git Guide	71-git.md	Git workflow and version control practices for the project
Test Suite	78-test.md	Testing frameworks and procedures
Code Coverage	75-codecoverage.md	Code coverage analysis and testing practices
Event Loop	76-eventloop.md	Understanding the event loop implementation
Core API	77-core_api.md	Reference for core API functions
RPM	73-rpm.md	Building and using RPM packages

Additional developer resources (supplements manual chapters):

File	Supplements	Description
doc/DEVELOPERS.md	Building pgagroal | 74-building.md	Detailed development environment setup
doc/ARCHITECT.md	Architecture | 72-architecture.md	Extended architecture documentation
doc/TEST.md	Test Suite | 78-test.md	Extended testing documentation

18.1.5 Project Management & Planning

Files in the project root directory:

File	Description
CONTRIBUTING.md	Contribution guidelines and legal information
README.md	Project overview and quick start
AUTHORS	List of project contributors
LICENSE	Project license information
CODE_OF_CONDUCT.md	Community guidelines and conduct policies

18.1.6 Security & Certificate Documentation

Security topics covered in this manual:

> **Navigation Note**: Use **Chapter** links when reading the PDF manual, **File** links when browsing individual markdown files.

Chapter	File	Description
Transport Level Security (TLS)	08-tls.md	Complete TLS configuration and certificate setup
Security Model	07-split_security.md	Advanced security models

18.1.7 Testing & Development Scripts

Testing covered in this manual:

> **Navigation Note**: Use **Chapter** links when reading the PDF manual, **File** links when browsing individual markdown files.

Chapter	File	Description
Test Suite	78-test.md	Complete testing procedures and frameworks
Code Coverage	75-codecoverage.md	Code coverage analysis

Additional testing documentation in project root:

File	Description
TEST.md	Root-level testing documentation

18.1.8 Configuration Examples & Templates

Configuration file templates and examples in `doc/etc/`:

File	Description
doc/etc/pgagroal.conf	Main configuration file template
doc/etc/pgagroal_hba.conf	Host-based authentication template
doc/etc/pgagroal_vault.conf	Vault configuration template
doc/etc/pgagroal.service	Systemd service file
doc/etc/pgagroal.socket	Systemd socket file

18.1.9 Contrib Directory (Additional Tools & Examples)

Community contributions and additional tools in `contrib/`:

Path	Description
contrib/docker/	Docker configuration examples and Dockerfiles
contrib/grafana/README.md	Grafana dashboard setup and configuration
contrib/prometheus_scrape/README.md	Prometheus metrics documentation generator
contrib/valgrind/README.md	Valgrind memory debugging configuration
contrib/shell_comp/	Shell completion scripts for bash and zsh

18.1.10 Man Pages (Reference Documentation)

Complete command-line and configuration reference in `doc/man/`:

File	Description
doc/man/pgagroal.1.rst	Main pgagroal command reference
doc/man/pgagroal-cli.1.rst	CLI tool reference
doc/man/pgagroal-admin.1.rst	Admin tool reference
doc/man/pgagroal-vault.1.rst	Vault tool reference
doc/man/pgagroal.conf.5.rst	Main configuration file reference
doc/man/pgagroal_hba.conf.5.rst	HBA configuration reference
doc/man/pgagroal_databases.conf.5.rst	Database limits configuration reference
doc/man/pgagroal_vault.conf.5.rst	Vault configuration reference

18.1.11 Reference Materials

> **Navigation Note**: Use **Chapter** links when reading the PDF manual, **File** links when browsing individual markdown files.

Chapter	File	Description
Acknowledgements	97-acknowledgement.md	Credits and contributors
Licenses	98-licenses.md	License information

Chapter	File	Description
References	99-references.md	Additional resources and references

18.2 Development Environment Setup

For detailed development environment setup, see Building pgagroal | 74-building.md. Here's a quick overview:

18.2.1 Prerequisites

For Fedora-based systems:

```
dnf install git gcc cmake make liburing liburing-devel openssl openssl-
    devel systemd systemd-devel python3-docutils libatomic zlib zlib-devel
    libzstd libzstd-devel lz4 lz4-devel bzip2 bzip2-devel libasan libasan-
    static binutils clang clang-analyzer clang-tools-extra
```

18.2.2 Quick Build

```
git clone https://github.com/pgagroal/pgagroal.git
cd pgagroal
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Debug -DCMAKE_INSTALL_PREFIX=/usr/local ..
make
make install
```

18.3 Contributing Workflow

1. **Fork the repository** on GitHub
2. **Clone your fork** locally
3. **Create a feature branch** from main
4. **Make your changes** following the coding guidelines
5. **Test your changes** thoroughly
6. **Submit a pull request** with a clear description

For detailed Git workflow, see Git guide | 71-git.md.

18.4 Key Development Principles

- **Security First**: Always consider security implications
- **Performance Matters**: pgagroal is a high-performance connection pool
- **Code Quality**: Follow established patterns and write tests
- **Documentation**: Update documentation with your changes
- **Community**: Engage with the community for feedback

18.5 Getting Help

- **GitHub Discussions** (https://github.com/pgagroal/pgagroal/discussions) - Ask questions
- **GitHub Issues** (https://github.com/pgagroal/pgagroal/issues) - Report bugs or request features
- **Code of Conduct** (see `CODE_OF_CONDUCT.md` in project root) - Community guidelines

18.6 Git guide

Here are some links that will help you

- How to Squash Commits in Git
- ProGit book

Start by forking the repository

This is done by the "Fork" button on GitHub.

Clone your repository locally

This is done by

```
git clone git@github.com:<username>/pgagroal.git
```

Add upstream

Do

```
cd pgagroal
git remote add upstream https://github.com/pgagroal/pgagroal.git
```

Do a work branch

```
git checkout -b mywork main
```

Make the changes

Remember to verify the compile and execution of the code.

Use

```
[#xyz] Description
```

as the commit message where `[#xyz]` is the issue number for the work, and `Description` is a short description of the issue in the first line

Multiple commits

If you have multiple commits on your branch then squash them

```
git rebase -i HEAD~2
```

for example. It is `p` for the first one, then `s` for the rest

Rebase

Always rebase

```
git fetch upstream
git rebase -i upstream/main
```

Force push

When you are done with your changes force push your branch

```
git push -f origin mywork
```

and then create a pull request for it

Format source code

Use the `clang-format.sh` script from the root directory of the project to apply consistent formatting:

```
./clang-format.sh
```

Note that the project uses `clang-format` version 21 (or higher).

Repeat

Based on feedback keep making changes, squashing, rebasing and force pushing

Undo

Normally you can reset to an earlier commit using `git reset <commit hash> --hard`.

But if you accidentally squashed two or more commits, and you want to undo that, you need to know where to reset to, and the commit seems to have lost after you rebased.

But they are not actually lost - using `git reflog`, you can find every commit the HEAD pointer has ever pointed to. Find the commit you want to reset to, and do `git reset --hard`.

18.7 Architecture

18.7.1 Overview

pgagroal use a process model (`fork()`), where each process handles one connection to PostgreSQL. This was done such a potential crash on one connection won't take the entire pool down.

The main process is defined in main.c. When a client connects it is processed in its own process, which is handle in worker.h (worker.c).

Once the client disconnects the connection is put back in the pool, and the child process is terminated.

18.7.2 Shared memory

A memory segment (shmem.h) is shared among all processes which contains the **pgagroal** state containing the configuration of the pool, the list of servers and the state of each connection.

The configuration of **pgagroal** (`struct configuration`), the configuration of the servers (`struct server`) and the state of each connection (`struct connection`) is initialized in this shared memory segment. These structs are all defined in pgagroal.h.

The shared memory segment is created using the `mmap()` call.

18.7.3 Atomic operations

The atomic operation library is used to define the state of each of the connection, and move them around in the connection state diagram. The state diagram has the follow states

State name	Description
`STATE_NOTINIT`	The connection has not been initialized
`STATE_INIT`	The connection is being initialized
`STATE_FREE`	The connection is free
`STATE_IN_USE`	The connection is in use
`STATE_GRACEFULLY`	The connection will be killed upon return to the pool
`STATE_FLUSH`	The connection is being flushed

State name	Description
`STATE_IDLE_CHECK`	The connection is being idle timeout checked
`STATE_MAX_CONNECTION_AGE`	The connection is being max connection age checked
`STATE_VALIDATION`	The connection is being validated
`STATE_REMOVE`	The connection is being removed

These state are defined in pgagroal.h.

18.7.4 Pool

The **pgagroal** pool API is defined in pool.h (pool.c).

This API defines the functionality of the pool such as getting a connection from the pool, and returning it. There is no ordering among processes, so a newly created process can obtain a connection before an older process.

The pool operates on the `struct connection` data type defined in pgagroal.h.

18.7.5 Network and messages

All communication is abstracted using the `struct message` data type defined in message.h.

Reading and writing messages are handled in the message.h (message.c) files.

Network operations are defined in network.h (network.c).

18.7.6 Memory

Each process uses a fixed memory block for its network communication, which is allocated upon startup of the worker.

That way we don't have to allocate memory for each network message, and more importantly free it after end of use.

The memory interface is defined in memory.h (memory.c).

18.7.7 Management

pgagroal has a management interface which defines the administrator abilities that can be performed when it is running. This include for example taking a backup. The `pgagroal-cli` program is used for these operations (cli.c).

The management interface is defined in management.h. The management interface uses its own protocol which uses JSON as its foundation.

Write

The client sends a single JSON string to the server,

Field	Type	Description
`compression`	uint8	The compression type
`encryption`	uint8	The encryption type
`length`	uint32	The length of the JSON document
`json`	String	The JSON document

The server sends a single JSON string to the client,

Field	Type	Description
`compression`	uint8	The compression type
`encryption`	uint8	The encryption type
`length`	uint32	The length of the JSON document
`json`	String	The JSON document

Read

The server sends a single JSON string to the client,

Field	Type	Description
`compression`	uint8	The compression type
`encryption`	uint8	The encryption type

Field	Type	Description
`length`	uint32	The length of the JSON document
`json`	String	The JSON document

The client sends to the server a single JSON documents,

Field	Type	Description
`compression`	uint8	The compression type
`encryption`	uint8	The encryption type
`length`	uint32	The length of the JSON document
`json`	String	The JSON document

Remote management

The remote management functionality uses the same protocol as the standard management method.

However, before the management packet is sent the client has to authenticate using SCRAM-SHA-256 using the same message format that PostgreSQL uses, e.g. StartupMessage, AuthenticationSASL, AuthenticationSASLContinue, AuthenticationSASLFinal and AuthenticationOk. The SSLRequest message is supported.

The remote management interface is defined in remote.h (remote.c).

18.7.8 I/O Layer

The I/O layer interface is primarily defined in ev.h (and implemented in ev.c).

These files contain the definition and implementation of the event loop for the three supported backends: `io_uring`, `epoll`, and `kqueue`.

liburing was used for setup and usage io_uring instances.

Each process has its own event loop, such that the process only gets notified when data related only to that process is ready. The main loop handles the system wide “services” such as idle timeout checks and so on.

Event Loop Context Management

The event loop system uses execution contexts to properly handle different pgagroal components:

Supported Contexts: - `PGAGROAL_CONTEXT_MAIN`: Main pgagroal process context - `PGAGROAL_CONTEXT_VAULT`: pgagroal-vault HTTP server context

Context Setting: Each process sets its context before initializing the event loop using `pgagroal_event_set_context()`. This ensures: - Correct configuration structure is accessed (main_configuration vs vault_configuration) - Proper event backend selection based on the component's configuration - Isolated event loop behavior per component

Backend Configuration: Both main and vault contexts support the same `ev_backend` configuration options: - `auto`: Selects best available backend automatically - `io_uring`: High-performance Linux backend (disabled with TLS) - `epoll`: Traditional Linux event notification - `kqueue`: BSD/macOS event mechanism

The context approach prevents configuration structure mismatches and allows independent event backend configuration for different pgagroal components.

18.7.9 Pipeline

pgagroal has the concept of a pipeline that defines how communication is routed from the client through **pgagroal** to PostgreSQL. Likewise in the other direction.

A pipeline is defined by

```
struct pipeline
{
   initialize initialize;
   start start;
   callback client;
   callback server;
   stop stop;
   destroy destroy;
   periodic periodic;
};
```

in pipeline.h.

The functions in the pipeline are defined as

Function	Description
`initialize`	Global initialization of the pipeline, may return a pointer to a shared memory segment
`start`	Called when the pipeline instance is started
`client`	Client to **pgagroal** communication

Function	Description
server	PostgreSQL to **pgagroal** communication
stop	Called when the pipeline instance is stopped
destroy	Global destruction of the pipeline
periodic	Called periodic

The functions start, client, server and stop has access to the following information

```
struct worker_io
{
   struct io_watcher io; /* The base type for io operations */
   int client_fd;        /* The client descriptor */
   int server_fd;        /* The server descriptor */
   int slot;             /* The slot */
   SSL* client_ssl;      /* The client SSL context */
   SSL* server_ssl;      /* The server SSL context */
};
```

defined in worker.h.

Performance pipeline

One of the goals for **pgagroal** is performance, so the performance pipeline will only look for the Terminate message from the client and act on that. Likewise the performance pipeline will only look for FATAL errors from the server. This makes the pipeline very fast, since there is a minimum overhead in the interaction.

The pipeline is defined in pipeline_perf.c in the functions

Function	Description
performance_initialize	Nothing
performance_start	Nothing
performance_client	Client to **pgagroal** communication
performance_server	PostgreSQL to **pgagroal** communication
performance_stop	Nothing
performance_destroy	Nothing
performance_periodic	Nothing

Session pipeline

The session pipeline works like the performance pipeline with the exception that it checks if a Transport Layer Security (TLS) transport should be used.

The pipeline is defined in pipeline_session.c in the functions

Function	Description
`session_initialize`	Initialize memory segment if disconnect_client is active
`session_start`	Prepares the client segment if disconnect_client is active
`session_client`	Client to **pgagroal** communication
`session_server`	PostgreSQL to **pgagroal** communication
`session_stop`	Updates the client segment if disconnect_client is active
`session_destroy`	Destroys memory segment if initialized
`session_periodic`	Checks if clients should be disconnected

Transaction pipeline

The transaction pipeline will return the connection to the server after each transaction. The pipeline supports Transport Layer Security (TLS).

The pipeline uses the ReadyForQuery message to check the status of the transaction, and therefore needs to maintain track of the message headers.

The pipeline has a management interface in order to receive the socket descriptors from the parent process when a new connection is added to the pool. The pool will retry if the client in question doesn't consider the socket descriptor valid.

The pipeline is defined in pipeline_transaction.c in the functions

Function	Description
`transaction_initialize`	Nothing
`transaction_start`	Setup process variables and returns the connection to the pool
`transaction_client`	Client to **pgagroal** communication. Obtain connection if needed

Function	Description
transaction_server	PostgreSQL to **pgagroal** communication. Keep track of message headers
transaction_stop	Return connection to the pool if needed. Possible rollback of active transaction
transaction_destroy	Nothing
transaction_periodic	Nothing

18.7.10 Signals

The main process of **pgagroal** supports the following signals SIGTERM, SIGINT and SIGALRM as a mechanism for shutting down. The SIGTRAP signal will put **pgagroal** into graceful shutdown, meaning that exisiting connections are allowed to finish their session. The SIGABRT is used to request a core dump (abort()). The SIGHUP signal will trigger a full reload of the configuration. When SIGHUP is received, **pgagroal** will re-read the configuration from the configuration files on disk and apply any changes that can be handled at runtime. This is the standard way to apply changes made to the configuration files.

In contrast, the SIGUSR1 signal will trigger a service reload, but **does not** re-read the configuration files. Instead, SIGUSR1 restarts sockets and listeners using the current in-memory configuration. This is useful for applying certain changes (such as re-opening sockets or refreshing listeners) without modifying or reloading the configuration from disk. Any changes made to the configuration files will **not** be picked up when using SIGUSR1; only the configuration already loaded in memory will be used.

The child processes support SIGQUIT as a mechanism to shutdown. This will not shutdown the pool itself.

It should not be needed to use SIGKILL for **pgagroal**. Please, consider using SIGABRT instead, and share the core dump and debug logs with the **pgagroal** community.

18.7.11 Reload

The SIGHUP signal will trigger a reload of the configuration.

However, some configuration settings requires a full restart of **pgagroal** in order to take effect. These are

- `hugepage`
- `log_path`
- `log_type`
- `max_connections`
- `pipeline`
- `unix_socket_dir`
- `pidfile`
- Limit rules defined by `pgagroal_databases.conf`
- TLS rules defined by server section

The configuration can also be reloaded using `pgagroal-cli -c pgagroal.conf conf reload`. The command is only supported over the local interface, and hence doesn't work remotely.

18.7.12 Prometheus

pgagroal has support for Prometheus when the `metrics` port is specified.

Note: It is crucial to carefully initialize Prometheus memory in any program files for example functions like `pgagroal_init_prometheus()` and `pgagroal_init_prometheus_cache()` should only be invoked if `metrics` is greater than 0.

The module serves two endpoints

- / - Overview of the functionality (`text/html`)
- `/metrics` - The metrics (`text/plain`)

All other URLs will result in a 403 response.

The metrics endpoint supports `Transfer-Encoding: chunked` to account for a large amount of data.

The implementation is done in prometheus.h and prometheus.c.

18.7.13 Failover support

pgagroal can failover a PostgreSQL instance if clients can't write to it.

This is done using an external script provided by the user.

The implementation is done in server.h and server.c.

18.7.14 Logging

Simple logging implementation based on a `atomic_schar` lock.

The implementation is done in logging.h and logging.c.

Level	Description
TRACE	Information for developers including values of variables
DEBUG	Higher level information for developers - typically about flow control and the value of key variables
INFO	A user command was successful or general health information about the system
WARN	A user command didn't complete correctly so attention is needed
ERROR	Something unexpected happened - try to give information to help identify the problem
FATAL	We can't recover - display as much information as we can about the problem and `exit(1)`

18.7.15 Protocol

The protocol interactions can be debugged using Wireshark or pgprtdbg.

18.8 RPM

pgagroal can be built into a RPM for Fedora systems.

Requirements

```
dnf install gcc rpm-build rpm-devel rpmlint make python bash coreutils
   diffutils patch rpmdevtools chrpath
```

Setup RPM development

```
rpmdev-setuptree
```

Create source package

```
git clone https://github.com/pgagroal/pgagroal.git
cd pgagroal
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make package_source
```

Create RPM package

```
cp pgagroal-$VERSION.tar.gz ~/rpmbuild/SOURCES
QA_RPATHS=0x0001 rpmbuild -bb pgagroal.spec
```

The resulting RPM will be located in ~/rpmbuild/RPMS/x86_64/, if your architecture is x86_64 .

18.9 Building pgagroal

18.9.1 Overview

pgagroal can be built using CMake, where the build system will detect the compiler and apply appropriate flags for debugging and testing.

The main build system is defined in [CMakeLists.txt][cmake_txt]. The flags for Sanitizers are added in compile options in [src/CMakeLists.txt][src/cmake_txt]

18.9.2 Compiling

Install the dependencies with

```
dnf install git gcc cmake make      \
            liburing liburing-devel \
            openssl openssl-devel   \
            systemd systemd-devel   \
            python3-docutils        \
            libatomic               \
            zlib zlib-devel         \
            libzstd libzstd-devel   \
            lz4 lz4-devel           \
            bzip2 bzip2-devel       \
            clang clang-analyzer clang-tools-extra
```

To build **pgagroal** in release mode:

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/usr/local ..
make
```

or in debug mode:

```
mkdir build
cd build
cmake -DCMAKE_C_COMPILER=clang -DCMAKE_BUILD_TYPE=Debug ..
make
```

The compiler can also be specified for example

```
cmake -DCMAKE_C_COMPILER=gcc -DCMAKE_BUILD_TYPE=Debug ..
# or
cmake -DCMAKE_C_COMPILER=clang -DCMAKE_BUILD_TYPE=Debug .
```

The build system will automatically detect the compiler version and enable the appropriate flags based on support.

18.9.3 Compiling the documentation

pgagroal's documentation requires

- pandoc
- texlive

```
dnf install pandoc texlive-scheme-basic \
            'tex(footnote.sty)' 'tex(footnotebackref.sty)' \
            'tex(pagecolor.sty)' 'tex(hardwrap.sty)' \
            'tex(mdframed.sty)' 'tex(sourcesanspro.sty)' \
            'tex(ly1enc.def)' 'tex(sourcecodepro.sty)' \
            'tex(titling.sty)' 'tex(csquotes.sty)' \
            'tex(zref-abspage.sty)' 'tex(needspace.sty)'
```

You will need the `Eisvogel` template as well which you can install through

```
wget https://github.com/Wandmalfarbe/pandoc-latex-template/releases/
   download/v3.3.0/Eisvogel-3.3.0.tar.gz
tar -xzf Eisvogel-3.3.0.tar.gz
mkdir -p ~/.local/share/pandoc/templates
mv Eisvogel-3.3.0/eisvogel.latex ~/.local/share/pandoc/templates/
```

where `$HOME` is your home directory.

18.9.4 Generate API guide

This process is optional. If you choose not to generate the API HTML files, you can opt out of downloading these dependencies, and the process will automatically skip the generation.

Download dependencies

```
dnf install graphviz doxygen
```

These packages will be detected during `cmake` and built as part of the main build.

18.9.5 Policy and guidelines for using AI

Our goal in the pgagroal project is to develop an excellent software system. This requires careful attention to detail in every change we integrate. Maintainer time and attention is very limited, so it's important that changes you ask us to review represent your best work.

You are encouraged to use tools that help you write good code, including AI tools. However, as noted above, you always need to understand and explain the changes you're proposing to make, whether

or not you used an LLM as part of your process to produce them. The answer to “Why did you make change X?” should never be “I’m not sure. The AI did it.”

Do not submit an AI-generated PR you haven’t personally understood and tested, as this wastes maintainers’ time. PRs that appear to violate this guideline will be closed without review. Using AI as a coding assistant and help you create a skeleton for your code.

- Don’t skip becoming familiar with the part of the codebase you’re working on. This will let you write better prompts and validate their output if you use an LLM. Code assistants can be a useful search engine/discovery tool in this process, but don’t trust claims they make about how pgagroal works. LLMs are often wrong, even about details that are clearly answered in the pgagroal documentation and surrounding code
- Don’t simply ask an LLM to add code comments, as it will likely produce a bunch of text that unnecessarily explains what’s already clear from the code. If using an LLM to generate comments, be really specific in your request, demand succinctness, and carefully edit the result.

Using AI for communication

As noted above, pgagroal’s contributors are expected to communicate with intention, to avoid wasting maintainer time with long, sloppy writing. We strongly prefer clear and concise communication about points that actually require discussion over long AI-generated comments.

When you use an LLM to write a message for you, it remains your responsibility to read through the whole thing and make sure that it makes sense to you and represents your ideas concisely. A good rule of thumb is that if you can’t make yourself carefully read some LLM output that you generated, nobody else wants to read it either.

Here are some concrete guidelines for using LLMs as part of your communication workflows.

- When writing a pull request description, do not include anything that’s obvious from looking at your changes directly (e.g., files changed, functions updated, etc.). Instead, focus on the why behind your changes. Don’t ask an LLM to generate a PR description on your behalf based on your code changes, as it will simply regurgitate the information that’s already there.
- Similarly, when responding to a pull request comment, explain your reasoning. Don’t prompt an LLM to re-describe what can already be seen from the code.
- Verify that everything you write is accurate, whether or not an LLM generated any part of it. pgagroal’s maintainers will be unable to review your contributions if you misrepresent your work (e.g., misdescribing your code changes, their effect, or your testing process).
- Complete all parts of the PR description template, maybe with screenshots and the self-review checklist. Don’t simply overwrite the template with LLM output.
- Clarity and succinctness are much more important than perfect grammar, so you shouldn’t feel obliged to pass your writing through an LLM. If you do ask an LLM to clean up your writing style,

be sure it does not make it longer in the process. Demand succinctness in your prompt.
- Quoting an LLM answer is usually less helpful than linking to relevant primary sources, like source code, reference documentation, or web standards. If you do need to quote an LLM answer in a pgagroal conversation, put the answer in a pgagroal quote block, to distinguish LLM output from your own thoughts.

18.9.6 Sanitizer

Before building pgagroal with sanitizer support, ensure you have the required packages installed:

- `libasan` - AddressSanitizer runtime library
- `libasan-static` - Static version of AddressSanitizer runtime library

On Red Hat/Fedora systems:

```
sudo dnf install libasan libasan-static
```

Package names and versions may vary depending on your distribution and compiler version.

18.9.7 Sanitizer Flags

AddressSanitizer (ASAN)

Address Sanitizer is a memory error detector that helps find use-after-free, heap/stack/global buffer overflow, use-after-return, initialization order bugs, and memory leaks.

UndefinedBehaviorSanitizer (UBSAN)

UndefinedBehaviorSanitizer is a fast undefined behavior detector that can find various types of undefined behavior during program execution, such as integer overflow, null pointer dereference, and more.

Common Flags

- `-fno-omit-frame-pointer` - Provides better stack traces in error reports
- `-Wall -Wextra` - Enables additional compiler warnings

GCC Support

- `-fsanitize=address` - Enables the Address Sanitizer (GCC 4.8+)
- `-fsanitize=undefined` - Enables the Undefined Behavior Sanitizer (GCC 4.9+)
- `-fno-sanitize=alignment` - Disables alignment checking (GCC 5.1+)
- `-fno-sanitize-recover=all` - Makes all sanitizers halt on error (GCC 5.1+)

- `-fsanitize=float-divide-by-zero` - Detects floating-point division by zero (GCC 5.1+)
- `-fsanitize=float-cast-overflow` - Detects floating-point cast overflows (GCC 5.1+)
- `-fsanitize-recover=address` - Allows the program to continue execution after detecting an error (GCC 6.0+)
- `-fsanitize-address-use-after-scope` - Detects use-after-scope bugs (GCC 7.0+)

Clang Support

- `-fsanitize=address` - Enables the Address Sanitizer (Clang 3.2+)
- `-fno-sanitize=null` - Disables null pointer dereference checking (Clang 3.2+)
- `-fno-sanitize=alignment` - Disables alignment checking (Clang 3.2+)
- `-fsanitize=undefined` - Enables the Undefined Behavior Sanitizer (Clang 3.3+)
- `-fsanitize=float-divide-by-zero` - Detects floating-point division by zero (Clang 3.3+)
- `-fsanitize=float-cast-overflow` - Detects floating-point cast overflows (Clang 3.3+)
- `-fno-sanitize-recover=all` - Makes all sanitizers halt on error (Clang 3.6+)
- `-fsanitize-recover=address` - Allows the program to continue execution after detecting an error (Clang 3.8+)
- `-fsanitize-address-use-after-scope` - Detects use-after-scope bugs (Clang 3.9+)

18.9.8 Additional Sanitizer Options

Developers can add additional sanitizer flags via environment variables. Some useful options include:

ASAN Options

- `ASAN_OPTIONS=detect_leaks=1` - Enables memory leak detection
- `ASAN_OPTIONS=halt_on_error=0` - Continues execution after errors
- `ASAN_OPTIONS=detect_stack_use_after_return=1` - Enables stack use-after-return detection
- `ASAN_OPTIONS=check_initialization_order=1` - Detects initialization order problems
- `ASAN_OPTIONS=strict_string_checks=1` - Enables strict string function checking
- `ASAN_OPTIONS=detect_invalid_pointer_pairs=2` - Enhanced pointer pair validation
- `ASAN_OPTIONS=print_stats=1` - Prints statistics about allocated memory
- `ASAN_OPTIONS=verbosity=1` - Increases logging verbosity

UBSAN Options

- `UBSAN_OPTIONS=print_stacktrace=1` - Prints stack traces for errors
- `UBSAN_OPTIONS=halt_on_error=1` - Stops execution on the first error
- `UBSAN_OPTIONS=silence_unsigned_overflow=1` - Silences unsigned integer overflow reports

18.9.9 Building with Sanitizers

To build **pgagroal** with sanitizer support:

```
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Debug ..
make
```

The compiler can also be specified

```
cmake -DCMAKE_C_COMPILER=gcc -DCMAKE_BUILD_TYPE=Debug ..
# or
cmake -DCMAKE_C_COMPILER=clang -DCMAKE_BUILD_TYPE=Debug .
```

The build system will automatically detect the compiler version and enable the appropriate sanitizer flags based on support.

18.10 Running with Sanitizers

When running **pgagroal** built with sanitizers, any errors will be reported to stderr.

To get more detailed reports, you can set additional environment variables:

```
ASAN_OPTIONS=detect_leaks=1:halt_on_error=0:detect_stack_use_after_return
   =1 ./pgagroal
```

You can combine ASAN and UBSAN options:

```
ASAN_OPTIONS=detect_leaks=1 UBSAN_OPTIONS=print_stacktrace=1 ./pgagroal
```

18.11 Advanced Sanitizer Options Not Included by Default

Developers may want to experiment with additional sanitizer flags not enabled by default:

- `-fsanitize=memory` - Enables MemorySanitizer (MSan) for detecting uninitialized reads (Note this can't be used with ASan)
- `-fsanitize=integer` - Only check integer operations (subset of UBSan)

- `-fsanitize=bounds` - Array bounds checking (subset of UBSan)
- `-fsanitize-memory-track-origins` - Tracks origins of uninitialized values (with MSan)
- `-fsanitize-memory-use-after-dtor` - Detects use-after-destroy bugs (with MSan)
- `-fno-common` - Prevents variables from being merged into common blocks, helping identify variable access issues

Note that some sanitizers are incompatible with each other. For example, you cannot use ASan and MSan together.

18.12 Code Coverage

18.12.1 Overview

Code coverage helps you understand which parts of the codebase are exercised by your tests. This project supports both local and containerized coverage workflows.

18.12.2 When is Coverage Available?

- Coverage is only available if you **build the project with GCC** and have both `gcov` and `gcovr` installed.
- If you use Clang, coverage reports will **not** be generated.
- Coverage is enabled automatically during the build if the requirements are met.

18.12.3 How to Generate Coverage Reports

1. Run the Test Suite*

First, run the test suite to generate coverage data. From your `build` directory:

```
./testsuite.sh
```

- This script sets up temporary PostgreSQL and pgagroal environments, runs all tests, and produces coverage data if enabled.

2. Generate Coverage Reports (Local)

After running the tests, generate the coverage reports:

```
# Run these commands from inside the build directory
mkdir -p ./coverage

gcovr -r ../src --object-directory . --html --html-details -o ./coverage/
   index.html
gcovr -r ../src --object-directory . > ./coverage/summary.txt
```

- The HTML report will be available at `build/coverage/index.html`
- A summary text report will be available at `build/coverage/summary.txt`

> **Note:** If the `coverage` directory does not exist, create it first using `mkdir -p ./coverage`.
>
> **Important:** `gcovr` only works with GCC builds.

3. Containerized Coverage (Optional)

If you have **Docker** or **Podman** installed, you can run tests and generate coverage in a container for a clean, isolated environment.

You have two options:

a. Using CTest

```
ctest -V
```

This will run all tests in a container if configured.

b. Using the Coverage Script

```
./coverage.sh
```

This script will: - Build and run the tests in a container - Generate coverage reports automatically in `build/coverage/` - Copy logs and coverage data back to your host

18.12.4 Summary Table

Task	Command(s)	Prerequisites
Run tests locally	`./testsuite.sh`	Built with GCC, PostgreSQL
Generate coverage locally	See commands above	`gcov`, `gcovr` installed
Run containerized tests	`ctest -V` or `./coverage.sh`	Docker or Podman installed

18.12.5 Notes

- Always run the coverage commands from the `build` directory.
- If coverage tools are not found, or the compiler is not GCC, coverage generation will be skipped and a message will be shown.
- You can always re-run the coverage commands manually if needed.

18.13 Event Loop

Here you find a concise developer-focused description of how the event loop implemented in `ev.c` and `ev.h` operates. This includes architecture, watcher mechanics, lifecycle, role separation, callback pipelines, and planned enhancements.

High-Level Architecture

The event loop sits in a continuous wait cycle, monitoring sources of events (I/O readiness, timer expirations, or delivered signals), dispatching the appropriate handler when an event occurs, and calling an user registered callback.

- Backends abstract the OS-specific wait or I/O mechanis used. The backend can be configured in the configuration file with the variable `ev_backend`.
- Watchers encapsulate interest in one type of event (I/O, signal, or periodic) and carry the callback and context required.
- Lifecycle routines manage setup, execution, interruption, and teardown of the loop.
- **Each process has its own event loop**: There is a **clear distinction** between **main** (accepting connections) and **worker** (handling established connections) event loops.

Data Structures

- `struct event_loop` centralizes all loop state, holding:
  - `running` flag governs the main loop.
  - `sigset` tracks which signals the loop intercepts.
  - An array of generic `event_watcher_t*` pointers represents active watchers.
  - Backend handles (`io_uring` ring, `epollfd`, or `kqueuefd`) interface directly with the kernel.
  - A scratch `buffer` or `io_uring` buffer ring is used to stage data transfers efficiently (the memory is defined elsewhere in pgagroal and used here as the buffer).

Watcher Types and Responsibilities

Every watcher embeds a small **common header** containing its type, enabling the loop to iterate over mixed watcher arrays.

1. **I/O Watchers** monitor one or two file descriptors.
   - *Main* watchers listen for new client connections and accept them.
   - *Worker* watchers handle serial request/response flows, blocking on receive then send.
2. **Signal Watchers** wrap POSIX signals into file descriptors. The loop unblocks these signals globally, then watches the FD for delivery events, invoking the registered callback.

3. **Periodic Watchers** fire at fixed millisecond intervals.

Event Loop Lifecycle

1. **Initialization** (`pgagroal_event_loop_init`):
2. **Running** (`pgagroal_event_loop_run`):
3. **Breaking** (`pgagroal_event_loop_break`):
4. **Destruction** (`pgagroal_event_loop_destroy`):
5. **Fork Handling** (`pgagroal_event_loop_fork`):

Main vs Worker I/O watchers

To simplify connection handling, the code forks a **Worker** process for each accepted client. Both processes run the same loop, but with different watchers registered:

- **Main Process**:
  1. Watches `listen_fd` for new connections.
  2. On accept, forks a Worker and continues listening.
- **Worker Process**:
  1. Registers I/O watchers on `rcv_fd` and `snd_fd`.
  2. Waits for `rcv_fd` to signal incoming data, then invokes a pipeline callback to process it.
  3. Sends responses on `snd_fd`.

Pipelines & Callback Flow

The loop's generic I/O `handler` delegates to a **pipeline** based on watcher type and context. A typical flow:

1. **I/O event**: `backend -> loop -> io_watcher.handler`
2. **Dispatch**: Handler inspects messages in buffer and selects the next pipeline stage function.
3. **Processing and Returning**: Pipeline stage validates the message payload and gets back to the loop.

This approach separates generic loop mechanics from application-specific message handling.

Enhancements

First, the main enhancement we could do is improve initial connection time. This could happen by initially caching the event loops beforehand and allowing for a connection to pick up one. Further examination of ftrace here is required.

Second, a series of compile-time flags mark areas for performance tuning. In my experience, none of these have been able to greatly improve performance (**haven't tested with iovecs**), but these may still require correct implementation and evaluation:

- **Zero Copy** (`MSG_ZEROCOPY` via io_uring) — reduce CPU overhead by skipping buffer copies.
- **Fast Poll** (`EPOLLET`) — edge-triggered epoll mode for high-throughput scenarios.
- **Huge Pages** (`IORING_SETUP_NO_MMAP`) — leverage large page mappings for buffer rings.
- **Multishot Recv** — one SQE to deliver multiple receive completions.
- **IOVecs** — scatter/gather I/O arrays for fewer system calls.

18.14 Core APIs

pgagroal offers data structures and APIs to help you write safer code and enable you to develop more advanced functionalities. Currently, we offer adaptive radix tree (ART), deque and JSON, which are all based on a universal value type system that help you manage the memory easily.

The document will mostly focus on design decisions, functionalities and things to be cautious about. It may offer some examples as to how to use the APIs.

18.14.1 Value

The `value` struct and its APIs are defined and implemented in value.h and value.c.

The `value` struct wraps the underlying data and manages its memory according to the type users specified. In some cases the data is stored inline, other times it stores a pointer to the actual memory. Most of the time the `value` struct is transparent to users. The most common use case would be that user put the data into some data structure such as a deque. The deque will internally wrap the data into a value object. When user reads the data, the deque will unwrap the value and return the internal data. An exception here is when you work with iterators, the iterator will return the value wrapper directly, which tells you the type of the value data. This allows you to store different types of value data into one data structure without worrying about losing the type info of the data when iterating over the structure.

When you free the deque, deque will automatically free up all the data stored within. In other words, you won't ever need to iterate over the deque and free all the stored data manually and explicitly.

The `value` struct can also print out the wrapped data according to its type. This is convenient for debugging and building output – since `deque`, `ART` and `JSON` are also value types, and their internal data are all wrapped in `value`, their content can be easily printed out.

Types We support the following value types:

type	type enum	free behavior (no-op if left blank)
`none`	`ValueNone`	
`int8_t`	`ValueInt8`	
`uint8_t`	`ValueUInt8`	
`int16_t`	`ValueInt16`	
`uint16_t`	`ValueUInt16`	

type	type enum	free behavior (no-op if left blank)
int32_t	ValueInt32	
uint32_t	ValueUInt32	
int64_t	ValueInt64	
uint64_t	ValueUInt64	
char	ValueChar	
bool	ValueBool	
char*	ValueString	free()
char*	ValueStringRef	
float	ValueFloat	
double	ValueDouble	
char*	ValueBASE64	free()
char*	ValueBASE64Ref	
struct json*	ValueJSON	pgagroal_json_destroy(), this will recursively destroy internal data
struct json*	ValueJSONRef	
struct deque*	ValueDeque	pgagroal_deque_destroy(), this will recursively destroy internal data
struct deque*	ValueDequeRef	
struct art *	ValueART	pgagroal_art_destroy(), this will recursively destroy internal data

type	type enum	free behavior (no-op if left blank)
`struct art *`	`ValueARTRef`	
`void*`	`ValueRef`	
`void*`	`ValueMem`	`free()`

You may have noticed that some types have corresponding `Ref` types. This is especially handy when you try to share data among multiple data structures – only one of them should be in charge of freeing up the value. The rest should only take the corresponding reference type to avoid double free.

There are cases where you try to put a pointer into the core data structure, but it's not any of the predefined types. In such cases, we offer a few options:

- If you want to free the pointed memory yourself, or it doesn't need to be freed, use `ValueRef`.
- If you just need to invoke a simple `free()`, use `ValueMem`.
- If you need to customize how to destroy the value, we offer you APIs to configure the behavior yourself, which will be illustrated below.

Note that the system does not enforce any kind of borrow checks or lifetime validation. It is still the programmers' responsibility to use the system correctly and ensure memory safe. But hopefully the system will make the burden a little lighter.

APIs

pgagroal_value_create

Create a value to wrap your data. Internally the value use a `uintptr_t` to hold your data in place or use it to represent a pointer, so simply cast your data into `uintptr_t` before passing it into the function (one exception is when you try to put in float or double, which requires extra work, see pgagroal_value_from_float/pgagroal_value_from_double for details). For `ValueString` or `ValueBASE64`, the value **makes a copy** of your string data. So if your string is malloced on heap, you still need to free it since what the value holds is a copy.

```
pgagroal_value_create(ValueString, (uintptr_t)str, &val);
// free the string if it's stored on heap
free(str);
```

pgagroal_value_create_with_config Create a value wrapper with `ValueRef` type and customized destroy and to-string callback. If you want to leave a callback as default, set the field to NULL.

You normally don't have to create a value yourself, but you will indirectly invoke it when you try to put data into a deque or ART with a customized configuration.

The callback definition is

```
typedef void (*data_destroy_cb)(uintptr_t data);
typedef char* (*data_to_string_cb)(uintptr_t data, int32_t format, char*
   tag, int indent);
```

pgagroal_value_to_string

This invokes the internal to-string callback and prints out the wrapped data content. You don't usually need to call this function yourself, as the nested core data structures will invoke this for you on each of its stored value.

For core data structure types, such as `deque`, `ART` or `JSON`, there are multiple supported types of format: * `FORMAT_JSON`: This prints the wrapped data in JSON format * `FORMAT_TEXT`: This prints the wrapped data in YAML-like format * `FORMAT_JSON_COMPACT`: This prints the wrapped data in JSON format, with all whitespaces omitted

Note that the format may also affect primitive types. For example, a string will be enclosed by `"` in JSON format, while in TEXT format, it will be printed as-is.

For `ValueMem` and `ValueRef`, the pointer to the memory will be printed.

pgagroal_value_data

Reader function to unwrap the data from the value wrapper. This is especially handy when you fetched the value with wrapper from the iterator.

pgagroal_value_type

Reader function to get the type from the value wrapper. The function returns `ValueNone` if the input is NULL.

pgagroal_value_destroy

Destroy a value, this invokes the destroy callback to destroy the wrapped data.

pgagroal_value_to_float/pgagroal_value_to_double

Use the corresponding function to cast the raw data into the float or double you had wrapped inside the value.

Float and double types are stored in place inside the `uintptr_t` data field. But since C cannot automatically cast a `uintptr_t` to float or double correctly, – it doesn't interpret the bit representation as-is – we have to resort to some union magic to enforce the casting.

pgagroal_value_from_float/pgagroal_value_from_double

For the same reason mentioned above, use the corresponding function to cast the float or double you try to put inside the value wrapper to raw data.

```
pgagroal_value_create(ValueFloat, pgagroal_value_from_float(float_val), &
   val);
```

pgagroal_value_to_ref

Return the corresponding reference type. Input `ValueJSON` will give you `ValueJSONRef`. For in-place types such as `ValueInt8`, or if the type is already the reference type, the same type will be returned.

18.14.2 Deque

The deque is defined and implemented in deque.h and deque.c. The deque is built upon the value system, so it can automatically destroy the internal items when it gets destroyed.

You can specify an optional tag for each deque node, so that you can sort of use it as a key-value map. However, since the introduction of ART and json, this isn't the recommended usage anymore.

APIs

pgagroal_deque_create

Create a deque. If thread safe is set, a global read/write lock will be acquired before you try to write to deque or read it. The deque should still be used with cautious even with thread safe enabled – it does not guard against the value you have read out. So if you had stored a pointer, deque will not protect the pointed memory from being modified by another thread.

pgagroal_deque_add

Add a value to the deque's tail. You need to cast the value to `uintptr_t` since it creates a value wrapper underneath. Again, for float and double you need to use the corresponding type casting function (`pgagroal_value_from_float` / `pgagroal_value_from_double`). The function acquires write lock if thread safe is enabled.

The time complexity for adding a node is O(1).

pgagroal_deque_add_with_config

Add data with type `ValueRef` and customized to-string/destroy callback into the deque. The function acquires write lock if thread safe is enabled.

```
static void
rfile_destroy_cb(uintptr_t data)
{
   rfile_destroy((struct rfile*) data);
}
```

```
static void
add_rfile()
{
struct deque* sources;
struct rfile* latest_source;
struct value_config rfile_config = {.destroy_data = rfile_destroy_cb, .
   to_string = NULL};
...

pgagroal_deque_add_with_config(sources, NULL, (uintptr_t)latest_source, &
   rfile_config);
}
```

pgagroal_deque_poll

Retrieve value and remove the node from the deque's head. If the node has tag, you can optionally read it out. The function transfers value ownership, so you will be responsible to free the value if it was copied into the node when you put it in. The function acquires read lock if thread safe is enabled.

The time complexity for polling a node is O(1).

```
pgagroal_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* tag = NULL;
char* value = (char*)pgagroal_deque_poll(deque, &tag);

printf("%s, %s!\n", tag, value) // "Hello, world!"

// remember to free them!
free(tag);
free(value);
```

```
// if you don't care about tag
pgagroal_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* value = (char*)pgagroal_deque_poll(deque, NULL);

printf("%s!\n", value) // "world!"

// remember to free it!
free(value);
```

pgagroal_deque_poll_last

Retrieve value and remove the node from the deque's tail. If the node has tag, you can optionally read it out. The function transfers value ownership, so you will be responsible to free the value if it was copied into the node when you put it in. The function acquires read lock if thread safe is enabled.

The time complexity for polling a node is O(1).

```
pgagroal_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* tag = NULL;
```

```
char* value = (char*)pgagroal_deque_poll_last(deque, &tag);

printf("%s, %s!\n", tag, value) // "Hello, world!"

// remember to free them!
free(tag);
free(value);
```

```
// if you don't care about tag
pgagroal_deque_add(deque, "Hello", (uintptr_t)"world", ValueString);
char* value = (char*)pgagroal_deque_poll_last(deque, NULL);

printf("%s!\n", value) // "world!"

// remember to free it!
free(value);
```

pgagroal_deque_peek

Retrieve value without removing the node from deque's head. The function acquires read lock if thread safe is enabled.

The time complexity for peeking a node is O(1).

pgagroal_deque_peek_last

Retrieve value without removing the node from deque's tail. The function acquires read lock if thread safe is enabled.

The time complexity for peeking a node is O(1).

pgagroal_deque_iterator_create

Create a deque iterator, note that iterator is **NOT** thread safe

pgagroal_deque_iterator_destroy

Destroy a deque iterator

pgagroal_deque_iterator_next

Advance the iterator to the next value. You will need to call it before reading the first item. The function is a no-op if it reaches the end and will return false.

pgagroal_deque_iterator_has_next

Check if iterator has next value without advancing it.

pgagroal_deque_iterator_remove

Remove the current node the iterator is pointing to. Then the iterator will fall back to the previous node.

For example, for a deque `a -> b -> c`, after removing node `b`, iterator will point to `a`, then calling `pgagroal_deque_iterator_next` will advance the iterator to `c`. If node `a` is removed instead, iterator will point to the internal dummy head node.

```
// remove nodes without a tag
pgagroal_deque_iterator_create(deque, &iter);
while (pgagroal_deque_iterator_next(iter)) {
    if (iter->tag == NULL) {
        pgagroal_deque_iterator_remove(iter);
    }
    else {
        printf("%s: %s\n", iter->tag, (char*)pgagroal_value_data(iter->
           value));
    }
}
pgagroal_deque_iterator_destroy(iter);
```

pgagroal_deque_size Get the current deque size, the function acquires the read lock

pgagroal_deque_empty

Check if the deque is empty

pgagroal_deque_to_string

Convert the deque to string of the specified format.

pgagroal_deque_list

Log the deque content in logs. This only works in TRACE log level.

pgagroal_deque_sort

Merge sort the deque. The time complexity is O(log(n)).

pgagroal_deque_get

Get the data with a specific tag from the deque.

The time complexity for getting a node is O(n).

pgagroal_deque_exists

Check if a tag exists in deque.

pgagroal_deque_remove

Remove all the nodes in the deque that have the given tag.

pgagroal_deque_clear

Remove all the nodes in the deque.

pgagroal_deque_set_thread_safe

Set the deque to be thread safe.

18.14.3 Adaptive Radix Tree (ART)

ART shares similar ideas as trie. But it is very space efficient by adopting techniques such as adaptive node size, path compression and lazy expansion. The time complexity of inserting, deleting or searching a key in an ART is always O(k) where the k is the length of the key. And since most of the time our key type is string, ART can be used as **an ideal key-value map** with much less space overhead than hashmap.

ART is defined and implemented in art.h and art.c.

APIs

pgagroal_art_create

Create an adaptive radix tree

pgagroal_art_insert

Insert a key value pair into the ART. Likewise, the ART tree wraps the data in value internally. So you need to cast the value to `uintptr_t`. If the key already exists, the previous value will be destroyed and replaced by the new value.

pgagroal_art_insert_with_config

Insert a key value pair with a customized configuration. The idea and usage is identical to `pgagroal_deque_add_with_config`.

pgagroal_art_contains_key

Check if a key exists in ART.

pgagroal_art_search

Search a value inside the ART by its key. The ART unwraps the value and return the raw data. If key is not found, it returns 0. So if you need to tell whether it returns a zero value or the key does not exist, use `pgagroal_art_contains_key`.

pgagroal_art_search_typed

Search a value inside the ART by its key. The ART unwraps the value and return the raw data. It also returns the value type through the output `type` parameter. If key is not found, it returns 0, and the type is set to `ValueNone`. So you can also use it to tell if a value exists.

pgagroal_art_delete

Delete a key from ART. Note that the function returns success(i.e. 0) even if the key does not exist.

pgagroal_art_clear

Removes all the key value pairs in the ART tree.

pgagroal_art_to_string

Convert an ART to string. The function uses an internal iterator function which iterates the tree using DFS. So unlike the iterator, this traverses and prints out keys by lexicographical order.

pgagroal_art_destroy

Destroy an ART.

pgagroal_art_iterator_create

Create an ART iterator, the iterator iterates the tree using BFS, which means it won't traverse the keys by lexicographical order.

pgagroal_art_iterator_destroy

Destroy an ART iterator. This will recursively destroy all of its key value entries.

pgagroal_art_iterator_remove

Remove the key value pair the iterator points to. Note that currently the function just invokes `pgagroal_art_delete()` with the current key. Since there's no rebalance mechanism in ART, it shouldn't affect the subsequent iteration. But still use with caution, as this is not thoroughly tested.

pgagroal_art_iterator_next

Advance an ART iterator. You need to call this function before inspecting the first entry. If there are no more entries, the function is a no-op and will return false.

pgagroal_art_iterator_has_next

Check if the iterator has next value without advancing it.

```
pgagroal_art_iterator_create(t, &iter);
while (pgagroal_art_iterator_next(iter)) {
    printf("%s: %s\n", iter->key, (char*)pgagroal_value_data(iter->value))
        ;
}
pgagroal_art_iterator_destroy(iter);
```

18.14.4 JSON

JSON is essentially built upon deque and ART. Find its definition and implementation in json.h and json.c.

APIs

pgagroal_json_create

Create a JSON object. Note that the json could be an array (`JSONArray`) or key value pairs (`JSONItem`). We don't specify the JSON type on creation. The json object will decide by itself based on the subsequent API invocation.

pgagroal_json_destroy

Destroy a JSON object

pgagroal_json_put

Put a key value pair into the json object. This function invokes `pgagroal_art_insert` underneath so it will override the old value if key already exists. Also when invoked for the first time, the function sets the JSON object to `JSONItem`, which will reject `pgagroal_json_append` from then on. Note that unlike ART, JSON only takes certain types of value. See JSON introduction for details.

pgagroal_json_append

Append a value entry to the json object. When invoked for the first time, the function sets the JSON object to `JSONArray`, which will reject `pgagroal_json_put` from then on.

pgagroal_json_remove

Remove a key and destroy the associated value within the json item. If the key does not exist or the json object is an array, the function will be no-op. If the JSON item becomes empty after removal, it will fall back to undefined status, and you can turn it into an array by appending entries to it.

pgagroal_json_clear

For `JSONArray`, the function removes all entries. For `JSONItem`, the funtion removes all key value pairs. The JSON object will fall back to undefined status.

pgagroal_json_get

Get and unwrap the value data from a JSON item. If the JSON object is an array, the function returns 0.

pgagroal_json_get_typed

Get and unwrap the value data from a JSON item, also returns the value type through output `type` parameter. If the JSON object is an array, the function returns 0. If the key is not found, the function sets `type` to `ValueNone`. So you can also use it to check if a key exists.

pgagroal_json_contains_key

Check if the JSON item contains a specific key. It always returns false if the object is an array.

pgagroal_json_array_length

Get the length of a JSON array

pgagroal_json_iterator_create

Create a JSON iterator. For JSON array, it creates an internal deque iterator. For JSON item, it creates an internal ART iterator. You can read the value or the array entry from `value` field. And the `key` field is ignored when the object is an array.

pgagroal_json_iterator_next

Advance to the next entry or key value pairs. You need to call this before accessing the first entry or kv pair.

pgagroal_json_iterator_has_next

Check if the object has the next entry or key value pair.

pgagroal_json_iterator_destroy

Destroy the JSON iterator.

pgagroal_json_parse_string

Parse a JSON string into a JSON object.

pgagroal_json_clone

Clone a JSON object. This works by converting the object to string and parse it back to another object. So the value type could be a little different. For example, an `int8` value will be parsed into an `int64` value.

pgagroal_json_to_string

Convert the JSON object to string.

pgagroal_json_print

A convenient wrapper to quickly print out the JSON object.

pgagroal_json_read_file

Read the JSON file and parse it into the JSON object.

pgagroal_json_write_file

Convert the JSON to string and write it to a JSON file.

18.15 Test Suite

18.15.1 Overview

This document explains how to run the pgagroal test suite, generate code coverage, and use containerized testing. All testing is now performed using the `check.sh` script with containerized PostgreSQL (recommended and default for all development and CI).

Running Specific Test Cases or Suites

You can run a specific test case or suite using the following environment variables:

- `CK_RUN_CASE=<test_case_name> ./check.sh` — runs a single test case
- `CK_RUN_SUITE=<test_suite_name> ./check.sh` — runs a single test suite

Alternatively, you can export the environment variable before running the script:

```
export CK_RUN_CASE=<test_case_name>
./check.sh
```

The environment variables will be automatically unset when the test is finished or aborted.

18.15.2 Containerized

The `check.sh` script is the main and recommended way to run the pgagroal test suite. It works on any system with Docker or Podman (Linux, macOS, FreeBSD, Windows/WSL2). It automatically builds a PostgreSQL 17 container, sets up the test environment, runs all tests, and generates coverage reports and logs. No local PostgreSQL installation is required.

Key Features

- **No local PostgreSQL required**: Uses Docker/Podman containers
- **Consistent environment**: Same PostgreSQL version (17) across all systems
- **Automatic cleanup**: Containers are removed after tests
- **Integrated coverage**: Coverage reports generated automatically
- **Isolated testing**: No interference with local PostgreSQL installations
- **Multiple configurations**: Supports running tests on multiple pgagroal configurations
- **Easy setup**: `./check.sh setup` installs all dependencies and builds the PostgreSQL image
- **Flexible CI support**: Used in CI for Linux, and will be used for all platforms after migration

Usage

```
./check.sh [sub-command]
```

Subcommands:

- `setup` Install dependencies and build PostgreSQL image (one-time setup)
- `clean` Clean up test suite environment and remove PostgreSQL image
- `run-configs` Run the testsuite on multiple pgagroal configurations (containerized)
- `ci` Run in CI mode (local PostgreSQL, no container)
- `run-configs-ci` Run multiple configuration tests using local PostgreSQL (like ci + run-configs)
- `ci-nonbuild` Run in CI mode (local PostgreSQL, skip build step)
- `run-configs-ci-nonbuild` Run multiple configuration tests using local PostgreSQL, skip build step
- (no sub-command) Default: run all tests in containerized mode

> **For local development, use only the `run-configs` and default (no sub-command) modes. Other modes (`ci`, `run-configs-ci`, etc.) are intended for CI and may interfere with your local PostgreSQL setup if used locally.**

Artifacts and Logs

After running containerized tests, you will find:

- Test logs: `/tmp/pgagroal-test/log/`
- PostgreSQL logs: `/tmp/pgagroal-test/pg_log/`
- Coverage reports: `/tmp/pgagroal-test/coverage/`

Adding New Test Cases

- Add new `.c` and `.h` files in `test/testcases/`.
- Register your test suite in `test/testcases/runner.c`.
- Add your test source to `test/CMakeLists.txt`:

```
set(SOURCES
  testcases/common.c
  testcases/your_new_test.c
  testcases/runner.c
)
```

Prerequisites

- **Docker or Podman** installed and running
- The `check` library installed for C unit tests
- **LLVM/clang** and **llvm-cov**/**llvm-profdata** installed (for coverage reports)

> **Note:** The `check.sh` script always builds the project with Clang in Debug mode for coverage and testability.

Notes

- The containerized approach automatically handles cleanup on exit.
- Use `./check.sh clean` to manually remove containers and test data.
- PostgreSQL container logs are available with debug5 level for troubleshooting.
- The script automatically detects and uses either Docker or Podman.
- It is recommended to **ALWAYS** run tests before raising a PR.
- Coverage reports are generated using LLVM tooling (clang, llvm-cov, llvm-profdata).
- For local development, use only the `run-configs` and default (no sub-command) modes. Other modes (`ci`, `run-configs-ci`, etc.) are intended for CI and may interfere with your local PostgreSQL setup if used locally.

18.16 Distribution-Specific Installation

This chapter provides installation instructions for different operating systems and distributions.

18.16.1 Dependencies

pgagroal requires the following dependencies:

- a C compiler like gcc 8+ (C17) or clang 8+
- cmake
- GNU make or BSD `make`
- libev
- OpenSSL
- rst2man
- libatomic
- Doxygen
- pdflatex
- zlib
- zstd
- lz4
- bzip2
- binutils
- on Linux platforms, there is also the need for
  - systemd

18.16.2 Rocky Linux / RHEL

All the dependencies can be installed via `dnf(8)` as follows:

```
dnf install git gcc cmake make    \
            libev libev-devel     \
            openssl openssl-devel \
            systemd systemd-devel \
            python3-docutils      \
            libatomic             \
            zlib zlib-devel       \
            libzstd libzstd-devel \
            lz4 lz4-devel         \
            bzip2 bzip2-devel     \
            binutils              \
            clang clang-analyzer clang-tools-extra
```

Please note that, on Rocky Linux, in order to install the `python3-docutils` package (that provides `rst2man` executable), you need to enable the `crb` repository:

```
dnf config-manager --set-enabled crb
```

18.16.3 FreeBSD

All the dependencies can be installed via `pkg(8)` as follows:

```
pkg install cmake           \
            libev libevent  \
            py311-docutils  \
            lzlib           \
            liblz4          \
            lbizp2          \
            texlive-formats \
            binutils
```

18.16.4 Fedora

For Fedora systems, use:

```
dnf install git gcc cmake make liburing liburing-devel openssl openssl-
   devel systemd systemd-devel python3-docutils libatomic zlib zlib-devel
   libzstd libzstd-devel lz4 lz4-devel bzip2 bzip2-devel libasan libasan-
   static binutils clang clang-analyzer clang-tools-extra
```

18.16.5 Ubuntu / Debian

For Ubuntu and Debian systems:

```
apt-get update
apt-get install build-essential cmake libev-dev libssl-dev libsystemd-dev
   python3-docutils libatomic1 zlib1g-dev libzstd-dev liblz4-dev libbz2-
   dev binutils
```

18.16.6 macOS

For macOS using Homebrew:

```
brew install cmake libev openssl@3 docutils zlib zstd lz4 bzip2
```

Note: On macOS, you may need to set additional environment variables for OpenSSL:

```
export OPENSSL_ROOT_DIR=$(brew --prefix openssl@3)
export PKG_CONFIG_PATH="$OPENSSL_ROOT_DIR/lib/pkgconfig:$PKG_CONFIG_PATH"
```

18.16.7 Building from Source

After installing dependencies, build pgagroal:

```
git clone https://github.com/pgagroal/pgagroal.git
cd pgagroal
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
sudo make install
```

18.16.8 Platform-Specific Notes

18.16.8.1 Linux

- Ensure `liburing` is available for optimal I/O performance
- Configure systemd service files for production deployments
- Consider using huge pages for better memory performance

18.16.8.2 FreeBSD

- Use the ports system for more control over build options
- Configure appropriate kernel parameters for network performance
- Consider using jails for isolation

18.16.8.3 macOS

- Some features may have limited support compared to Linux
- Use Homebrew for dependency management
- Consider using Docker for consistent environments

18.16.9 Troubleshooting

18.16.9.1 Common Issues **Missing liburing on older systems:**

```
# Install liburing from source if not available in package manager
git clone https://github.com/axboe/liburing.git
cd liburing
make
sudo make install
```

OpenSSL version conflicts:

```
# Specify OpenSSL path explicitly
cmake -DOPENSSL_ROOT_DIR=/usr/local/ssl ..
```

Permission issues:

```
# Ensure proper permissions for installation
sudo chown -R $(whoami) /usr/local/
```

18.16.9.2 Build Flags

For debug builds:

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

For release builds with optimizations:

```
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_C_FLAGS="-O3 -march=native" ..
```

19 Acknowledgement

19.1 Authors

pgagroal was created by the following authors:

```
Jesper Pedersen <jesperpedersen.db@gmail.com>
David Fetter <david@fetter.org>
Will Leinweber <will@bitfission.com>
Junduo Dong <andj4cn@gmail.com>
Luca Ferrari <fluca1978@gmail.com>
Nikita Bugrovsky <nbugrovs@redhat.com>
Lawrence Wu <lawrence910426@gmail.com>
Yongting You <2010youy01@gmail.com>
Ashutosh Sharma <ash2003sharma@gmail.com>
Henrique de Carvalho <decarv.henrique@gmail.com>
Yihe Lu <t1t4m1un@gmail.com>
Eugenio Gigante <giganteeugenio2@gmail.com>
Mohanad Khaled <mohanadkhaled87@gmail.com>
Haoran Zhang <andrewzhr9911@gmail.com>
Christian Englert <code@c.roboticbrain.de>
Georg Pfuetzenreuter <georg.pfuetzenreuter@suse.com>
Tejas Tyagi <tejastyagi.tt@gmail.com>
Aryan Arora <aryanarora.w1@gmail.com>
Sangkeun J.C. Kim <jchrys@me.com>
Arshdeep Singh <balarsh535@gmail.com>
Vanes Angelo <k124k3n@gmail.com>
Bassam Adnan <mailbassam@gmail.com>
Sara Nabih <nabihsara8@gmail.com>
Mahmoud Hamdy (TutTrue) <mahmoud.hamdy5113@gmail.com>
Ankush Mondal <mondalankush9851@gmail.com>
Shashank Singh <shashanksgh3@gmail.com>
```

19.2 Committers

```
Jesper Pedersen <jesperpedersen.db@gmail.com>
Luca Ferrari <fluca1978@gmail.com>
```

19.3 Contributing

Contributions to **pgagroal** are managed on GitHub

- Ask a question
- Raise an issue
- Feature request

- Code submission

Contributions are most welcome!

Please, consult our Code of Conduct policies for interacting in our community.

Consider giving the project a star on GitHub if you find it useful. And, feel free to follow the project on Twitter as well.

20 License

```
Copyright (C) 2025 The pgagroal community

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above
copyright notice, this list of conditions and the following
disclaimer in the documentation and/or other materials provided
with the distribution.

3. Neither the name of the copyright holder nor the names of
its contributors may be used to endorse or promote products
derived from this software without specific prior written
permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF SUCH DAMAGE.
```

BSD-3-Clause

20.1 libart

Our adaptive radix tree (ART) implementation is based on The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases and libart which has a 3-BSD license as

```
Copyright (c) 2012, Armon Dadgar
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
```

```
* Redistributions of source code must retain the above copyright
  notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright
  notice, this list of conditions and the following disclaimer in the
  documentation and/or other materials provided with the distribution.

* Neither the name of the organization nor the
  names of its contributors may be used to endorse or promote products
  derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ARMON DADGAR
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY,
OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE
OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

21 References

This section contains reference links used throughout the manual.

21.1 External Links

- **pgagroal**: https://github.com/pgagroal/pgagroal
- **PostgreSQL**: https://www.postgresql.org
- **Rocky Linux**: https://www.rockylinux.org
- **Fedora**: https://getfedora.org/
- **RHEL**: https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux
- **AppStream**: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/installing_mana space_components/using-appstream_using-appstream
- **FreeBSD**: https://www.freebsd.org/
- **OpenBSD**: http://www.openbsd.org/
- **GCC**: https://gcc.gnu.org
- **CMake**: https://cmake.org
- **Make**: https://www.gnu.org/software/make/
- **libev**: http://software.schmorp.de/pkg/libev.html
- **OpenSSL**: http://www.openssl.org/
- **systemd**: https://www.freedesktop.org/wiki/Software/systemd/
- **rst2man**: https://docutils.sourceforge.io/
- **Pandoc**: https://pandoc.org/
- **Pandoc LaTeX Template**: https://github.com/Wandmalfarbe/pandoc-latex-template
- **TeX Live**: https://www.tug.org/texlive/
- **Clang**: https://clang.llvm.org/
- **Git Squash Guide**: https://www.git-tower.com/learn/git/faq/git-squash
- **ProGit Book**: https://github.com/progit/progit2/releases
- **Prometheus**: https://prometheus.io/
- **Wireshark**: https://www.wireshark.org/
- **pgprtdbg**: https://github.com/jesperpedersen/pgprtdbg
- **ART Paper**: http://www-db.in.tum.de/~leis/papers/ART.pdf
- **libart**: https://github.com/armon/libart

21.2 Documentation

- **Manual (English)**: https://github.com/pgagroal/pgagroal/tree/master/doc/manual/en

- **Main Documentation Folder**: https://github.com/pgagroal/pgagroal/tree/master/doc
- **RPM Documentation**: https://github.com/pgagroal/pgagroal/blob/master/doc/RPM.md
- **Configuration Documentation**: https://github.com/pgagroal/pgagroal/blob/master/doc/CONFIGURATION.md

21.3 Configuration Examples

- **Configuration Examples Folder**: https://github.com/pgagroal/pgagroal/tree/master/doc/etc
- **Sample Configuration**: https://github.com/pgagroal/pgagroal/blob/master/doc/etc/pgagroal.conf
- **HBA Configuration**: https://github.com/pgagroal/pgagroal/blob/master/doc/etc/pgagroal_hba.conf
- **Vault Configuration**: https://github.com/pgagroal/pgagroal/blob/master/doc/etc/pgagroal_vault.conf

21.4 Source Code

21.4.1 Main Source Files

- **main.c**: https://github.com/pgagroal/pgagroal/blob/master/src/main.c
- **cli.c**: https://github.com/pgagroal/pgagroal/blob/master/src/cli.c
- **admin.c**: https://github.com/pgagroal/pgagroal/blob/master/src/admin.c
- **vault.c**: https://github.com/pgagroal/pgagroal/blob/master/src/vault.c

21.4.2 Include Files

- **shmem.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/shmem.h
- **pgagroal.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/pgagroal.h
- **message.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/message.h
- **network.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/network.h
- **memory.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/memory.h
- **management.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/management.h
- **remote.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/remote.h
- **prometheus.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/prometheus.h
- **logging.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/logging.h
- **value.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/value.h
- **deque.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/deque.h
- **art.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/art.h
- **json.h**: https://github.com/pgagroal/pgagroal/blob/master/src/include/json.h

21.4.3 Library Implementation Files

- **message.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/message.c
- **network.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/network.c
- **memory.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/memory.c
- **remote.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/remote.c
- **prometheus.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/prometheus.c
- **logging.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/logging.c
- **value.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/value.c
- **deque.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/deque.c
- **art.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/art.c
- **json.c**: https://github.com/pgagroal/pgagroal/blob/master/src/libpgagroal/json.c

21.5 Contributing

- **Ask Questions**: https://github.com/pgagroal/pgagroal/discussions
- **Report Issues**: https://github.com/pgagroal/pgagroal/issues
- **Feature Requests**: https://github.com/pgagroal/pgagroal/issues
- **Submit Code**: https://github.com/pgagroal/pgagroal/pulls
- **Code of Conduct**: https://github.com/pgagroal/pgagroal/blob/master/CODE_OF_CONDUCT.md
- **Star the Project**: https://github.com/pgagroal/pgagroal/stargazers
- **Follow on Twitter**: https://twitter.com/pgagroal/
- **BSD-3-Clause License**: https://opensource.org/licenses/BSD-3-Clause